Progression Heuristics for Planning with Probabilistic LTL Constraints

Ian Mallett

A thesis submitted for the degree of
Bachelor of Advanced Computing
The Australian National University

October 2019

© Ian Mallett 2019

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

Ian Mallett
28 October 2019

To my computer, which has survived the last 5 years through numerous break downs, and has finally reached retirement alive.

Acknowledgments

I can't express enough thanks to my supervisors Sylvie Thiébaux and Felipe Trevizan, whose guidance, ideas and patience is the only reason I ever completed this project. Their dedication to helping me improve my thesis has made the complexities of my project infinitely more approachable, and their passion for and extensive knowledge in this area is evident in every discussion we have.

I have gratitude towards my peers, who have inspired me, encouraged me, and worked through the herculean task of honours together with me. The regulars in the honours space of the Hanna Neumann building have provided plenty of pleasant distractions. Of particular note is Chamin Hewa Koneputugodage who lent me his desk, his computer and a listening ear for the many times I want to talk through a difficulty in my project.

I would like to thank Yiping Su also for providing a listening ear for my thesis and my mental wellbeing. She has being eternally patient and has constantly supported me when I am stressed, tired or snowed over with work.

My friends have provided me much needed breaks every weekend by sating my addiction to board games, and providing the relaxation that comes with friendship. Their encouragement and advice from their own honours projects has inspired me and helped me get to where I am now.

Finally, I have eternal gratitude to my family, who have raised me to love solving problems and learning new things. They have looked after my in every way, and cheered me on throughout my time at ANU and especially throughout the thesis. Thank you so much.

Abstract

In the field of planning, Stochastic Shortest Path problems (SSP) are a standard model for problems where the aim is optimise the expected cost of reaching some goal, using actions that have stochastic effects. Finding an optimal policy which reaches the goal, however, is not enough. It is desirable for agents to comply with safety considerations. These are modelled as probabilistic linear temporal logic (PLTL) constraints. Solving SSPs constrained by PLTL constraints is an active area of research in planning and formal methods.

Representing the state of the SSP and the state of each PLTL constraint (i.e., how close to being satisfied the constraint is) results in a prohibitively large combined state space. Recent research has applied heuristic search to this problem, a technique which can find a policy without enumerating the complete state space, but the heuristics used focus on a particular representation of the state of the PLTL constraints.

This thesis presents a novel heuristic for SSPs constrained by multiple PLTL constraints using a different PLTL state representation, *formula progression*. This thesis presents the first heuristic that uses formula progression as the representation for PLTL constraints. This heuristic combines two relaxations: *projection* that relaxes the dynamics of the SSP; and *decomposition* that relaxes the dynamics of the PLTL constraint.

Specifically, this novel heuristic constructs several complementary relaxations (called projections) of the underlying SSP, along with PLTL constraints adjusted appropriately for these projections. Each projection is converted into a novel planning problem called a Concurrent Constrained SSP, which relaxes the problem of satisfying a large PLTL constraint by instead having multiple agents simultaneously satisfy smaller PLTL constraints. This conversion decomposes logic statements into smaller ones based on a standard transformation of linear temporal logic.

This heuristic is integrated with the state-of-the-art heuristic planner and outperforms both existing heuristics as well as other algorithms for SSPs constrained by multiple PLTL constraints. The proposed heuristic scales better in most problems, showing the promise of progression based approaches.

Contents

Acknowledgments **vii**

Abstract **ix**

1 Introduction **1**
1.1 Overview of the Research Problem . . . 1
1.1.1 Probabilistic Planning . . . 1
1.1.2 Linear Temporal Logic . . . 2
1.1.3 Applying LTL in Probabilistic Planning . . . 3
1.1.4 Heuristics . . . 3
1.2 Contributions . . . 4
1.2.1 Projection . . . 4
1.2.2 Decomposition . . . 5
1.3 Thesis Outline . . . 6

2 Background **7**
2.1 Paths . . . 7
2.2 Linear Temporal Logic . . . 8
2.2.1 Syntax and Semantics . . . 8
2.2.2 Infinite Extension . . . 10
2.2.3 Identities . . . 11
2.2.4 Normal Forms . . . 12
2.2.5 Progression . . . 14
2.3 Probabilistic Planning . . . 17
2.3.1 Stochastic Shortest Path Problems . . . 17
2.3.2 Representations . . . 22
2.3.3 Heuristic Search . . . 23
2.4 MO-PLTL Constrained Planning . . . 26
2.5 Summary . . . 28

3 Related Work **31**
3.1 LTL In Planning . . . 31
3.1.1 Control Knowledge . . . 31
3.1.2 LTL Compilation . . . 32
3.2 LTL Strategy Synthesis . . . 34
3.2.1 Planning as Strategy Synthesis . . . 34
3.2.2 PRISM Model Checker . . . 35

3.3 PLTL-dual 36
3.3.1 Algorithm 36
3.3.2 Heuristics 37
3.4 Summary 38

4 LTL Projection **41**
4.1 Proposition Independence Assumption 41
4.2 Proposition Assignment 42
4.3 Choosing Variables 44
4.4 Summary 46

5 Probability Estimation by Decomposition **49**
5.1 LTL Decomposition 49
5.2 Concurrent Constrained SSPs 51
5.2.1 State Redistribution 52
5.2.2 CC-SSP Definition 54
5.2.3 Formal Concepts for CC-SSPs 55
5.2.4 CC-SSPs as Heuristic Estimation 59
5.3 LP Formulation 60
5.3.1 Occupation Measures 60
5.3.2 Linear Program 61
5.4 Limitations 64
5.4.1 Innate Limitations of Decomposition 64
5.4.2 Loops 66
5.5 Summary 69

6 Decomposition Heuristic for Planning **71**
6.1 Interfacing with PLTL-dual 71
6.1.1 Weighted Initial States 71
6.1.2 Tying Constraints 72
6.2 Tracing Accepting Flow 74
6.2.1 Determinisation 75
6.2.2 Upper Bounding 76
6.2.3 Spontaneous Flow Generation 76
6.2.4 Tying Constraints 77
6.3 LP Formulation 77
6.3.1 Linear Program 78
6.4 Limitations 81
6.5 Summary 82

7 Experiments **85**
7.1 Setup 85
7.1.1 Algorithms 85
7.1.2 Domains 86
7.2 Results 89

7.3 Discussion 89
7.3.1 Wall-e Domain 91
7.3.2 Factory Domain 91
7.3.3 Priority Search Domain 92
7.3.4 Conclusions 94
7.4 Summary 94

8 Conclusion **95**
8.1 Future Work 96
8.1.1 Work in this Thesis 96
8.1.2 Future Related Work 97

Appendices **99**

A Further Results **99**

List of Figures

2.1 Temporal Operators 10
2.2 Heuristic Search 24

5.1 Venn Diagrams for Inequalities 50
5.2 Agent redistribution 53
5.3 A run of a CC-SSP 56
5.4 Flow Redistribution 62
5.5 Trivial SSPs with significant overestimates 65
5.6 Loop Abuse: ‘and’ case 67
5.7 Loop Abuse: ‘or’ case 68
5.8 Spontaneous Flow Generation 69

6.1 Invalidity of Tying Constraints 73
6.2 Accepting Flow Example 75
6.3 Traced Decomposition Heuristic 80
6.4 Spontaneous Flow Generation For Accepting Flow 82

7.1 Wall-e and Factory Graphs 90

List of Tables

7.1 Time in seconds to complete each parametrisation of the priority search domain, aggregated over 10 random problems. 90

A.1 Time in seconds to complete each Wall-e instance. 99
A.2 States expanded for each problem in the Wall-e domain. 100
A.3 Aggregated number of states to complete each parametrisation of the priority search domain. 100
A.4 Time in seconds to complete each factory instance. 101
A.5 States expanded for each problem in the factory domain. 102

Notation

The symbols and operators used throughout this thesis are listed below for quick reference. There is an extraordinary amount of notation in this thesis, but it exists to simplify and abstract what is a very complicated series of constructions. We have done our best to make the notation clear and intuitive, e.g., note the similarities between symbols with similar meanings. However this may have made following the notation as much harder for some readers as it makes it easier for others. It is recommended (especially when reading chapters 5 and 6) that this list is kept on hand.

CC-SSP Representation

$\mathcal{A}_i$	The actions taken at the ith step by agents that take at least i steps in a CC-SSP.
$\tilde{\mathrm{S}}^{+}$	The set of incomplete paths through a CC-SSP.
$\hat{\mathrm{G}}$	Augmented goal states of a CC-SSP, where the LTL is accepted.
$\hat{\mathrm{F}}$	Absorbing augmented states in a CC-SSP.
$\hat{\mathrm{S}}$	A set of augmented states forming the search space of a CC-SSP.
$\tilde{\mathrm{S}}^{+\hat{\mathrm{F}}}$	The set of complete paths through a CC-SSP.
$\hat{\mathrm{S}}_{\mathrm{init}}$	The set of states which could be an initial state when computing $h^{\mathrm{dec}}_{\mathcal{S}}$ or its variants for PLTL-dual.
$\tilde{\mathrm{S}}$	The X-literal states in a CC-SSP.
$\tilde{\mathrm{S}}^{+\hat{\mathrm{G}}}$	The set of complete paths through a CC-SSP terminated by a goal.
$\Pi_{\mathcal{C}}$	The set of all bounded policies for $\mathcal{C}$
R	A run of a CC-SSP.
$\mathcal{C}$	A Concurrent Constrained SSP.
$\mathcal{P}_i$	A set of paths representing the first i steps of each agent in a CC-SSP.
$\mathcal{P}_{i+1,p}$	The paths in $\mathcal{P}_{i+1}$ prefixed by the path p.
$\langle s, \Psi \rangle$	An augmented state.
$\langle s, \phi \rangle$	An X-literal state, being the pair of an SSP state and an X-literal.

$\mathcal{S}_{\mathcal{C}}$	The SSP underlying a CC-SSP.
$\mathbf{P}_{s,\Psi}$	In PLTL-dual, the expected number of agents in the known state space reaching a state that maps to $\langle s, \Psi \rangle$ in a CC-SSP.

Linear Program Functions

$\text{accIn}(s, \Psi)$	The accepting flow that would enter the augmented state $\langle s, \Psi \rangle$.
$\text{accOut}(s, \phi)$	The accepting flow leaving the X-literal state $\langle s, \Psi \rangle$.
$\text{accReceive}(s, \phi)$	The accepting flow being redirected to the X-literal state $\langle s, \Psi \rangle$.
$\text{accRedist}(s, \Psi)$	The accepting flow being redirected from the augmented state $\langle s, \Psi \rangle$.
$\text{in}(s, \Psi)$	The expected number of agents passing through $\langle s, \Psi \rangle$.
$\text{out}(s, \phi)$	The expected number of agents leaving $\langle s, \phi \rangle$.
$\text{receive}(s, \phi)$	The expected number of agents reaching $\langle s, \phi \rangle$ from other states.

Heuristic Functions

h^{BA}	The non-deterministic Büchi automaton heuristic.
$h^{\text{dec}}_{\mathcal{S}}$	The decomposition heuristic for the SSP $\mathcal{S}$.
$h^{\text{sdt}}_{\mathcal{S},\psi}$	The split decomposition trace heuristic.
h^{pom}	The projection occupation measure heuristic.
$h^{\text{s-dec}}_{\mathcal{S},\psi}$	The split decomposition heuristic for the SSP $\mathcal{S}$ and PLTL constraint ψ.
$h^{\text{t-dec}}_{\mathcal{S}}$	The traced decomposition heuristic.

LTL Symbols

$\text{CNF}(\psi, s)$	A CNF set equivalent to ψ simplified according to s.
Ψ	A CNF set.
Φ	A clause in a CNF set.
$\perp$	The false primitive.
ψ	An LTL formula.
$\psi_1 \mathbf{R} \psi_2$	ψ_1 becoming true releases the requirement that ψ_2 holds.
$\psi_1 \mathbf{U} \psi_2$	ψ_1 holds until ψ_2 holds.
$\psi_1 \rightarrow \psi_2$	If ψ_1 holds then ψ_2 must hold.

$\psi_1 \vee \psi_2$	The disjunction of ψ_1 and ψ_2.
$\psi_1 \wedge \psi_2$	The conjunction of ψ_1 and ψ_2.
$\text{idle}(\Psi, s)$	Whether Ψ is satisfied by staying in s forever.
$\mathsf{F}\psi$	There exists a suffix of the path for which ψ holds.
$\mathsf{G}\psi$	Every suffix of the path must satisfy ψ.
$\mathsf{X}\psi$	The formula ψ holds in the next state.
ϕ	An MO-PLTL constraint.
$\neg\psi$	The negation of ψ.
$\top$	The true primitive.
$\text{un-}\mathsf{X}(\Psi)$	A set of sets of formulae with all instances of **X** removed.
ϕ	An X-literal.
$p \models_{\text{IE}} \psi$	The path p IE-satisfies ψ
$p \models \psi$	The path p satisfies ψ.

Operators

$\text{alive}(\mathcal{P}_i, \mathcal{A}_i)$	The paths in $\mathcal{P}_i$ which are extended in $\mathcal{P}_{i+1}$ in a run of a CC-SSP.
$\text{decompose}(s, \Psi)$	The X-literal states generated by decomposing Ψ.
$\text{Egoals}(\langle s, \cdot \rangle, \pi)$	The expected number of agents that reach a goal in a CC-SSP from the X-literal or augmented state $\langle s, \cdot \rangle$ under policy π
$\ell(\cdot)$	The length of a path or run.
$\text{minvars}(\psi)$	The minimal projection variables for ψ to be non-trivial.
$\text{assignFree}(\psi, \mathcal{V}_p)$	Project ψ onto $\mathcal{V}_p$ under the proposition independence assumption.
$\text{reduce}(\mathcal{S})$	Removes any elements in a set of sets $\mathcal{S}$ for which a smaller subset is present.

SAS Representation

$C_{\psi,i}$	One combination of variables of those output by $\text{minvars}(\psi)$.
$\mathcal{D}_v$	The domain of v.
$\mathcal{V}_{\psi,i}$	A batch of the SAS$^+$ variables in $\mathcal{V}_\psi$.
$\mathcal{V}_\psi$	The variables which appear in the formula for ψ.

$\text{proj}(s, \mathcal{V}_p)$	The projection of s onto $\mathcal{V}_p$.
α_g	An artificial action leading to the artificial goal of a projection.
$s' \subseteq s$	s' is a partial state of s.
v	A SAS$^+$ variable.
$\mathcal{V}$	The variables in a SAS$^+$ problem.
$\mathcal{S}_{\mathcal{V}_p}$	The SAS$^+$ problem $\mathcal{S}$ projected onto $\mathcal{V}_p$.
$s[v]$	The value of v in the s. Either $d \in \mathcal{D}_v$ or undefined (u).
$\mathcal{V}_s$	The vars for which s is defined.

SSP Notation

A	A set of actions.
$s^\times$	A product state $\langle s, m \rangle$
$\mathrm{S}^\times$	The set of all product states.
S^+	The set of finite paths through S.
G	The goal states of an SSP.
π	A policy.
π^*	An optimal policy.
$C(\alpha)$	The cost of transitioning to s' from s by α.
$\text{Gruns}(s, \pi)$	The set of all runs starting at s possible under π which end at a goal.
$\mathcal{S}$	An SSP.
$V^\pi(s)$	The expected cost of reaching a goal from s under π.
S	The state space for an SSP.
$\mathrm{T}(s' \mid s, \alpha)$	The probability of transitioning to s' from s by α.
M	The set of values an arbitrary mode can take.
m	The value of a mode.
r	A run for an SSP.

Linear Program Variables and Constants

$\tilde{\mathrm{Y}}_{\mathcal{C}}$	The set of accepting flow variables of the form $y_{s,\phi,\alpha,s'}$.

$\hat{\mathrm{Y}}_{\mathcal{C}}$	The set of accepting flow variables of the form $y_{s,\Psi,\phi}$.
$\mathrm{X}_{\mathcal{C}}$	The set of occupation measures for $\mathcal{C}$.
$i_{s,\Psi}$	What flow entering at state $\langle s, \Psi \rangle$ is accepting flow.
$x_{s,\phi,\alpha}$	An occupation measure for a CC-SSP.
$y_{s,\Psi,\phi}$	An accepting flow variable labelling the flow redirected from $\langle s, \Psi \rangle$ to $\langle s, \phi \rangle$.
$y_{s,\phi,\alpha,s'}$	An accepting flow variable labelling the flow from state $\langle s, \phi \rangle$ towards s' by action α.

Chapter 1

Introduction

Automatically determining safe plans to solve problems in stochastic environments is a problem that is a point of interest in recent research, and has proven to be quite hard to scale to anything but small problems. An example of such a problem is managing traffic lights in a city to minimise traffic congestion, in such a way that (assuming the traffic obeys the signals), there is never a possibility of a collision. Other constraints could be placed on the plan, for example that during rush hour, an inbound car on a major road should not be stopped more than once with a high probability, minimising inconvenience to traffic in a hurry, or that no car should have to wait longer than a certain amount of time at any intersection.

Defining constraints on problems like this also provides some proportion of customisability. For example, given a problem of choosing academic courses for a student, different student's preferences and requirements can be encoded as constraints, meaning that one problem definition can serve many different students.

1.1 Overview of the Research Problem

This thesis sits at the crossroads between two well-researched fields, that of probabilistic planning, to generate policies under uncertainty, and probabilistic linear temporal logic, to express constraints that such plans must satisfy. To express the problem clearly, some terminology and concepts related to these two fields is introduced, followed by a description of the problem to be addressed.

1.1.1 Probabilistic Planning

Planning is the problem in the field of Artificial Intelligence (AI) of choosing actions so as to achieve the goal, ideally optimising some metric such as the cost of those actions. In planning, the world is modelled as a set of states, including an initial state and goal states, and a set of actions that transition between these states. A solution to such a problem is a *policy*, stating which actions should be taken in what situations so that an agent gets from the initial state to some goal state.

The world is not deterministic, and there are often many factors outside a model which cannot be predicted. To account for this, it is natural to include some stochastic behaviour in a planning model. Probabilistic planning allows actions to have stochastic

effects, where each time an action is taken, the resulting state is chosen randomly from a pre-defined probability distribution of outcomes. When planning in a stochastic environment, an optimal solution minimises the expectation of a metric over all the possible paths taken when following a policy. The model for such a problem is referred to as a Stochastic Shortest Path problem (SSP) [Bertsekas and Tsitsiklis, 1991], so called because it is a generalisation of the problem of finding the shortest path through a graph.

There has recently been an increase in interest among the planning community in planning under constraints defined in terms of the temporal properties of the policy. For example, such a constraint might state that a Mars Rover must always maintain its battery levels above a certain critical threshold, or that in the case of a dust storm, it must reach shelter within a certain time limit.

1.1.2 Linear Temporal Logic

There are many languages for expressing constraints, but the language used in this thesis is probabilistic linear temporal logic (PLTL). PLTL expresses the probabilistic temporal properties of the paths taken while following a policy. PLTL is an extension of linear temporal logic (LTL) [Pnueli, 1977], which is in turn an extension of propositional logic. A PLTL constraint is a probability bound on the satisfaction of an LTL formula under a policy.

Using PLTL, various concepts from verification can be enforced in a policy, for example:

- safety constraints: e.g., with probability 0.99, a robot must never do anything that might cause it to drop an explosive item;

- liveness constraints: e.g., with probability 1, the traffic lights for every direction at an intersection should always eventually go green, i.e., even when they have been green before;

- responsiveness constraints: e.g., with probability 0.95, every time a server shuts down in a network, it must eventually be re-activated;

but the set of constraints for PLTL are not limited to these properties. PLTL is quite expressive because the associated LTL formulae can be nested, providing formulae like "*an aeroplane must be maintained regularly up until it never flies again*".

Checking whether a given path satisfies an LTL formula involves constructing a representation of the LTL formula which tracks the state of the formula as the path is traversed. For example, consider a formula that states that customers a, b and c must eventually be served. The representation for this formula would have to keep track of which of the three had been served so far, so that when a and c have already been served, observing a state in which b is served would satisfy the formula.

1.1.3 Applying LTL in Probabilistic Planning

The application of one or more PLTL constraints to an SSP is referred to as a multiple-objective probabilistic linear temporal logic Stochastic Shortest Path problem (MO-PLTL SSP problem). A solution to a MO-PLTL SSP problem must both reach the goal of the SSP and satisfy the associated PLTL constraints. Finding a solution to an MO-PLTL SSP problem is typically done by creating a state based representation of each LTL formulae and considering a state as a combination of an LTL state and an SSP state, e.g., [Kwiatkowska and Parker, 2013; Baumgartner et al., 2018].

The number of states in a typical planning problem is exponential in the *factored* description of the SSP, and the number of states in a representation of an LTL formula is worst-case double exponential in the length of the formula, meaning that explicitly considering every combination of these states is computationally intractable.

1.1.4 Heuristics

Finding a policy for a problem with so many states can often be done by considering only a fraction of the combined state space. This subset must be sufficient to include all the states reachable in the policy, and to be effective, an algorithm needs to find this subset while minimising unnecessary states considered. This is done by creating a *heuristic function*. A heuristic function guides the search algorithm towards the goal by estimating how difficult it is to reach the goal from any given state.

A heuristic function can be constructed for a specific class of problems called a domain, e.g., the domain of parcel delivery problems, or constructed independently of the domain, referred to as a *domain-independent heuristic*. Domain-independent heuristics exploit the structure of the problem representation. In this thesis, the probabilistic SAS^+ formalism is used as a problem representation, where a SAS^+ problem defines an SSPs, and PLTL constraints are represented by *progression*.

Heuristic search has been applied to MO-PLTL SSP problems with success by Baumgartner et al. [2018], through their state-of-the-art algorithm PLTL-dual. They provide heuristics which make estimates based on both the SSP state and the LTL state, however, they provide only one heuristic for LTL, which relies on a transformation of LTL to non-determinisitic Büchi automata (NBA). The NBA constructed by this transformation has size exponential in the size of the LTL formula in the worst case. Given this issue, they observe that for problems with large LTL constraints, constructing this NBA and using it for the heuristic takes long enough that it can counteract the gains achieved from using heuristic search.

The tests performed by Baumgartner et al. [2018] showed that using a different LTL representation called progression had results that were almost competitive with their NBA heuristic despite not using any heuristic. They conjecture that this was because progression does not have to construct the very large NBAs. This aligns with preliminary results found earlier by Kerjean et al. [2006] that showed that on random formulae, progression is much more efficient than Büchi automata translations.

This raises the question:

> *How can LTL progression be used as the basis for a domain-independent heuristic for solving MO-PLTL SSP problems? Can using such a heuristic improve the performance of MO-PLTL SSP problem solvers?*

This thesis sets out to construct such a heuristic and evaluate its effectiveness.

1.2 Contributions

This thesis constructs a heuristic based on progression by combining two *relaxations* of MO-PLTL SSP problems in order to create greatly simplified planning problems. The optimal solution of these relaxed planning problems can be used as a heuristic estimate. A relaxation is a simplification of a problem which makes the problem easier to solve, typically by ignoring aspects of the original problem. The two relaxations employed are referred to as *projection* and *decomposition*, which relax the dynamics of the SSP and the PLTL constraints respectively.

1.2.1 Projection

Projection is an existing concept that has been used for many heuristics for classical planning and probabilistic planning [Helmert et al., 2007; Haslum et al., 2007; Trevizan et al., 2017a]. The idea behind projection is that SSPs are represented as a set of variables, and a state is an assignment to these variables. By simply removing most of the variables and appropriately adapting the structure of actions, the SSP is reduced to the problem of changing the values of only the remaining variables to their goal values. For example, consider a transport problem, in which trucks and aeroplanes move around to carry packages to their destinations. A projection onto the position of a single package reduces the problem to the movements that *that* single package needs to take to reach its goal. The positions of the other packages are ignored, and the positions of the trucks and aeroplanes are ignored, the relaxation simply assumes they are where they need to be, when they need to be there. Typically a set of complementary projections are chosen in such a way that the estimate from each can be combined.

If projections were to be used on a problem with PLTL constraints, an issue arises. Variables removed by projection become free in the PLTL constraint, which makes progression significantly more difficult. This thesis contributes in several ways to the problem of projecting with PLTL constraints:

- Projection is extended efficiently to PLTL constraints, defining an algorithm which, given an LTL formula and variables to project onto, constructs a formula defined only over those variables which is satisfied by at least the same paths.
- The conditions under which projection will result in a PLTL formula becoming trivial are presented.

- A simple algorithm is presented which chooses a set of complementary projections for a PLTL constraint, such that every projection is non-trivial.

1.2.2 Decomposition

Decomposition of LTL formula is a novel concept introduced to relax the dynamics of an LTL formula in the context of progression. An LTL formula can be represented as a set of subformulae connected by 'and' and 'or' connectives, and the concept of decomposition is that the difficulty of satisfying the original formula can be estimated in terms of the difficulty of satisfying each subformulae.

Intuitively, decomposition works by placing agents in states associated with subformulae depending on how the subformulae are connected. Given that two subformulae should both be satisfied, an agent is designated one at random, which is obviously easier to satisfy than both. If one or both of two subformulae should be satisfied, then the agent is *cloned* and one clone is designated each subformula. If one or more of the agents satisfy their designated formula, it can be considered that the formula is satisfied overall. Often, the subformulae are as complex as the original formula, so decomposition is performed repeatedly at each step of the relaxed problem to counteract this.

The contributions of this thesis in regards to decomposition are separated into two stages, first an implementation for a heuristic based on decomposition is presented, followed by an analysis of this heuristic, and a subsequent adaptation to address some of the issues and make it possible to integrate into the state-of-the-art planner PLTL-dual. The first stage of contributions consists of:

- The novel concept of formula decomposition.

- A planning model called a Concurrent Constrained SSP (CC-SSP) which, when constructed from an SSP and a PLTL constraint, simulates repeated decomposition.

- A heuristic for MO-PLTL SSP problems, referred to as the decomposition heuristic, which is computed by a linear program formulation of a CC-SSP.

The decomposition heuristic suffers from some major limitations, and cannot be integrated directly into PLTL-dual, so the second stage involves an adaptation of it. The contributions in the second stage are

- An adaptation of the decomposition heuristic with improved informativeness, done by tracing the movement of agents which eventually reach goals in such a way that they satisfy the PLTL constraint.

- A further adaptation of the decomposition heuristic to include the features of a heuristic for PLTL-dual, specifically a probabilistic initial state and tying constraints.

- A heuristic for PLTL-dual, called the split decomposition trace (SDT) heuristic, which combines projection and decomposition to relax both aspects of the problem.

The efficacy of the SDT heuristic is empirically tested on three planning domains, one of which is a novel adaptation of an existing domain, and the other two introduced by Baumgartner et al. [2018]. The results for the heuristic are generally favourable, but show clear areas for improvement. In the Wall-e domain introduced by Baumgartner et al. [2018], the SDT heuristic excels, solving problems an order of magnitude faster, whereas in their factory domain, it performs on par with other approaches for large problem instances. A new domain adapted from [Trevizan et al., 2016] is introduced, for which the results demonstrate that the SDT heuristic scales to problems with harder objectives better than any other approach.

1.3 Thesis Outline

This remainder of this thesis is structured in a standard fashion:

- Chapter 2 provides formal definitions and explanations of the background concepts necessary to understand this thesis, primarily those for probabilistic planning and PLTL.
- Chapter 3 details some other research into the use of LTL in planning or synthesis. Notably, two existing solvers for MO-PLTL SSP problems are detailed.
- Projection of PLTL constraints is presented in chapter 4.
- The first stage of contributions for LTL decomposition are presented in chapter 5, detailing the novel planning model, the Concurrent Constrained SSP, and the associated heuristic.
- The second stage of contributions for LTL decomposition are presented in chapter 6, primarily the integration of LTL decomposition into PLTL-dual.
- The empirical evaluation of the SDT heuristic is presented in chapter 7.
- Chapter 8 contains a summary of the thesis and discussion of directions for future research.

Chapter 2

Background

Planning in stochastic environments has been studied in depth over the last several decades, with many branches for varieties of model and objective. This thesis addresses a sub-branch of planning in stochastic environments where the objective is to reach the goal under constraints on the probability that paths will satisfy certain temporal properties.

This chapter details the background knowledge from this field used in this thesis. This branch of planning is, if you will, a combination of probabilistic planning and probabilistic linear temporal logic (PLTL), which is an extension of linear temporal logic (LTL). LTL was proposed initially by Pnueli [1977] for formally defining the correctness of a computer program for the purpose of verification. The work in this thesis primarily derives from planning, so that is the primary focus in this chapter.

Section 2.1 defines the notation that will be used for paths, which is necessary for the definition of both LTL and probabilistic planning.

To facilitate the introduction of PLTL later on, section 2.2 defines LTL as it is used in the context of this thesis, focussing on symbolic progression of formulae and LTL normal forms. One of the most common methods for analysing PLTL is by converting it into Rabin or Büchi automata, but automata are not used for any part of the contribution of this thesis. As such, the details of Rabin and Büchi automata are not included in this chapter, nor is the associated conversion. The notation for progression used in this thesis is also presented in this section.

Section 2.3 defines the models and concepts used throughout this thesis related to planning. The model used is the Stochastic Shortest Path (SSP) problem, specifically using the SAS$^+$ representation. Various concepts related to SSPs are introduced, as well as the related concept of heuristic search and constrained planning.

This thesis sits in the intersection of these two fields, discussing a type of problem where the policies for SSPs are restricted by PLTL constraints. Section 2.4 defines PLTL and this PLTL constrained SSP, referred to as a MO-PLTL SSP problem.

2.1 Paths

This section details the notation and concepts surrounding finite and infinite paths, which is necessary for defining both LTL and is useful in probabilistic planning. A

path $p = s_1, s_2, \ldots$ is a sequence of elements in some set of states Γ. Given a set Γ, the set of all finite paths in Γ is denoted Γ^+, and the set of all infinite paths in Γ is Γ^ω. The *length* of a path p is denoted $\ell(p)$, and for $p \in \Gamma^\omega$, $\ell(p) = \infty$. The first state in a path is $p[1]$, then $p[2]$ and so on, so the ith state is $p[i]$. The last state in a path is $\text{last}(p) \equiv p[\ell(p)]$, and is therefore undefined if $\ell(p) = \infty$.

A *prefix* of a path $p \in \Gamma^+ \cup \Gamma^\omega$ is a path $p' \in \Gamma^+$ such that $\ell(p) \geq \ell(p')$ and $p[i] = p'[i]$ for $1 \leq i \leq \ell(p')$. The prefix of p up to and including state i is denoted $p[\leq i]$. A *suffix* of a path $p \in \Gamma^+ \cup \Gamma^\omega$ is a path $p' \in \Gamma^+ \cup \Gamma^\omega$ such there exists $j \in \mathbb{N}$ such that $p'[i] = p[j+i]$ for $1 \leq i \leq \ell(p')$. The suffix of p starting from the ith state is denoted $p[\geq i]$.

2.2 Linear Temporal Logic

LTL is a form of logic for describing the properties of infinite sequences. Propositional LTL is used in this thesis, which is propositional logic extended with modalities that allow it to express relationships between the labels in the sequence.

LTL can express many concepts common in model checking and program verification such as

- fairness: given multiple options, each must be chosen an infinite number of times;
- liveness: the system never deadlocks, i.e., a specific property must always eventually be satisfied;
- safety: some property must always be satisfied, e.g., opposite traffic lights must never both be green;
- responsiveness: each time a request is recieved, there must be a response eventually;

and many other temporal properties.

The definition of LTL presented here largely follows the definition of LTL and its identities in [Baier and Katoen, 2008] and [Duret-Lutz, 2016], but diverges when describing normal forms and progression.

2.2.1 Syntax and Semantics

LTL is defined over a set AP of atomic propositions, which are statements about the world, for example, *reactor is stable* may be a proposition representing whether a nuclear reactor is stable, and would likely be abbreviated as r or similar. Any propositional variable is either true ($\top$) or false ($\bot$). A state $s \in 2^{AP}$ is the collection of propositional variables which are true in that state. LTL describes the properties of infinite paths $s_0, s_1, \cdots \in \left(2^{AP}\right)^\omega$. The properties of a path are described using the next ($\mathbf{X}$) operator, which expresses a property in the next state, and the until ($\mathbf{U}$) operator, which expresses that some property must hold until another does. As an extension of propositional logic, LTL also includes and ($\wedge$), not ($\neg$) and true ($\top$).

The **syntax** of an LTL formula ψ is defined by the grammar:

$$\psi \implies \top \mid a \mid (\psi) \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi$$

where a can be any proposition in AP. The unary operators $\mathbf{X}$ and $\neg$ have precedence over the binary operators, and the temporal operator $\mathbf{U}$ has the next precedence, so $\mathbf{X}\psi_1\mathbf{U}\psi_2 \wedge \psi_3$ is equivalent to $((\mathbf{X}\psi_1)\mathbf{U}\psi_2) \wedge \psi_3$, however unnecessary parentheses are often included to aid clarity.

Syntactic sugar is added in the form of the following extra propositional and temporal operators:

- The false primitive $\bot \equiv \neg\top$.
- The or operator $\psi_1 \vee \psi_2 \equiv \neg(\neg\psi_1 \wedge \neg\psi_2)$.
- The implication operator $\psi_1 \rightarrow \psi_2 \equiv \neg\psi_1 \vee \psi_2$.
- The future operator $\mathbf{F}\psi \equiv \top\mathbf{U}\psi$.
- The globally operator $\mathbf{G}\psi \equiv \neg\mathbf{F}\neg\psi$.
- The release operator $\psi_1\mathbf{R}\psi_2 \equiv \neg(\neg\psi_1\mathbf{U}\neg\psi_2)$.

The **semantics** of an LTL formula are defined in terms of paths $p \in \left(2^{AP}\right)^\omega$, and a path p is said to satisfy a formula ψ if p is has the properties defined by the semantics of ψ. $p \models \psi$ denotes the statement "p satsifies ψ", and $p \not\models \psi$ denotes "p does not satisfy ψ". The $\models$ relation is defined recursively for LTL formulae below, and following that an intuition for the operators (including the syntactic sugar symbols) will be provided.

$$\begin{aligned}
&p \models \top \\
&p \models a && \iff a \in p[1] \\
&p \models \neg\psi && \iff p \not\models \psi \\
&p \models \psi_1 \wedge \psi_2 && \iff p \models \psi_1 \text{ and } p \models \psi_2 \\
&p \models \mathbf{X}\psi && \iff p[\geq 2] \models \psi \\
&p \models \psi_1\mathbf{U}\psi_2 && \iff \exists j \geq 0 \text{ s.t.}\, p[\geq j] \models \psi_2 \text{ and } \forall i < j,\, p[\geq i] \models \psi_1
\end{aligned}$$

The intuition for the non-temporal operators are very straightforward. Any path p satisfies $\top$, so conversely no path satisfies $\bot$. The formula a requires that a is true 'now', which means in the first state in the path, and places no restriction on other states in the path. Note that the concept of 'now' depends on the context in the formula; for example, $\mathbf{X}a$ states that in the next state, a is true, so the presence of a does not necessarily refer always to the first state in the path. $\wedge$ and $\vee$ represent 'and' and 'or' from propositional logic respectively. $\psi_1 \wedge \psi_2$ is satisfied by a path which satisfies both ψ_1 and ψ_2. $\psi_1 \vee \psi_2$ is satisfied by any path which satisfies either ψ_1 or ψ_2 (or both). Finally, $\psi_1 \rightarrow \psi_2$ is satisfied by p if ψ_1 is not satisfied by p or if ψ_2 is satisfied by p.

To aid the understanding of temporal operators, figure 2.1 shows the intuition for each as a path through states.

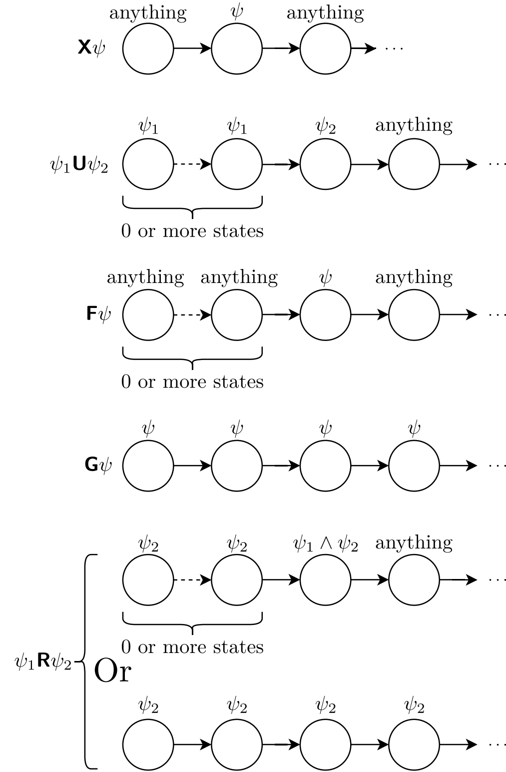


Figure 2.1: Intuition for temporal operators in LTL.

The next operator is straight forward, for $\mathbf{X}\psi$ to hold for a path, ψ should hold for the paths starting at the next state. Although LTL describes properties of paths, a ψ can nonetheless be thought of as holding in a state, so long as the future beyond that state is fixed. In this intuition, $\mathbf{X}\psi$ holds in a state if ψ holds in the next state.

The until operator behaves mostly as is expected from the name. $\psi_1\mathbf{U}\psi_2$ holds for a path if ψ_1 holds for every suffix of that path until the first suffix where ψ_2 holds. Additionally, there *must* be some suffix for which ψ_2 holds. Note that if ψ_2 holds for the path from the beginning, there need not be any suffix for which ψ_1 holds.

The future operator can also be thought of as "eventually". For $\mathbf{F}\psi$ to hold for a path, there must exist some suffix of the path for which ψ holds.

The globally operator, like the future operator, can be thought of as "always". For $\mathbf{G}\psi$ to hold for a path, every suffix of that path (including the path itself) must satisfy ψ.

The release operator is very similar to the until operator. The formula $\psi_1\mathbf{R}\psi_2$ can be read ψ_1 releases ψ_2. Under this operator, there is a constraint that ψ_2 must be true at all times, and ψ_1 becoming true "releases" ψ_2 from its constraint. Superficially, ψ_2 must be true until ψ_1 first becomes true, resulting in a state from which both ψ_1 and ψ_2 are true. The other primary difference from the until operator is that there is no requirement that ψ_1 is true for any suffix of the path. In this case, ψ_2 must be true from every state in the path.

2.2.2 Infinite Extension

In this thesis, the paths of interest are finite, and LTL has been extended in several different ways to specify the properties of a finite path. In this thesis, infinite extension semantics [Bacchus and Kabanza, 1998] are applied. The concept of infinite extension semantics can be summarised as turning a finite path into an infinite path by repeating the last state an infinite number of times.

Definition 1. For some finite path $p \in \left(2^{AP}\right)^+$ and formula ψ, it is said that p satisfies ψ under infinite extension semantics, denoted $p \models_{\text{IE}} \psi$, if and only if $p\,\text{last}(p)^\omega \models \psi$ where $\text{last}(p)^\omega$ is an infinite repetition of the last state in p.

There are several other formalisms for extending LTL to finite paths, such as LTL_f [De Giacomo and Vardi, 2013], where the next operator evaluates to false if it refers to a time beyond the current step, whereas FLTL, one of several formalisms reviewed in [Bauer et al., 2010] introduces a weak next operator to complement this, which evaluates to true if it refers to a time beyond the current one. [Baier and Mcilraith, 2006] defines f-FOLTL which introduces another symbol `final`, which evaluates to true only in the last state of finite path. Of these formalisms, IE-semantics are commonly used for planning, and importantly, the state-of-the-art planner PLTL-dual uses IE-semantics, meaning that integrating with it in turn requires the use of IE-semantics.

Satisfaction under infinite extension semantics (IE-semantics) is equivalent in many ways to LTL. Bauer and Haslum [2010] find that any equivalence between LTL formulae exists also between the formulae under IE-semantics. They also show that the problem of LTL satisfaction under IE-semantics can be reduced to LTL satisfaction, by a short extension of the LTL formula in question representing the infinite extension. Bauer and Haslum also review other finite path LTL semantics.

2.2.3 Identities

Two LTL formulae ψ and ψ' are considered equivalent if, for all paths $p \in \left(2^{AP}\right)^{\omega}$, $p \models \psi$ if and only if $p \models \psi'$. Under this definition, certain families of formulae are obviously equivalent to each other, and some of these identities are listed in this section.

Various LTL identities are used to simplify formulae and also to convert formulae to normal forms, detailed in the next section. The list of identities presented here is far from a comprehensive list of identities for LTL, but lists all the ones used in this thesis.

The *trivial identities* for LTL and propositional logic are used to simplify formulae, in very common cases, and for the most part involve trivial subformulae $\top$ or $\bot$. This list is as follows:

$$\begin{array}{rrr}
\psi \wedge \bot \equiv \bot & \psi \wedge \top \equiv \psi & \\
\neg\neg\psi \equiv \psi & & \\
\mathbf{X}\top \equiv \top & \mathbf{X}\bot \equiv \bot & \\
\psi\mathbf{U}\top \equiv \top & \psi\mathbf{U}\bot \equiv \bot & \bot\mathbf{U}\psi \equiv \psi
\end{array}$$

And for the derived operators, the following trivial identities can be derived:

$$\begin{array}{rrr}
\psi \vee \bot \equiv \psi & \psi \vee \top \equiv \top & \\
\psi \rightarrow \bot \equiv \neg\psi & \psi \rightarrow \top \equiv \top & \\
\bot \rightarrow \psi \equiv \top & \top \rightarrow \psi \equiv \psi & \\
\mathbf{G}\top \equiv \top & \mathbf{G}\bot \equiv \bot & \mathbf{GG}\psi \equiv \mathbf{G}\psi \\
\mathbf{F}\top \equiv \top & \mathbf{F}\bot \equiv \bot & \mathbf{FF}\psi \equiv \mathbf{F}\psi
\end{array}$$

There are various identities used to move negation inwards and also to separate the requirements on the first state in the formula from the others, referred to as *duality laws*. The duality laws are largely based on the derived operators, and in fact the $\mathbf{R}$ operator is included specifically to be a dual operator for $\mathbf{U}$. The duality laws are as follows:

$$\begin{aligned} \neg(\psi_1 \vee \psi_2) &\equiv \neg\psi_1 \wedge \neg\psi_2 & \neg(\psi_1 \wedge \psi_2) &\equiv \neg\psi_1 \vee \neg\psi_2 \\ \neg\mathbf{X}\psi &\equiv \mathbf{X}\neg\psi & \neg(\psi_1 \rightarrow \psi_2) &\equiv \psi_1 \wedge \neg\psi_2 \\ \neg(\psi_1\mathbf{U}\psi_2) &\equiv \neg\psi_1\mathbf{R}\neg\psi_2 & \neg(\psi_1\mathbf{R}\psi_2) &\equiv \neg\psi_1\mathbf{U}\neg\psi_2 \\ \neg\mathbf{G}\psi &\equiv \mathbf{F}\neg\psi & \neg\mathbf{F}\psi &\equiv \mathbf{G}\neg\psi. \end{aligned}$$

There are several rewriting rules for moving $\wedge$ and $\vee$ up and down in the formula found in propositional logic and called the *distributive laws*, defined as follows:

$$\begin{aligned} \psi_1 \vee (\psi_2 \wedge \psi_3) &\equiv (\psi_1 \vee \psi_2) \wedge (\psi_1 \vee \psi_3) \\ \psi_1 \wedge (\psi_2 \vee \psi_3) &\equiv (\psi_1 \wedge \psi_2) \vee (\psi_1 \wedge \psi_3). \end{aligned}$$

There are distributive laws for LTL also, but they are not applied to construct the normal forms used in this thesis, and so are omitted from this section. Note that rewriting a formula according to these rules can (in the worst case) nearly double its length, whereas the duality and trivial identities add at most one symbol to the formula, or shorten it.

And finally the most important set of identities is the *expansion laws*, used to separate the parts of a formula referring to the future and the parts of a formula referring to 'now'.

$$\begin{aligned} \psi_1\mathbf{U}\psi_2 &\equiv \psi_2 \vee (\psi_1 \wedge \mathbf{X}(\psi_1\mathbf{U}\psi_2)) & \psi_1\mathbf{R}\psi_2 &\equiv \psi_2 \wedge (\psi_1 \vee \mathbf{X}(\psi_1\mathbf{R}\psi_2)) \\ \mathbf{F}\psi &\equiv \psi \vee \mathbf{XF}\psi & \mathbf{G}\psi &\equiv \psi \wedge \mathbf{XG}\psi \end{aligned}$$

The expansion laws can be used to reason about each successive step in a path, by recursively applying them to isolate the constraint on each successive state. Like the distributive laws, rewriting a formula according to these identities can double the size of the formula.

2.2.4 Normal Forms

There are two normal forms for LTL used in this thesis, one being the standard negation normal form extended for LTL, and the other is a form based on conjunctive normal form for propositional logic, which diverges from the expected form somewhat.

Definition 2. Negation normal form is a structure of logical formulae in which the negation operator ($\neg$) is applied only to propositional atoms. Formulae in negation normal form are exactly those constructed by the grammar

$$\psi \implies \top \mid \bot \mid a \mid \neg a \mid (\psi) \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \psi\mathbf{R}\psi$$

where a can be any proposition in AP.

For any formula ψ, there exists a formula ψ' in negation normal form equivalent to ψ, which can be found be repeatedly applying the duality laws and double negation ($\neg\neg\psi \equiv \psi$) until all instances of negation are pushed to atomic propositions. In negation normal form, subformulae of the form a and $\neg a$ are referred to as *literals*, and by reasoning about literals only, negation is effectively removed from discourse. Note however that in this form the inclusion of $\vee$ and $\mathbf{R}$ is necessary, as they are the dual operators for $\wedge$ and $\mathbf{U}$. The $\mathbf{G}$ and $\mathbf{F}$ operators are not necessary, but in practice are included for convenience. Conversely, in negation normal form, it is convenient to rewrite $\psi_1 \rightarrow \psi_2$ as $\neg\psi_1 \vee \psi_2$ according to its definition.

The second normal form used is based on conjunctive normal form in propositional logic, and by an abuse of terminology, it will be referred to as conjunctive normal form (CNF) throughout this thesis.

Definition 3. A formula in conjunctive normal form is a formula in negation normal form such that all temporal operators are restricted to subformulae referred to as X-literals (the reason for this name will become evident), such that the main operator on the subformula is $\mathbf{X}$. Instances of disjunction not inside X-literals may only be applied to literals and X-literals, where the resulting disjunction is referred to as *clause*, and clauses may be connected by conjunctions. LTL formulae in CNF are exactly those constructed by the following grammar:

$$\begin{aligned}
\varphi &\Longrightarrow \texttt{clause} \mid \texttt{clause} \wedge \varphi \\
\texttt{clause} &\Longrightarrow \texttt{literal} \mid \texttt{literal} \vee \texttt{clause} \\
\texttt{literal} &\Longrightarrow a \mid \neg a \mid \phi \\
\phi &\Longrightarrow \mathbf{X}(\psi) \\
\psi &\Longrightarrow \top \mid \bot \mid a \mid \neg a \mid (\psi) \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U} \psi \mid \psi \mathbf{R} \psi
\end{aligned}$$

where a can be any proposition in AP.

In this definition, and throughout this thesis, ϕ is used as a symbol for a X-literal, and ψ is used for an arbitrary LTL formula, often in negation normal form.

Given any formula ψ, by converting it to negation normal form and then repeatedly applying the distributive law $\psi_1 \vee (\psi_2 \wedge \psi_3) \equiv (\psi_1 \vee \psi_2) \wedge (\psi_1 \vee \psi_3)$ and the expansion laws, an equivalent formula ψ' in CNF can be constructed. It was mentioned in section 2.2.3 that rewriting according to both expansion laws and distributive identities doubles the length of the formula, so after repeated rewriting, in the worst case, the length of ψ' may be exponential in the length of ψ. The value of CNF is not only that it is very straightforward to represent and reason about, but also that there is a clear distinction between temporal and non-temporal parts of the formula, with X-literals being temporal formulae which do not refer to the first state in a path, and the non-temporal literals restricting only the first state in the path.

2.2.5 Progression

In order to determine whether an LTL formula is satisfied in the realm of planning, a commonly used approach is *formula progression*, introduced by Bacchus and Kabanza [1998]. Intuitively, the concept of formula progression is updating a formula each time a new state is observed. As an iterative process, progression can only be used for formulae which can be proven satisfiable or unsatisfiable by a finite prefix of a path, making it less applicable for LTL defined on infinite paths. However on finite paths, formula progression is feasible and can be more computationally efficient than the alternatives in practice.

Formula progression is a transformation which, given an LTL formula ψ and a state $s \in 2^{AP}$, constructs a formula ψ' such that $sp \models_{\mathrm{IE}} \psi$ if and only if $p \models_{\mathrm{IE}} \psi'$ for all paths $p \in (2^{AP})^+$. In this section, this transformation is performed by a pair of operators $\mathrm{CNF}(\cdot, \cdot)$ which converts a formula to CNF, simultaneously simplifying it according to some state, and un-$\mathbf{X}(\cdot)$, which removes the $\mathbf{X}$ from the start of each X-literal. However, to make notation and representation easier, instead of using formulae directly, formulae are represented using a set of sets of formulae.

Definition 4. A CNF set Ψ is a set of sets Φ, referred to as clauses, where each Φ is a set of X-literals ϕ. A CNF set Ψ represents the formula

$$\bigwedge_{\Phi \in \Psi} \bigvee_{\phi \in \Phi} \phi.$$

Given a state s and a formula ψ, $\mathrm{CNF}(\psi, s)$ is a CNF set Ψ constructed by applying the following steps:

1. Convert ψ to negation normal form.

2. Traverse the syntax tree to find an instance of $\mathbf{U}$, $\mathbf{R}$, $\mathbf{G}$ and $\mathbf{F}$ that is not inside the scope of another temporal operator ($\mathbf{U}$, $\mathbf{R}$, $\mathbf{G}$, $\mathbf{F}$, $\mathbf{X}$). Rewrite the sub-formula rooted at this operator according to the expansion laws,

3. Evaluate all literals not inside a the scope of a temporal operator according to the state interpretation s, i.e., replace all such propositions a with $\top$ if $a \in s$ and with $\bot$ otherwise, and the opposite replacement for $\neg a$.

4. Simplify the resulting formula according to the trivial identities for LTL.

5. If there exists a temporal operator other than $\mathbf{X}$ not inside the scope of an $\mathbf{X}$, go back to step 2.

6. Use the distribution laws to convert the formula to CNF.

7. Construct a CNF set from the resulting X-literals.

Note that by step 7 there will be no literals outside the scope of a temporal operator, because of the evaluation performed in step 3. If the formula was already simplified before step 2, step 4 need only start from the propositions that were evaluated.

As example of the CNF operator, consider the LTL formula $\psi = (a \lor \mathbf{X}a)\mathbf{U}(\mathbf{G}b \lor \mathbf{G}c)$ in the state $s = \{c\}$. The following is a trace of the transformations on ψ to construct $\text{CNF}(\psi, s)$ conflating steps 3 and 4 for brevity.

Step 1: ψ is already in negation normal form.
Step 2: $(\mathbf{G}b \lor \mathbf{G}c) \lor ((a \lor \mathbf{X}a) \land \mathbf{X}(\psi))$
Step 3-4: $(\mathbf{G}b \lor \mathbf{G}c) \lor (\mathbf{X}a \land \mathbf{X}(\psi))$
Step 2: $((b \land \mathbf{XG}b) \lor \mathbf{G}c) \lor (\mathbf{X}a \land \mathbf{X}(\psi))$
Step 3-4: $\mathbf{G}c \lor (\mathbf{X}a \land \mathbf{X}(\psi))$
Step 2: $(c \land \mathbf{XG}c) \lor (\mathbf{X}a \land \mathbf{X}(\psi))$
Step 3-4: $\mathbf{XG}c \lor (\mathbf{X}a \land \mathbf{X}(\psi))$
Step 5: All instances of $\mathbf{U}, \mathbf{R}, \mathbf{G}$ and $\mathbf{F}$ are within the scope of an $\mathbf{X}$
Step 6: $(\mathbf{XG}c \lor \mathbf{X}a) \land (\mathbf{XG}c \lor \mathbf{X}(\psi))$
Step 7: $\{\{\mathbf{XG}c, \mathbf{X}a\}, \{\mathbf{XG}c, \mathbf{X}(\psi)\}\}$

So $\text{CNF}(\psi, s) = \{\{\mathbf{XG}c, \mathbf{X}a\}, \{\mathbf{XG}c, \mathbf{X}(\psi)\}\}$.

The un-$\mathbf{X}(\phi)$ operator removes the $\mathbf{X}$ from the beginning of ϕ, and when applied to a set, recursively applies un-$\mathbf{X}$ to each element of the set. Therefore, given a CNF set Ψ, un-$\mathbf{X}(\Psi)$ will remove the $\mathbf{X}$ operator from the beginning of every X-literal in every clause in Ψ. The resulting set of sets of formulae Ψ' is not a CNF set, as the formulae in it are not necessarily X-literals. Semantically, Ψ' represents the formula

$$\bigwedge_{\Phi' \in \Psi'} \bigvee_{\psi \in \Phi'} \psi,$$

and when the two operators are applied simultaneously, $\text{CNF}(\text{un-}\mathbf{X}(\Psi), s)$ is the CNF set constructed from this formula under the state interpretation s.

Formula progression is performed by chaining these two operators in an alternating fashion. Given a formula ψ and a finite path $p \in \left(2^{AP}\right)^+$, whether $p \models_{\text{IE}} \psi$ can be determined by stepping through the states in p, labelling the first with $\Psi_1 = \text{CNF}(\psi, p[1])$ and labelling each consecutive state $p[i+1]$ with the next CNF set $\text{CNF}(\text{un-}\mathbf{X}(\Psi_i), s_{i+1})$ for each $i \geq 1$.

At the end of p, $\Psi_{\ell(p)}$ will not necessarily be $\top$ or $\bot$, but Bacchus and Kabanza [1998] provide the idle operator, which determines whether staying in a given state forever will satisfy a formula. Formally,

$$\text{idle}(\psi, s) \equiv \begin{cases} \top & \text{if } s^\omega \models \psi \\ \bot & \text{if } s^\omega \not\models \psi. \end{cases}$$

The idle operator can be adapted for the CNF set representation simply by considering the formula represented by it, so $\text{idle}(\Psi, s)$ is used interchangeably. $\text{idle}(\psi, s)$ can be computed recursively by algorithm 1, defined for an arbitrary formula ψ.

To discuss the difficulty of progression, it is important to consider what CNF sets

Algorithm 1 An algorithm for determining whether staying in a state indefinitely will satisfy a formula.

$$\begin{aligned}
\text{idle}(\top, s) &= \top \\
\text{idle}(a, s) &= a \in s \\
\text{idle}(\neg\psi, s) &= \neg\,\text{idle}(\psi) \\
\text{idle}(\psi_1 \wedge \psi_2, s) &= \text{idle}(\psi_1, s) \wedge \text{idle}(\psi_2, s) \\
\text{idle}(\mathbf{X}\psi, s) &= \text{idle}(\psi) \\
\text{idle}(\psi_1 \mathbf{U} \psi_2, s) &= \text{idle}(\psi_2)
\end{aligned}$$

or X-literals could be reached by progressing a formula ψ on any combination of states. That is, what can be said about the set

$$\{\Psi_p \mid \Psi_p = \text{CNF}(\text{un-}\mathbf{X}(\ldots \text{CNF}(\psi, p[1]) \ldots), \text{last}(p)), \forall p \in \left(2^{AP}\right)^{+}\}. \qquad (2.1)$$

To motivate the following definition, consider the formula $\psi = (a \vee \mathbf{X}a)\mathbf{U}(\mathbf{G}b \wedge \mathbf{G}c)$. $\text{CNF}(\psi, \{b, c\})$ is the CNF set

$$\begin{aligned}
\{&\{\mathbf{XG}b, \mathbf{X}a\}, \\
&\{\mathbf{XG}c, \mathbf{X}a\}, \\
&\{\mathbf{XG}b, \mathbf{X}(\psi)\}, \\
&\{\mathbf{XG}c, \mathbf{X}(\psi)\}\}
\end{aligned}$$

Note that within these clauses are multiple instances of each of only a few unique X-literals, and these X-literal are constructed from the subformulae of ψ such that the syntax tree of the subformula has a temporal operator as its root. From such a subformula, an X-literal is constructed by putting an $\mathbf{X}$ at the start.

This makes it more natural to refer to the set of X-literals that can appear in a CNF set reached by progression of some arbitrary formula ψ, which is denoted $\Sigma(\psi)$, and defined as the set of all subformulae of the negation normal form of ψ where the main operator in the subformula is a temporal operator. Formally, assume without loss of generality that ψ is in negation normal form, $\Sigma(\psi)$ is defined recursively as

$$\begin{aligned}
\Sigma(\top) = \Sigma(\bot) &\equiv \emptyset \\
\Sigma(a) = \Sigma(\neg a) &\equiv \emptyset \\
\Sigma(\psi_1 \wedge \psi_2) = \Sigma(\psi_1 \vee \psi_2) &\equiv \Sigma(\psi_1) \cup \Sigma(\psi_2) \\
\Sigma(\mathbf{X}\psi) &\equiv \{\mathbf{X}\psi\} \cup \Sigma(\psi) \\
\Sigma(\psi_1 \mathbf{U} \psi_2) &\equiv \{\mathbf{X}(\psi_1 \mathbf{U} \psi_2)\} \cup \Sigma(\psi_1) \cup \Sigma(\psi_2) \\
\Sigma(\psi_1 \mathbf{R} \psi_2) &\equiv \{\mathbf{X}(\psi_1 \mathbf{R} \psi_2)\} \cup \Sigma(\psi_1) \cup \Sigma(\psi_2)
\end{aligned}$$

Under this definition, the set of reachable CNF sets expressed in equation 2.1 is a

proper subset of $2^{2^{\Sigma(\psi)}}$, and while this is always a proper subset, there exists a family of LTL formulae, discussed in [Roşu and Havelund, 2005], for which $O(2^{2^{\sqrt{|\Sigma(\psi)|}}})$ unique formulae can be reached by progression, and any representation (CNF, formula or otherwise) must represent these uniquely to be correct.

2.3 Probabilistic Planning

The field of *automated planning* in artificial intelligence centres on finding plans for agents to traverse a problem to reach a goal state, having started at an initial state. A problem is modelled as a set of world states, and actions which transition the world to a new state and are associated with a cost. An optimal plan is one which reaches the goal while minimising the cost, which represents some resource, such as time or money.

Many practical problems involve uncertainty or stochastic behaviour, ranging from uncertainty about the location of a missing person, or whether a robotic gripper will fail to pick up an object. Stochastic behaviour is modelled by actions with multiple possible effects: when an agent performs an action, the world transitions to a new state chosen at random from a fixed probability distribution. One such model is called the Stochastic Shortest Path problem (SSP), and is the primary model used in this thesis.

2.3.1 Stochastic Shortest Path Problems

SSPs model planning problems as a set of states through which an agent moves, using actions, in order to reach one of several goal states. Actions have costs, and an optimal solution to an SSP is one which minimises the total cost to reach a goal from some initial state. Uncertainty is modelled in an SSP by associating with each action a probability distribution over outcomes. Apart from the inclusion of a goal state and initial state, this is very similar to a Markov Decision Process (MDP), and in fact several common classes of MDPs are special cases of SSPs. The definition of SSPs in this thesis primarily follows that of [Mausam and Kolobov, 2012], though several formal concepts are defined similarly to those in [Baumgartner et al., 2018], as they are better tailored to defining MO-PLTL SSP problems.

Definition 5. An SSP is a tuple $\mathcal{S} = \langle \mathrm{S}, s_0, \mathrm{G}, \mathrm{A}, \mathrm{T}, C \rangle$, where S is a finite set of states, $s_0 \in \mathrm{S}$ is the initial state, $\mathrm{G} \subset \mathrm{S}$ is the set of goal states, A is a set of actions, and $\mathrm{A}(s) \subseteq \mathrm{A}$ is the set of actions available in the state $s \in \mathrm{S}$. $\mathrm{A}(s_g) = \emptyset$ for all $s_g \in \mathrm{G}$. T is the transition function, where $\mathrm{T}(s' \mid s, \alpha)$ is the probability that an agent in state s, taking action $\alpha \in \mathrm{A}(s)$, will be in s' in the next step. The set of states s' such that $\mathrm{T}(s' \mid s, \alpha) > 0$ is referred to as the *outcomes* of action α from state s. Finally, C is a function $C : \mathrm{A} \to \mathbb{R}$ which defines the cost associated with each outcome of each action from each state, so $C(\alpha)$ is the cost resulting from taking action α.

SSPs are defined under an additional two constraints which cannot be succinctly defined here, but using notation and terminology defined below, these constraints are: a proper policy must exist; and for all improper policies π, $V^{\pi}(s_0) = \infty$. Equivalently, these constraints are that the goal can be reached from any state, and there are no 0 cost or negative cost loops in the SSP, respectively.

2.3.1.1 Formal Concepts

The solution to an SSP is a *policy*. Intuitively, a policy is a decision rule telling an agent what actions to take when. A policy is conditioned on something, i.e., the information which is used to decide what action to take, and the decision can be either stochastic or deterministic. The most general construct that a policy can be conditioned on is the complete history of the agent, which is a path through the state space S, and a deterministic choice is a random choice where the probability of one specific action is 1. The most general structure of a policy is therefore that of a *history-dependent stochastic* policy, which is a function $\pi : S^+ \times \mathrm{A} \to [0,1]$, defining a probability $\pi(p,\alpha)$ for each action α following the history p. As a probability distribution, it has the property

$$\forall p \in S^+, \sum_{\alpha \in \mathrm{A}(\mathrm{last}(p))} \pi(p,\alpha) = 1.$$

All policies are undefined for any path p with $\mathrm{last}(p) \in \mathrm{G}$, as no actions can be taken from a goal state. Naturally, defining a function over the set S^+ is hard in the general case, and evaluating such a policy is also hard, so policies typically use a more compact representation for the history.

Two other conditioning structures for policies are commonly studied for SSPs; *memoryless* or *stationary* policies, and *finite memory* policies. A finite memory policy is one which is dependent only on the current state s and a *mode*, which can take a finite set of values M, and is updated each time an agent reaches a new state. For a finite memory policy, $\pi(\langle s,m \rangle, \alpha)$ is the probability of an agent choosing action α when in state s and the mode has the value m. A stationary policy is a policy which depends only on the last state in the history, so if $\mathrm{last}(p) = \mathrm{last}(p')$ for some paths $p, p' \in S^+$, then $\pi(p,\alpha) = \pi(p',\alpha)$. For a stationary policy, $\pi(s,\alpha)$ is the probability that α is chosen from the state s, regardless of the path to that state.

For a *deterministic* policy, the second parameter to π is omitted, and instead a policy is a function $\pi : S^+ \to \mathrm{A}$ where $\pi(p)$ is the action taken from the history p, and the path parameter can be substituted for deterministic stationary and finite memory policies similarly to stochastic policies.

A *run* r of an SSP is a path $p \in S^+ \cup S^\omega$ annotated with the action taken from each state, $r = s_1 \overset{\alpha_1}{\to} s_2 \overset{\alpha_2}{\to} \ldots$. If r is finite, the final state $\mathrm{last}(r)$ is not annotated with an action. The notation for paths is used for runs, using subscript to denote actions or states, e.g., the ith state is denoted $r[i]_s$, and the ith action is $r[i]_\alpha$, but when clear from context, s_i and α_i are used for brevity and ease of reading. A run can be used as a path so, for example, $\pi(r,\alpha)$ is the probability that action α will be chosen at the end of the run r.

By definition, a run must have, for all $i < \ell(r)$, $\alpha_i \in \mathrm{A}(s_i)$ and $\mathrm{T}(s_{i+1} \mid s_i, \alpha_i) > 0$. Given a run r, the cost of r is denoted $C(r)$, and is the sum

$$C(r) \equiv \sum_{i=1}^{\ell(r)-1} C(\alpha_i).$$

A run r is said to be *possible* under a policy π if, for all $i < \ell(r)$, $\pi(r[\leq i]_s, \alpha_i) > 0$. The probability of a run under a policy π, denoted $\mathsf{Pr}^{\pi}(r)$, is the product of the probability all the transitions and choices in the run,

$$\mathsf{Pr}^{\pi}(r) \equiv \prod_{i=1}^{\ell(r)-1} \mathrm{T}(s_{i+1} \mid s_i, \alpha_i) \times \pi(r[\leq i]_s, \alpha_i).$$

The set of all runs starting from a given state s ($r[1]_s = s$) possible under a policy π is denoted $\mathrm{runs}(s, \pi)$, and the set of all runs r such that $\mathrm{last}(r) \in \mathrm{G}$ is denoted $\mathrm{Gruns}(s, \pi) \subseteq \mathrm{runs}(s, \pi)$.

A policy π is *proper* if

$$\sum_{r \in \mathrm{Gruns}(s,\pi)} \mathsf{Pr}^{\pi}(r) = 1$$

for all $s \in \mathrm{S} \setminus \mathrm{G}$. Recall that an SSP must have at least one proper policy. This means there are no dead ends in an SSP, because if there were, then from those states there would be no policy which can reach the goal, though there may be pairs of states such that there is no path between them.

Practical problems do, of course, have dead ends, and they can be dealt with in several ways. The typical way is to add to the SSP a 'give up' or a 'try again' action, representing a failure to solve the problem, with some large associated cost chosen given some domain knowledge about the problem. For example, a failure to pilot a Mars Rover to a specimen without damaging it would require a new specimen to be found. If the cost represents time, then the time it takes to find a new specimen could be estimated, which would be the cost of a 'try again' action. Alternately, there is some research into solving SSPs with dead ends, which can be found in [Kolobov et al., 2011; Steinmetz et al., 2016; Trevizan et al., 2017b], however this thesis assumes that all SSPs have no dead ends.

For some policy π, the expected cost of reaching a goal state from a state $s \in \mathrm{S}$ is denoted $V^{\pi}(s)$, computed as

$$\begin{aligned} V^{\pi}(s) &\equiv \mathbb{E}_{r \in \mathrm{Gruns}(s,\pi)} C(r) \\ &= \sum_{r \in \mathrm{Gruns}(s,\pi)} C(r) \times \mathsf{Pr}^{\pi}(r), \end{aligned}$$

sometimes referred to as the *value* of s under π. The objective for an SSP is to find a policy π that minimises $V^{\pi}(s_0)$. Let Π be the set of all proper policies, then an optimal policy π^* is a policy such that $V^{\pi^*}(s_0) \leq V^{\pi}(s_0)$ for all $\pi \in \Pi$. The optimal value function $V^{\pi^*}(s)$, is abbreviated as $V^*(s)$. It is well known that for any SSP, there exists an optimal policy which is deterministic and stationary, meaning that an optimal policy can be found while searching only through the set of stationary deterministic policies.

2.3.1.2 Solutions

The optimal policy for an SSP can be found in many ways, but most of them are derived from the Bellman equations:

$$V^*(s) = \min_{\alpha \in \mathrm{A}(s)} \left[\sum_{s' \in \mathrm{S}} \mathrm{T}\left(s' \mid s, \alpha\right) \times \left(C(\alpha) + V^*(s')\right) \right]$$

Which recursively define the value function as the cost to reach the next state plus the value of the next, and the best action in this state is dependent only on the value of the other states, not the history nor the actions taken in other states. Algorithms relying on the Bellman equations find the value function directly, searching through the value function space, so to speak. This is referred to as the primal space.

Most of the methods discussed in this thesis instead search in the *occupation measure dual space*. An occupation measure is a variable $x_{s,\alpha}$ which represents the expected number of times that an agent starting in the initial state s_0 takes action α from the state s. The set of all occupation measures for an SSP is

$$\mathrm{X} \equiv \{x_{s,\alpha} | s \in \mathrm{S} \setminus \mathrm{G}, \alpha \in \mathrm{A}(s)\}.$$

Given a valid assignment to X, a policy π and the value function at the initial state $V^{\mathrm{X}}(s_0)$ can be inferred. The stochastic stationary policy π defined according to X is:

$$\pi(s, \alpha) \equiv \frac{x_{s,\alpha}}{\sum_{\alpha' \in \mathrm{A}(s)} x_{s,\alpha'}},$$

and the cost function from the initial state derived from the occupation measures is:

$$V^{\mathrm{X}}(s_0) = \sum_{s \in \mathrm{S}, \alpha \in \mathrm{A}(s)} x_{s,\alpha} \times C(\alpha).$$

A valid assignment to X is one which represents the expected number of times each action is taken from each state, and the set of valid assignments to X can be characterised by observing that the expected number of times that the agent enters a given non-goal state $s \in \mathrm{S} \setminus \mathrm{G}$ must be equal to the number of times that they leave it; that the agent eventually reaches a goal state; and that the agent begins in the initial state. These observations can be encoded as a series of linear equations over X, which is presented in LP1, optimising for the minimum $V^{\mathrm{X}}(s_0)$.

Optimising a series of linear equations is a well studied problem, and such a problem is referred to as a linear program (LP), and each linear equation in an LP is referred to as a constraint. LPs can be solved by an off-the-shelf LP solver, such as Gurobi [Gurobi Optimization, 2019]. The methods for solving LPs are outside the scope of this thesis, but in general terms, there exist algorithms which solve LPs in time polynomial in the number of variables and size of the coefficients, though algorithms which have a poorer worst case complexity prove more effective in practice.

To make the function of LP1 clearer, two functions are introduced beforehand, in(s)

and $\text{out}(s)$, representing the flow into and out of the state s respectively. These are defined as:

$$\text{in}(s) \equiv \sum_{s' \in \text{S}, \alpha \in \text{A}(s')} x_{s',\alpha} \times \text{T}\left(s \mid s', \alpha\right)$$
$$\text{out}(s) \equiv \sum_{\alpha \in \text{A}(s)} x_{s,\alpha}$$

Note that $\text{in}(s)$ sums the agents coming from other states, and so does not count the agent entering at the initial state. Instead, LP1 includes a constraint specifically to encode this case. Below is LP1, and following that a brief justification for the constraints, as is the norm for all linear programs presented throughout this thesis.

$$\begin{aligned}
\min \quad & V^{\text{X}}(s_0) && \text{(LP1)} \\
\text{s.t.} \quad & x_{s,\alpha} \geq 0 && \forall x_{s,\alpha} \in \text{X} \quad \text{(C1)} \\
& \text{out}(s_0) - \text{in}(s_0) = 1 && \text{(C2)} \\
& \text{out}(s) - \text{in}(s) = 0 && \forall s \in \text{S} \setminus (\text{G} \cup \{s_0\}) \quad \text{(C3)} \\
& \sum_{s_g \in \text{G}} \text{in}(s_g) = 1 && \text{(C4)}
\end{aligned}$$

LP1 can be viewed as modelling the SSP as a *flow network*, with 1 unit of flow entering at the source (the initial state), travelling through the network, and leaving through the sink (all the goal states). This terminology will be used often throughout this thesis to refer to occupation measure linear program, and 'units of flow' is used interchangeably with the expected number of times the agent enters a state or takes an action. Later on, for problems with multiple agents, it is interchangeable with the expected number of times agents enter a state or take an action.

The constraints for LP1 can be justified as follows, skipping constraint C1 as it is a self evident declaration of the variables:

Flow Source (C2). In expectation, the agent leaves the initial state one more time than enters it *from other states.* Equivalently in terms of the flow network, the initial state behaves as a source for the flow network, creating exactly 1 unit of flow.

Flow Conservation (C3). This constraint encodes the concept that when the agent enters a non-goal state it must exit it again. Equivalently, the amount of flow entering a state is equal to the flow leaving it.

Sink (C4). This constraint specifies that eventually the agent must reach the goal, essentially forcing X to encode a proper policy. The goal state acts as a sink for the flow network, all the flow entering the network eventually reaches the goal and does not leave it.

2.3.2 Representations

Several representations exist for SSPs. SSPs are represented in this thesis using the *probabilistic SAS$^+$* formalism, which represents the state of SSPs as a set of variables that take multiple values. The most common languages for expressing SSPs, used in the International Conference on Automated Planning and Scheduling (ICAPS) International Planning Competitions, are the Probabilistic Planning Domain Definition Language (PPDDL), introduced by Younes and Littman [2004], and the Relational Dynamic Influence Diagram Language (RDDL), introduced by Sanner [2010]. The planning problems used in this thesis are written in PPDDL, so RDDL is omitted from this section.

2.3.2.1 SAS$^+$

The SAS$^+$ representation, introduced by Bäckström and Nebel [1995], is a factored formalism for planning problems, defining states in terms of a set of *variables* $\mathcal{V}$, which each take one of multiple values specified by their *domain* $\mathcal{D}_v$ for $v \in \mathcal{V}$. In the SAS$^+$ representation, a (partial) state s is an assignment $s[v] = d$ such that $d \in \mathcal{D}_v$ to each variable $\mathcal{V} \in \mathcal{V}_s$, where $\mathcal{V}_s \subseteq \mathcal{V}$. For the undefined variables $v \in \mathcal{V} \setminus \mathcal{V}_s$, the state value is denoted $s[v] = u$.

A total state s is one which has $\mathcal{V}_s = \mathcal{V}$, and for two states s and s', $s' \subseteq s$ denotes that $s[v] = s'[v]$ for all $v \in \mathcal{V}_{s'}$. In other words, $s' \subset s$ if s is defined on at least the same variables as s', and where they overlap they are the same. This can be thought of as s' being a partial state of s. The initial state of a SAS$^+$ problem is a total state, and the goal specification is a partial state s_g such that any total state s with $s_g \subseteq s$ is a goal state.

Trevizan et al. [2017a] extend SAS$^+$ for probabilistic actions and an associated cost function, meaning that under their definition, a probabilistic SAS$^+$ problem defines an SSP. In this SSP, $\mathrm{S} = \bigtimes_{v \in \mathcal{V}} \mathcal{D}_v$ is the set of all total states for the SAS$^+$ problem. Actions under this definition have effects which change the values of some variables, and preconditions that require certain variables to have certain values. Preconditions are partial states, and an action with the precondition p can only be taken in states s with $p \subseteq s$. Similarly, the effects of actions are partial states, and in the case that the effect e is applied to a state s, the resulting state s is s except updated so that $s'[v] = s[v]$ for $v \in \mathcal{V}_e$. The complete formal definition of a probabilistic SAS$^+$ problem is omitted because the intuition is sufficient for the definitions in this thesis, and the notation is not used.

2.3.2.2 PPDDL

PPDDL is an effects based language for specifying domains and problems, extended from the Planning Domain Definition Language (PDDL). That is, PPDDL can specify a type of problem (a domain) in the abstract, defining types, predicates, and actions. For example, consider the domain exploding blocks world (International Planning Competition 2008), in which there are blocks, which can be on top of other blocks, on the

table, detonated, destroyed and so on, and there are actions to pick up and put down blocks. PPDDL allows for specific instances of problems to be defined for this domain. For example, a problem in exploding blocks world might have 3 blocks, where block `a` is on block, `b` and blocks `b` and `c` are both on the table, and might have goal where block `c` is on `b`.

A given domain and problem together fully specify an SSP in a *factored* fashion, meaning that a complicated SSP is described using an exponentially smaller definition, and the structure of the SSP is available to a planner if it can use the extra information. On top of this, the capacity to make a single domain and many problems of different size but similar structure makes PPDDL quite convenient for defining standard benchmarks for planners.

2.3.3 Heuristic Search

Using a factored representation of an SSP, the size of the SSP is exponential in the length of the representation, meaning that it is typically computationally intractable to enumerate (expand) all the states of the SSP. In the worst case, an agent may have to visit every state in the SSP to reach the goal, so sometimes this is necessary. However, for many practical problems only a smaller (polynomial) subset of the state space is necessary for an agent to reach the goal.

Given this, an important task in planning then becomes finding the necessary states efficiently while ignoring the unnecessary states, and then finding the solution within this subset of the state space. This technique is referred to as heuristic search, because it uses a *heuristic function* to estimate the value function, and searches the factored state space in a manner informed by this estimate.

Various algorithms exist which perform heuristic search on SSPs, including LAO* [Hansen and Zilberstein, 2001] and LRTDP [Bonet and Geffner, 2003], sometimes classified as “find and revise” algorithms, so called because they maintain a table of the current value function, and repeatedly find and update states for which the bellman equation is violated; the primary distinguishing aspects being the order in which states are found.

2.3.3.1 Choosing Heuristics

Figure 2.2 shows a simple case of where heuristic search can be useful, using a deterministic SSP for simplicity's sake. In the presented SSP, there exists a policy shown in blue which reaches the goal s_g (represented as a square, to distinguish it as an absorbing state) in only two actions, but whether or not this is the best policy is not known for sure. $h(s_1)$ here represents the value of the heuristic function h at the state s_1. If the quality of the policy found is not an issue, then this is a success for heuristic search, as only 3 states need to be enumerated to find a policy, when the unexplored state space may be millions of states.

On the other hand, if an optimal policy is desired, it may be that this policy is optimal, but this can't be ascertained without either enumerating the whole state space or having some form of guarantee on h. Consider the case where h is guaranteed

to be an underestimate of the true cost to reach the goal, i.e., $h(s) \leq V^*(s)$. Given this guarantee, the known policy and the heuristic form upper and lower bounds on the value function, i.e., $h(s_1) \leq V^*(s_1) \leq C(\alpha)$ in this case. If $h(s_1) = C(\alpha)$, then the value function at state s_1 is $V^*(s_1) = C(\alpha)$, so no policy can do better than taking action α from s_1.

This lower bounding property is called *admissibility*. That is, a heuristic h is admissible if $h(s) \leq V^*(s)$ for all $s \in \mathrm{S}$. When guided by an admissible heuristic, heuristic search algorithms are guaranteed to find optimal solutions, despite potentially expanding only a fraction of the state space.

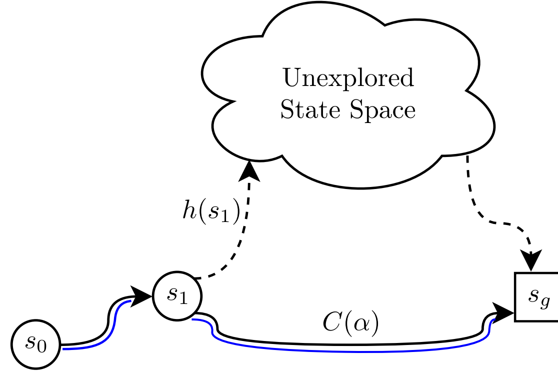


Figure 2.2: A schematic diagram of the heuristic search through a simple SSP.

Constructing an admissible heuristic is a difficult problem, and they are often based on some domain knowledge. For example, in a path finding problem (with Euclidean geometry), an effective heuristic might be the euclidean distance from the current state to the goal. This is a classical example of a *relaxation*; if the obstacles in the pathfinding problem are ignored, the optimal solution is the euclidean distance. Relaxation of parts of a problem is one of the standard ways of choosing a heuristic, as a relaxed problem is both easier to find a solution to, and the solution will have a smaller cost, making it admissible and efficient.

The euclidean distance heuristic is an example of a *domain specific* heuristic, as it only applies in pathfinding domains with euclidean geometry. This thesis focuses on *domain independent* heuristics, which do not relax the structure of the domain, they relax the structure of the problem representation. Domain independent heuristics are more general, as they can be used on any problem, so long as it is defined using the same representation.

2.3.3.2 All-Outcomes Determinisation

Heuristics for SSPs historically have relied on determinisation. This is a relaxation of the structure of the problem in which the stochastic properties of the problem are removed. The *all-outcomes determinisation* of an SSP is an identical SSP except that A is replaced with a set of determinised actions A'. To construct A', each action $\alpha \in \mathrm{A}$ is replaced with several actions $\alpha_{s_1,s'_1}, \ldots \alpha_{s_n,s'_n}$ for each pair $s, s' \in \mathrm{S}$ such that $\alpha \in \mathrm{A}(s)$ and $\mathrm{T}(s' \mid s, \alpha) > 0$. Each $\alpha_{s,s'}$ is an action with $\mathrm{T}(s' \mid s, \alpha_{s,s'}) = 1$, and the new set $\mathrm{A}'(s)$ of actions available in a state s is defined

$$\mathrm{A}'(s) = \{\alpha_{s,s'} \mid \alpha \in \mathrm{A}(s), s' \in \mathrm{S} \text{ s.t. } \mathrm{T}(s' \mid s, \alpha) > 0\}.$$

In English, the all-outcomes determinisation of an SSP is another SSP where agents

are free to choose which outcome every action results in. The cost of the optimal solution to the all-outcomes determinisation from some state s is an underestimate of the optimal value function $V^*(s)$, so could be used as an admissible heuristic. However, the all-outcomes determinisation of an SSP is a classical (deterministic) planning problem. Finding the optimal solution to a classical planning problem is PSPACE-hard, which is prohibitively difficult, so typically the determinisation is relaxed even further by using heuristics for classical planning problems.

The main focus of this thesis is, however, on heuristics that estimate the probability of certain events occurring in the execution of a policy. Obviously, removing all the probabilistic aspects from a problem makes this impossible, apart from the boolean classification 'it is possible' and 'it is not possible'.

2.3.3.3 Projection

There are a number of heuristics for classical planning which are based on the optimal solutions to projections. Intuitively, a projection of a probabilistic SAS$^+$ problem onto a subset of its variables $\mathcal{V}_p \subset \mathcal{V}$ is another probabilistic SAS$^+$ problem $\mathcal{S}_{\mathcal{V}_p}$ with all the other variables $\mathcal{V} \setminus \mathcal{V}_p$ removed. The complete formal definition for a projection as used in this thesis can be found in [Trevizan et al., 2017a], but only the necessary formal concepts and notation is provided here, and the rest is left to intuition.

Given a state s for a probabilistic SAS$^+$ problem, and a subset of the SAS$^+$ variables $\mathcal{V}_p \subset \mathcal{V}$, let $\text{proj}(s, \mathcal{V}_p)$ be the projection of s onto $\mathcal{V}_p$, with $\text{proj}(s, \mathcal{V}_p) \subseteq s$, and defined for the variables

$$\mathcal{V}_{\text{proj}(s,\mathcal{V}_p)} \equiv \mathcal{V}_p \cap \mathcal{V}_s.$$

Recall that the actions for a probabilistic SAS$^+$ problem have preconditions and effects, which are partial states. The actions for $\mathcal{S}_{\mathcal{V}_p}$ are constructed by projecting each precondition and effect onto $\mathcal{V}_p$. This may make leave them defined over the empty set of variables, in which case the action can be taken from any state or has no effects, respectively. Given the initial state s_0 for $\mathcal{S}$, the initial state for $\mathcal{S}_{\mathcal{V}_p}$ is $\text{proj}(s_0, \mathcal{V}_p)$.

For the goal specification s_g, the obvious approach would be to have the goal specification be $\text{proj}(s_g, \mathcal{V}_p)$, but a different definition is used in this thesis. Consider $\mathcal{V}_p$ such that $\mathcal{V}_{s_g} \setminus \mathcal{V}_p \neq \emptyset$, in this case there are non-goal states s in $\mathcal{S}_{\mathcal{V}_p}$ for which $\text{proj}(s_g, \mathcal{V}_p) \subseteq \text{proj}(s, \mathcal{V}_p)$, so projecting the goal as well would make these states goal states. As actions cannot be taken from goal states, this would not preserve the optimal policy when projected onto $\mathcal{V}_p$. The extreme case for this is when $\mathcal{V}_{s_g} \cap \mathcal{V}_p = \emptyset$, in which case every state would become the goal state, and no actions could be taken. Uses of projection in this thesis require that the policies for the original SAS$^+$ problem are preserved, so the direct approach is not applicable.

No state $s \in \bigtimes_{v \in \mathcal{V}_p} \mathcal{D}_v$ is a goal state in the projection, instead an artificial state g is introduced, along with an artificial action α_g, which transitions to g with probability 1. α_g is available only in states s such that $\text{proj}(s_g, \mathcal{V}_p) \subseteq s$. In this way, for a projection which has $\mathcal{V}_{s_g} \cap \mathcal{V}_p = \emptyset$, α_g is available in all states, but any policy valid in $\mathcal{S}$ can be projected into $\mathcal{S}_{\mathcal{V}_p}$.

Various heuristics for deterministic planning problems use several projections to

simplify the state space, making it possible to solve each projection using a simple forward search, and then use the optimal value functions of these projections as a heuristic. These heuristics are generally referred to as Pattern Database heuristics [Culberson and Schaeffer, 1998]. The primary advantage of using projections to define heuristics is that while the number of states in the probabilistic SAS^+ problem is exponential in the sizes of the variable domains, creating a set of separate projections can use an exponentially smaller number of states. In the case where the set of projections is a projection onto each variable, the number of states in the projections put together the sum of the domain sizes. Obviously, projections onto larger subsets of the variables leads to exponentially larger projections, so for efficiency's sake, smaller projections are generally preferred.

2.4 MO-PLTL Constrained Planning

It is natural in a practical problem to consider bounds under which a policy must operate. For example planning a route of travel between cities that minimises the travel time might also have the requirement that the expected fuel usage is within some bound, where this fuel cost is a secondary cost to the travel time. Another example is where the operator of a Mars rover may wish to guarantee that in the advent of a dust storm it can safely avoid having its solar panels covered in dust, by always being near shelter. The case where multiple costs are considered is the most common type of constraint on planning, and one model for such a problem is a Constrained SSP (C-SSP).

A C-SSP is an SSP with, instead of a single cost function C, a vector of n cost functions $C_1, C_2, \ldots, C_n$, each representing a different resource. Accordingly, there are n value functions for a given policy $V_1^\pi, \ldots, V_n^\pi$. The objective of a C-SSP is to reach the goal while minimising the *primary* value function $V_1^\pi(s_0)$, with a bound $V_i^\pi(s_0) < c_i$ on the expected cost of the policy for each other cost function. An optimal policy π^* for a C-SSP is one which, for all policies π that satisfy all the bounds, has $V_1^{\pi^*}(s_0) \leq V_1^\pi(s_0)$. However, this thesis considers a different type of constraint, specifically SSPs constrained by PLTL constraints.

A PLTL constraint ψ for a probabilistic SAS^+ problem $\mathcal{S}$ is an LTL formula ψ associated with a probability interval $z \subset [0, 1]$. ψ is defined over the propositional atoms $AP = \{(v = d) \mid v \in \mathcal{V}, d \in \mathcal{D}_v\}$, where $(v = d)$ is true in a state s if and only if $s[v] = d$. ψ requires that from the initial state s_0 of $\mathcal{S}$, any policy π must have the property that the probability of a run r being chosen such that $r \models_{\text{IE}} \psi$ is in the interval $z = [\underline{z}, \overline{z}]$.[1] This is denoted

$$\psi = \mathsf{Pr}\,(\psi) \in z,$$

Given a policy, the probability that ψ is satisfied from some state s is denoted $\mathsf{Pr}^\pi\,(\psi \mid s)$,

[1]Note here that r is being used as a path. In reality $r \models_{\text{IE}} \psi$ is short hand for $p \models_{\text{IE}} \psi$, where p is the path represented by r.

and can be computed as

$$\mathsf{Pr}^{\pi}\left(\psi \mid s\right) \equiv \sum_{\substack{r \in \text{Gruns}(s,\pi): \\ r \models_{\text{IE}} \psi}} \mathsf{Pr}^{\pi}\left(r\right).$$

Hence the constraint ψ requires that all policies π for the constrained SSP must have the property $\underline{z} \leq \mathsf{Pr}^{\pi}\left(\psi \mid s_0\right) \leq \overline{z}$. For the probability from the initial state, the state is omitted, i.e., $\mathsf{Pr}^{\pi}\left(\psi\right) \equiv \mathsf{Pr}^{\pi}\left(\psi \mid s_0\right)$

A multi-objective linear temporal logic (MO-PLTL) constraint is a conjunction of PLTL constraints. That is, a MO-PLTL constraint ϕ is a set of PLTL constraints $\{\psi_1, \ldots, \psi_n\}$ such that any policy π for an SSP constrained by ϕ must have the property $\mathsf{Pr}^{\pi}\left(\psi_i\right) \in z_i$ for all $1 \leq i \leq n$. An SSP constrained in such a way is referred to as a *MO-PLTL SSP problem*.

The set of policies for a MO-PLTL SSP problem $\mathcal{S}_{\phi}$ is $\Pi_{\phi} \subseteq \Pi$, defined as the set of all policies π for $\mathcal{S}_{\phi}$ such that $\mathsf{Pr}^{\pi}\left(\psi_i\right) \in z_i$ for all $1 \leq i \leq n$. An optimal policy for $\mathcal{S}_{\phi}$ is a policy $\pi^* \in \Pi_{\phi}$ such that $V^{\pi^*}(s_0) \leq V^{\pi}(s_0)$ for all policies $\pi \in \Pi_{\phi}$. Because of the constrained nature of MO-PLTL SSP problems, and the path dependent semantics of LTL, an optimal policy for a MO-PLTL SSP problem may be need to be *stochastic* and *finite memory*, respectively [Kwiatkowska and Parker, 2013].

A simple example for why this is the case is a (obviously contrived) constraint that, with probability $[0.5, 1]$, $\mathbf{X}(v_1 = \mathsf{1})$ must hold. Consider a SAS$^+$ problem where $s_0[v_1] = \mathsf{1}$, and there are two actions, α_1, which maintains the current state for one step, and α_2, which transitions to a goal s_1 in which $s_1[v_1] = \mathsf{2}$. The formula can be satisfied by a policy which chooses α_1 then α_2, but this particular policy chooses two different actions in the same state, hence requiring a finite memory policy. This path, however, costs more than the optimal path to the goal, α_2. An optimal policy chooses between these paths at random, satisfying $\mathbf{X}(v_1 = \mathsf{1})$ by taking the first path with probability 0.5, and the cheaper path with probability 0.5.

A finite memory policy for a MO-PLTL SSP problem can use one of several modes, though the mode of interest in this thesis is the progression mode [Baumgartner et al., 2018], which uses as the mode value the progressed CNF set generated up to that state. The progression mode for MO-PLTL SSP problems has a tuple of CNF sets, one CNF set for each constraint in ϕ. As a result, the finite set of values that the progression mode can take is $M = \bigtimes_{\psi_i \in \phi} 2^{2^{\Sigma(\psi_i)}}$.

Finite memory policies can be converted into stationary policies by compiling the mode into the SSP itself. A product C-SSP is a constrained SSP where each state is pair $s^{\times} = \langle s, m \rangle$, referred to as a product state. The state space is then $\mathrm{S}^{\times} = \mathrm{S} \times M$, and actions transition to new product states dependent on both the transition function and the mode update function $\mathrm{next}(s, m)$, which updates the mode each time a new state is observed. In a product C-SSP, the optimal policy is stationary, but may still be stochastic to satisfy the constraints.

In the intersection of constrained and heuristic planning, heuristics estimate the optimal value function for the product C-SSP, $V^*(s^{\times})$. This means a heuristic for a product C-SSP is a function $h : \mathrm{S}^{\times} \to \mathbb{R}$, dependent on both the mode and the SSP state. To effectively design a domain independent heuristic for constrained planning,

it must take into account both the SSP state and the mode, or at the least, several heuristics should be used which take account of each, if independently. Heuristic search is especially important for product C-SSPs, because S is already exponential in size, and the set of values M for some modes can be very large as well. For instance, the progression mode is double exponential in the length of the formulae in ϕ.

> *The aim of this thesis is to introduce a novel heuristic for product C-SSPs constructed from MO-PLTL SSP problems, designed for the progression mode.*

For efficiency reasons, heuristics for MO-PLTL SSP problems consider only one constraint, and an instance of the heuristic is computed for each constraint. As such, the input to a heuristic for MO-PLTL SSP problems is an *augmented state*, which is a pair $\langle s, m_i \rangle$. An augmented state can be constructed trivially from a product state $s^\times = \langle s, m_1, m_2, \dots, m_n \rangle$. In this thesis, an augmented state is always constructed from the progression mode, so it is denoted $\langle s, \Psi \rangle$.

2.5 Summary

This chapter introduced linear temporal logic, an extension of propositional logic to define the properties of infinite paths, and probabilistic planning, the problem of finding a plan which reaches the goal of a problem in a stochastic environment. The intersection of these is the MO-PLTL SSP problem, a planning problem where any valid policy satisfies a set of PLTL constraints, which state that certain LTL formulae are satisfied with some given probability.

So that LTL can be defined for planning, it is extended with infinite extension semantics, where the final state in a path is repeated indefinitely. Given this, it is defined on finite paths, so it is reasonable to apply an iterative rewriting process to it, referred to as progression. Progression in this thesis is defined on CNF sets, which are sets of sets of X-literals, where every X-literal is a temporal formula starting with the next operator, and a CNF set represents an LTL formula in conjunctive normal form.

Probabilistic planning centres around the Stochastic Shortest Path problem, which is a shortest path problem in which actions have stochastic effects. SSPs are represented using SAS$^+$, which is a factored representation with multi-value variables, which permits projection onto a subset of these variables. Because the full state space of an SSP is exponential in the size of the SAS$^+$ representation, it is computationally intractable to enumerate all the states, so a technique called heuristic search is applied to explore the state space intelligently until a sufficient subset of it has been explored to find a solution. Projection can be used to construct heuristics which keep the probabilistic properties of the underlying problem, as an alternative to the more common determinisation heuristics.

SSPs constrained by multiple PLTL constraints, referred to as MO-PLTL SSP problems, can be represented as a product C-SSP, which combines a mode for the PLTL constraints with the states for the SSP. The resulting state space is the product

of the set of mode values and the state space of the SSP. Progression can be used as a mode, where the mode takes the value of a tuple of CNF sets, one for each constraint in the MO-PLTL constraint. A domain independent heuristic for a product C-SSP uses the mode and the state to estimate the cost of reaching the goal under the constraints.

In the next chapter, a brief summary is provided of other research in the intersection between planning and LTL. Of particular note are the PRISM model checker, and PLTL-dual, which both provide a solution to the MO-PLTL SSP problem.

Chapter 3

Related Work

The field of LTL in planning has a long history and this thesis is of course not the first to address PLTL as applied to planning, let alone LTL. This chapter details some of the work that has been done in the intersection of LTL and planning, organised as follows:

Section 3.1 details the ways in which LTL has been integrated into planning, primarily as temporally extended goals.

Section 3.2 discusses model checking and strategy synthesis, which focus on LTL rather than planning, but various work in that field has involved planning, and the methods are quite reminiscent of those in section 3.1.

Section 3.3 presents the state-of-the-art planner PLTL-dual, which solves MO-PLTL SSP problems using heuristic search.

3.1 LTL In Planning

For two decades now, the planning community has known that constraining the temporal properties of plans allows planning problems to be much more expressive. These constraints have often been referred to as *temporally extended goals*, as the constraints can also be viewed as objectives, or can be the sole objective of a problem.

3.1.1 Control Knowledge

LTL was first included in planning as control knowledge by Bacchus and Kabanza [1996]. Control knowledge is domain specific information which aids a planner in finding a solution to any problem in that domain. Bacchus and Kabanza [1996] defined control knowledge in terms of first order LTL, where instead of adding constraints which policies must satisfy, the provided LTL formula encode properties that good quality policies have. They implemented this in a planner called TLPlan, which uses progression to update the formulae synchronously with the policy, and paths that would violate the LTL formulae are abandoned. With good quality control knowledge, TLPlan was able to complete problems up to an order of magnitude larger than planners without control knowledge.

TLPlan was later succeeded by TALPlaner [Doherty and Kvarnstram, 2001], which used a set of logics called *Temporal Action Logics* (TAC). These differ from LTL

in that TAL was designed to refer to actions and change in an environment, where LTL was designed outside the field of planning, and has seen use in several fields. TALPlaner was shown to perform much better than TLPlan.

In their later work, Bacchus and Kabanza [1998] introduced the concept of temporally extended goals. They extended TLPlan to solve planning problems where, instead of having a goal state, the goal is defined as the satisfaction of a given LTL formula. To this end, they introduced infinite extension semantics and the idle operator.

Notably, their approach was simply to prune from the search space the paths which violated this formula, but their approach did not guide the search in any other way towards achieving the goal. This works fine for safety goals, e.g., *a robot must not do something which may cause it to drop an explosive*, because the formula is immediately violated when the robot does do such a thing. On the other hand, eventualities, e.g., *the robot must eventually clean out every room*, can only be violated by an infinite path in which that never happens. For goals like this, TLPlan must resort to blind search.

3.1.2 LTL Compilation

Guiding a planner towards a temporally extended goal requires some form of heuristic search to scale to large problem instances. This was recognised and a common approach in research into temporally extended goals is to use classical planning heuristics for LTL. This is done by compiling the LTL formula into the planning problem, i.e., constructing another planning problem with a final goal state instead of a temporally extended goal, where the final goal state is equivalent to satisfying the LTL formula.

The techniques involving compilation often rely on conversion to Non-deterministic Büchi automata (NBA). NBAs are ω-automata, which define languages of infinite words[1], and for any LTL formula ψ, there exists an NBA which accepts a path p if and only if $p \models \psi$. An NBA constructed from a given formula, in the worst case, has a number of states exponential in the length of the formula. For LTL on finite paths, NBAs are treated as non-deterministic finite automata (NFA).

The non-determinism of an NFA is *clairvoyant*, i.e., non-deterministic transitions are chosen such that if a word can be accepted it will be. Equivalently, an NFA accepts a word if there is any path which accepts it. To represent this, a common technique is that when transitioning, an NFA goes to *all* possible states simultaneously. This is called powerset construction, or on-the-fly determinisation. Under this interpretation, an NFA is in multiple states at once, and if any of those states are accepting states after observing a word, it accepts that word.

Edelkamp [2006] and Baier and Mcilraith [2006] provide compilations of temporally extended goals into final state goals in classical planning. Both approaches involve the use of NFAs, and can output the resulting planning problem in PDDL. Baier and Mcilraith [2006] make use of finite path first order LTL (f-FOLTL), allowing formulae like *all servers must always be connected to some other server* to be represented compactly. They construct parameterised NFAs, which are a compact form of NBA, where the

[1] The terminology *word* is typically used for automata, as they define *languages*, but so long as the alphabet is 2^{AP}, this is equivalent to paths for LTL as presented in this thesis.

parametrisation behaves similarly to quantifiers for simultaneously representing all the objects in the domain. After constructing a parameterised NFA, they compile it into the planning problem as axioms, which are essentially actions that are taken for free as soon as they are available. These axioms represent the movement between the states of the NBA, and a goal state requires that the NFA has reached an accepting state.

On the other hand, the work by Edelkamp [2006] uses propositional LTL. This work focuses on PDDL3, which includes support for temporally extended goals and *preferences*. The language provides only a limited set of temporally extended goal structures, which are strictly less expressive than LTL. Preferences are measures of a plan's *quality*, and achieving them is weighted by a cost in the primary metric. Hence a plan which does not achieve a preference is better than a much more expensive plan which does achieve the preference, provided the difference is larger than the cost associated with the preference. Temporally extended preferences in PDDL3 are preferences defined using the same structures as temporally extended goals.

To deal with temporally extended goals and preferences, Edelkamp [2006] converts them to LTL, and constructs an NBA for the resulting formula. The state of the automata is represented in the planning problem, and he adds actions to transition between states of the automata, as well as flags to force the automata and the planning problem to stay synchronised. He adds a flag for each preference to state whether that preference is being achieved, and this flag is set to false by an action which costs as much as it does to fail to achieve the preference.

Both of these works output a classical planning problem, for which there are many well known heuristics. In this way, any classical heuristic planner can search in a guided fashion towards the achievement of the temporally extended goals and preferences.

In a later work, Baier et al. [2009] address temporally extended preferences in a different manner, compiling the temporally extended preferences to parameterised NFAs similarly to [Baier and Mcilraith, 2006]. They add a flag for each temporally extended preference which is equivalent to the satisfaction of that preference. Hence this flag can be used as a simple preference in the final state. For simple preferences, they present a series of heuristics which can be used to estimate the achievable value from a state taking into account the preferences. Notably, their heuristics are not admissible and therefore do not necessarily generate optimal plans.

The approach of compilation has been applied also to non-deterministic planning by Camacho et al. [2017], in an almost identical way to [Baier and Mcilraith, 2006], where they construct an NFA and encode it in the effects of actions in the non-deterministic planning problem. Camacho et al. [2017] also present a compilation of LTL for infinite paths into non-deterministic planning. This is done with synchronisation actions, and to represent the acceptance of an infinite path, the problem is constructed such that at some point a solution must demonstrate a loop while in an accepting state of the NBA, proving that there is an infinite path which satisfies the LTL.

As well as in the planning community, there is plenty of work in the area of verification and synthesis for LTL by the formal methods community. The approaches used by this community also generally rely on automata.

3.2 LTL Strategy Synthesis

The question addressed by a solution to a MO-PLTL SSP problem is "how can certain properties be reliably achieved while also reaching a goal with a minimum cost." Model checking and strategy synthesis are where the first half of this question originates. Intuitively, model checking is the problem of "does a system (e.g., a strategy) satisfy certain properties," and strategy synthesis is the question "how can certain properties be satisfied?" These are both done primarily by constructing automata and analysing its properties.

3.2.1 Planning as Strategy Synthesis

Intuitively, LTL synthesis is defined as a sort of game, in which the agent controls one subset of the propositional variables $\mathcal{Y} \subset AP$ and the environment controls a disjoint subset $\mathcal{X} \subset AP$ with $\mathcal{X} \cup \mathcal{Y} = AP$. These variables are assigned values in sequence, first the environment chooses Y_1, then in response the agent X_1, and play alternates. The agent 'wins' the game if the infinite word $(X_1, Y_1), (X_2, Y_2), \ldots$ constructed by combining these assignments satisfies a given LTL formula ψ. LTL is useful for automatically synthesising a controller from its specifications, or debugging those specifications.

Perhaps the most closely related part of strategy synthesis for this thesis is that of synthesis for LTL on finite paths. This problem is addressed by De Giacomo and Vardi [2015], and their approach is which is to construct a deterministic finite automaton (DFA) with the property that any word is accepted by the automaton if and only if it satisfies a given LTL_f formula. Constructing a DFA requires first the construction of an NFA, followed by the determinisation of this automata. Both steps are exponential in the input, so the resulting DFA may have an doubly exponential number of states in the length of the formula. They perform a sort of reachability analysis on the resulting DFA, identifying states which can be transitioned to in one step no matter the environment's choice of assignment to $\mathcal{X}$, and iteratively identifying states which can reach a goal state in 1 step, then 2 steps and so on until convergence.

Planning seems very similar to strategy synthesis for LTL on finite paths, and in fact the problems can be cast as each other. Camacho et al. [2018] observe this and provide a reduction of non-deterministic planning to strategy synthesis for LTL_f and vice versa. Their paper focusses on applying the reduction from synthesis to planning to use state-of-the-art planners to synthesise a strategy, or alternatively find a certificate that there is no such strategy. Their reduction from synthesis to planning has flags in the planning problem which force it to alternate actions representing the environment playing and the agent assigning variables. The planning problem has variables representing the states of an NFA constructed from the given LTL_f formula, and when the agent makes an assignment these variables are updated based on the choice of the agent and the environment.

Notably, Camacho et al. [2018] make use of a technique where they split large automata into several smaller automata, where a path satisfies the original formula if

and only if that path is accepted by all automata. This allows them to reduce the size of very large automata in many cases. They found that splitting the automata could result in much better performance, but splitting the automata into too many parts resulted in worse performance than not splitting at all.

3.2.2 PRISM Model Checker

The PRISM model checker [Kwiatkowska et al., 2011] is a tool developed for a variety of model checking and strategy synthesis problems. Of interest in this thesis are the techniques implemented in PRISM for strategy synthesis on MDPs [Kwiatkowska and Parker, 2013], particularly strategy synthesis for PLTL in MDPs. For a single PLTL constraint, PRISM constructs a *deterministic Rabin automata* (DRA) [Rabin, 1969] from the LTL formula in the PLTL constraint and constructs the synchronous product of the DRA with the MDP.

A DRA is a type of deterministic automata with the same expressive power as an NBA. Like the DFA in the work of De Giacomo and Vardi [2015], DRAs in the worst case have size double exponential in length of the original LTL formula. The reason DRAs are used instead of using on-the-fly determinisation is because the synchronous product of the two is made up-front, as the approach used by Kwiatkowska and Parker [2013] requires a holistic analysis of this product.

The result of constructing the synchronous product of the two is a new MDP $\mathcal{M}^\times$ where the states are pairs $\langle s, q \rangle$, s is a state of the original MDP and q is a state of the DRA. In $\mathcal{M}^\times$, PRISM identifies the *accepting end components*, that is, *sub-MDPs* inside $\mathcal{M}^\times$ in which every state is reachable from every other state. As it is an MDP, all the actions available in the MDP stay in it, actions from $\mathcal{M}^\times$ which would leave the sub-MDP are removed, and [Baier and Katoen, 2008, pg. 878] provides an algorithm for identifying these end components. An end component is accepting depending on the accepting condition of the DRA, as states in an end component are visited an infinite number of times.

Once the accepting end components are identified, the approach in [Kwiatkowska and Parker, 2013] is to perform value or policy iteration to find the maximum probability of reaching these accepting components. If the best policy reaches accepting end components with a probability outside the bounds of the PLTL constraint, then it is known that there is no policy which satisfies the PLTL constraint. Otherwise, the policy can be extracted after value/policy iteration is completed.

This approach is extended to multiple PLTL constraints (MO-PLTL) on an MDP in a very straightforward manner. For each LTL formula in the PLTL constraints, a DRA is constructed and the synchronous product of all the DRAs and the MDP is computed. In this product MDP, the accepting end components for each constraint are identified, and to allow for the trade-off between objectives, an LP is constructed which, like LP1, represents the expected number of times actions are used in various states and there is an LP constraint for each PLTL constraint that the flow entering accepting components for that PLTL constraint should be within the given bound.

PRISM implements this algorithm also for the case of SSPs, where it requires very

little adaptation, and thus PRISM provides a solution to MO-PLTL SSP problems.

3.3 PLTL-dual

The state-of-the-art for solving MO-PLTL SSP problems is the PLTL-dual algorithm, presented by Baumgartner et al. [2018]. This algorithm is a heuristic search algorithm employing a series of LPs encompassing an increasing subset of the state space. PLTL-dual integrates heuristics in an innovative way, computing the heuristic value of every state on the fringe of the explored state space simultaneously.

3.3.1 Algorithm

To find a solution to an MO-PLTL SSP problem $\mathcal{S}_\phi$, PLTL-dual constructs the product C-SSP of the SSP $\mathcal{S}$ with a mode for each PLTL constraint in ϕ. Baumgartner et al. [2018] define two modes for PLTL constraints, the progression mode and the NBA mode. The progression mode performs progression on the formula of each PLTL constraint, so the mode value for a state is a tuple of CNF sets. The NBA mode constructs an NBA for each PLTL constraint, and performs on-the-fly determinisation on them. The mode values of the NBA mode for each constraint are a sets of states in each NBA, and the separate constraints are collected as a tuple. For each constraint, each mode defines a subset of the product state space as accepting that constraint: for the progression mode, states where that constraint has $\text{idle}(s, \Psi_i) = \top$, and for the NBA mode, states where staying in s forever would satisfy the Büchi acceptance condition from the current set of states.

Given a subset *envelope* $\subset \mathrm{S}^\times$ of states explored so far, PLTL-dual constructs an LP reminiscent of LP1 which represents flow through *envelope*. Let $fringe$ be the states on the edge of envelope which have not been expanded, and so no actions are available. Instead of having occupation measures to represent flow out of $fringe$, flow is redirected to a series of LPs which compute an array of heuristics from the fringe states simultaneously. Included in this LP is a constraint that the amount of flow reaching the accepting set for each PLTL constraint is within the bound of that constraint. A solution to this LP simultaneously computes a policy in *envelope* and computes the heuristic value of fringe states which this policy reaches. The heuristic value of other states is not necessarily needed, but may be still be implicitly computed during the process of solving the LP.

The outline of PLTL-dual is presented informally in algorithm 2, finding a solution to the MO-PLTL SSP problem $\mathcal{S}_\phi$. The algorithm starts by defining *envelope* as the initial state $s_0^\times = \langle s_0, m_{1_0}, m_{2_0}, \ldots, m_{n_0} \rangle$ of the product C-SSP constructed from $\mathcal{S}_\phi$ under any mode. Following this, it finds the nodes on the fringe which were reached expands every node on the fringe which was reached by the most recently found policy, denoted $fringe_R$. It expands each of these states, extending the LP to include the newly found neighbours of states in $fringe_R$, and solves the resulting LP. If the solution found by the LP has no flow entering the fringe states, then the solution defines a policy π which does not leave *envelope*, so is defined for all states reachable under π. The policy

Algorithm 2 An algorithm to find a policy for a MO-PLTL SSP problem $\mathcal{S}_\phi$ using heuristic search.

```
 1: procedure PLTL-DUAL(S_ϕ)
 2:     envelope ← {⟨s_0, m_{1_0}, m_{2_0}, ..., m_{n_0}⟩}        ▷ The set of explored states
 3:     fringe ← {⟨s_0, m_{1_0}, m_{2_0}, ..., m_{n_0}⟩}   ▷ The states on the fringe of the explored set
 4:     fringe_R ← {⟨s_0, m_{1_0}, m_{2_0}, ..., m_{n_0}⟩} ▷ The fringe states reached in the current policy
 5:     while fringe ≠ ∅ do
 6:         Let new be the states reachable from fringe_R in one action
 7:         envelope ← envelope ∪ new
 8:         fringe ← (fringe \ fringe_R) ∪ (new \ G)
 9:         Solve LP for envelope   ▷ Find the optimal policy for this subset of the state space
10:         fringe_R ← states in fringe which have flow into them
11:     end while
12:     Extract π from the solution to the last LP
13:                                     ▷ π does not leave envelope, as fringe_R = ∅
14: end procedure
```

Output: π is an optimal strategy for $\mathcal{S}_\phi$, satisfying all the constraints in ϕ.

π is a solution to $\mathcal{S}_\phi$.

3.3.2 Heuristics

The heuristics for PLTL-dual are in two sets, a set of *projection occupation measure* heuristics and a set of heuristics for the PLTL constraints. The projection occupation measure heuristic h^{pom} [Trevizan et al., 2017a], constructs a set of projections which are solved synchronously, with a constraint between them so they influence each other.

Given a probabilistic SAS$^+$ problem $\mathcal{S}$, the projection occupation measure heuristic h^{pom} constructs for each variable $v \in \mathcal{V}$ the projection onto $\{v\}$, denoted $\mathcal{S}_v$, and for each of these projections, the set of occupation measures $\mathrm{X}_{\mathcal{S}_v}$ for the associated SSP. These occupation measures are denoted $x^v_{d,\alpha}$ for each action α and $d \in \mathcal{D}_v \cup \{g\}$. The occupation measure $x^v_{d,\alpha}$ denotes the expected number of times that action α is taken from the state $s[v] = d$ in the projection $\mathcal{S}_v$.

Given these projections, h^{pom} constructs an LP with flow constraints on $\mathrm{X}_{\mathcal{S}_v}$ (similarly to LP1) for each $v \in \mathcal{V}$, and adds to the resulting LP a set of tying constraints, defined as:

$$\sum_{d \in \mathcal{D}_v \cup \{g\}} x^v_{d,\alpha} = \sum_{d \in \mathcal{D}_{v'} \cup \{g\}} x^{v'}_{d,\alpha} \quad \forall v, v' \in \mathcal{V}, \alpha \in \mathrm{A}$$

where A is the set of all actions in $\mathcal{S}$. These constraints need not be defined between every pair of variables, one reasonable implementation is to define them in a chain, or from one specific variable to all others. Tying constraints require that the total number of times each action is taken in one projection is equal to the number of times that action is taken in all other projections, but there is no restriction on the order they are taken in each projection. In this way, if an action is necessary in one projection, it may influence other projections, forcing them to take associated actions.

Including h^{pom} in the LP for PLTL-dual is as simple as having the amount of flow

reaching any fringe state $\langle s, m_1, \ldots, m_n \rangle$ enter each projection in h^{pom} from the state $\text{proj}(s, \{v\})$, where v is the variable for that projection. To incorporate the heuristic estimate, a projection is chosen (because of the tying constraints, it doesn't matter which) and the cost of actions in that projection is included in the objective. Hence the LP minimises the combined cost of the policy in *envelope* and also in the heuristic.

In PLTL-dual, it is assumed without loss of generality that all PLTL constraints are of the form $\psi_i = \mathsf{Pr}(\psi_i) \geq \underline{z}_i$. If a PLTL heuristic has an upper bound this can be removed by adding another constraint $\psi_i' = \mathsf{Pr}(\neg\psi_i) \geq \overline{z}_i$. Given this, an *admissible* heuristic h^{ψ_i} for a PLTL constraint ψ_i is one that computes an upper bound on the probability that the LTL formula for ψ_i is satisfied from a given product state $s^\times \in \mathrm{S}^\times$ under an optimal policy for the given MO-PLTL SSP problem.

There are two PLTL heuristics presented in [Baumgartner et al., 2018], the first is the *trivial heuristic* derived from the definition of an admissible heuristic, which simply classes all flow which reaches fringe states as accepting, except when flow reaches a state for which the mode trivially has 0 probability of reaching to an accepting value, i.e., $m_i = \bot$ for the progression mode, and $m_i = \emptyset$ for the NBA mode.

The second heuristic is the NBA heuristic h^{BA}, which treats the NBA for constraint ψ_i as an SSP, and finds the shortest path to an accepting state. Actions in this SSP $\mathcal{S}_{\psi_i}$ are constructed from the set of actions A in the original MO-PLTL SSP problem. For each action $\alpha \in \mathrm{A}$, a set of new actions are constructed in $\mathcal{S}_{\psi_i}$ for each combination of transitions in the NBA that each effect in the action could possibly cause. The NBA heuristic h^{BA} can be computed by constructing a flow network for $\mathcal{S}_{\psi_i}$, with the added complication that powerset construction leads to the initial state for h^{BA} to be a set of states which must be non-deterministically chosen from, so the choice of initial state is left up to the LP solver, allowing it to make the best possible choice of initial state. Integrating this heuristic into the LP for PLTL-dual is relatively straightforward: any flow reaching a fringe state $\langle s, m_1, \ldots, m_n \rangle$ enters $\mathcal{S}_{\psi_i}$ non-deterministically among the states in m_i.

To integrate PLTL heuristics into PLTL-dual, the flow being accepted in the PLTL heuristic and flow reaching accepting goal states within *envelope* is summed up and constrained so that the sum of accepting flow for each PLTL constraint must satisfy that constraint. Furthermore, tying constraints are extended to the NBA heuristic, so that actions that are necessary for the constraints are performed in the projection heuristics, and hence influence the cost estimate also.

3.4 Summary

This chapter introduced other approaches which investigate combined application of LTL or PLTL and planning. LTL has been used in planning for control knowledge and temporally extended goals or preferences. Control knowledge applications typically used progression to prune the search space, improving the efficiency of the solver, while applications with temporally extended goals typically use a transformation from LTL formulae to non-deterministic finite automata to represent a temporal property which behaves as the goal of the planning problem.

Closely related to planning with temporally extended goals is the topic of synthesis for LTL on finite paths, which is finding a strategy that is guaranteed to satisfy a given LTL formula. Methods for this either construct a deterministic finite automaton with on-the-fly determinisation, or a non-deterministic finite automaton. Strategy synthesis has also been shown to be equivalent to non-deterministic planning. Of note is the problem of policy synthesis for MDPs with multiple PLTL constraints, which is solved by constructing the synchronised product of the MDP with a deterministic Rabin automata for each PLTL constraint. Solutions to this problem are easily adapted to MO-PLTL SSP problems.

The state-of-the-art in solving MO-PLTL SSP problems is PLTL-dual, which performs heuristic search by constructing an LP for an iteratively expanding subset of the state space, which is constructed by taking the product of the given SSP with the values of some mode for the PLTL constraints. PLTL-dual integrates heuristics in an innovative way such that they are calculated synchronously with each other, allowing them to influence each other and improve the overall estimate. One non-trivial heuristic exists for the PLTL constraints in PLTL-dual, which relies on NBAs and the NBA mode.

In the next chapter, projection is extended to PLTL constraints, making it possible to consider the satisfaction of a PLTL constraint within a projection. Projection of PLTL is analysed and cases are found where the projection trivialises the PLTL constraint, so an algorithm is presented which constructs a series of projections safely without any individual projection having a trivial constraint.

Chapter 4

LTL Projection

It is a proven strategy to relax a SAS$^+$ problem by projecting onto one or more subsets of the variables in the domain as defined in section 2.3.3.3, and vitally for this thesis, it maintains the probabilistic properties of the underlying problem. The projection of a planning problem onto a subset of variables $\mathcal{V}_p$ is well defined, but to use projection in the context of LTL constraints, projections of LTL formulae must be defined.

The obvious definition for projection onto a set of variables $\mathcal{V}_p$ is that for every variable $v \notin \mathcal{V}_p$ any valid assignment $(v = d)$ can be made in any state. Given this, the CNF$(\cdot, \cdot)$ operator can no longer completely simplify formulae based on the interpretation $s[v]$ for $v \in \mathcal{V}_p$. Instead, the resulting CNF set Ψ contains clauses which include non-temporal literals $(v = d)$ or $\neg(v = d)$ as well as the temporal X-literals.

The notion of LTL projection used in this thesis does *not* use this definition. The inclusion of non-temporal literals increases the number of unique formulae in augmented states, and would mean that at every state, a satisfiability problem should be solved to determine if the CNF should be reduced to $\bot$. Similarly, acceptance would be equivalent to the solution of a satisfiability problem, as opposed to the polynomial algorithm idle$(\cdot, \cdot)$ in [Bacchus and Kabanza, 1998]. Instead a further relaxation is performed where every instance of a proposition $(v = d)$ is assumed to be independent.

In this chapter, this assumption is first detailed in section 4.1. In section 4.2, a method for doing this projection is presented, and considerations for this projection are raised and addressed in section 4.3, mainly addressing which variables are appropriate to project onto, given a certain formula.

4.1 Proposition Independence Assumption

Projection of LTL in this thesis is done with regards to an extra relaxation which is referred to as the proposition independence assumption. This relaxation is that every instance of a free variable (i.e., a variable not in the projection) in an LTL formula can be independently assigned a value. This is captured formally in definition 6

Definition 6. Under the proposition independence assumption, the projection of an LTL formula ψ onto a set of variables $\mathcal{V}_p \subseteq \mathcal{V}$ is another formula $\psi_{\mathcal{V}_p}$ identical to ψ except every proposition $(v = d)$ with $v \notin \mathcal{V}_p$ is replaced *individually* with $\top$ or $\bot$ in such a way that any path $p = s_1 s_2 \cdots s_l \in S^+$ such that $p \models_{\text{IE}} \psi$ also has $p \models_{\text{IE}} \psi_{\mathcal{V}_p}$.

The relaxation from the proposition independence assumption simplifies the formula greatly, as the assignment is done on every free variable in the formula, including those in temporal subformulae. Statically assigning values to propositions in temporal subformulae is only possible because of the proposition independence assumption. Consider the formula

$$\psi = \mathbf{F}(v_1 = 1) \wedge \mathbf{F}(v_1 = 2) \wedge (v_1 = 3)\mathbf{U}(\neg(v_2 = 1))$$

which states that v_1 must at some point be equal to 1 and 2, but it must first remain 3 until v_2 changes from 1. Naturally, there exists no path which satisfies ψ where v_1 does not change, so if this formula were projected onto $\{v_2\}$, no static assignment could be made to v_1 which would allow ψ to be satisfied. On the other hand, allowing each proposition involving v_1 to become true independently results in the formula

$$\begin{aligned}\psi_{v_2} &= \mathbf{F}\top \wedge \mathbf{F}\top \wedge \top\mathbf{U}(\neg(v_2 = 1)) \\ &= \mathbf{F}(\neg(v_2 = 1))\end{aligned}$$

The resulting formula ψ' does not involve v_1 and requires simply that v_2 eventually change from the value of 1. It can easily be seen that this is satisfied by any path which satisfies ψ.

Performing projection in this way would allows variables not in the projection to be completely eliminated from the formula, and also simplifies the formula to subformulae which contain the projection variables. When the resulting formula is progressed in conjunction with expanding a projected SAS$^+$ problem, there are significantly fewer subformulae to progress to, reducing the augmented state space of the projected problem.

4.2 Proposition Assignment

The projection of

$$\psi = \mathbf{F}(v_1 = 1) \wedge \mathbf{F}(v_1 = 2) \wedge (v_1 = 3)\mathbf{U}(\neg(v_2 = 1))$$

onto $\{v_1\}$ requires that the proposition $(v_2 = 1)$ become *false* in order to correctly simplify the formula. Making the proposition false converts the formula to

$$\begin{aligned}\psi_{v_1} &= \mathbf{F}(v_1 = 1) \wedge \mathbf{F}(v_1 = 2) \wedge (v_1 = 3)\mathbf{U}(\neg\bot) \\ &= \mathbf{F}(v_1 = 1) \wedge \mathbf{F}(v_1 = 2)\end{aligned}$$

which simply specifies that v_1 must at some point be equal to 1 and 2, and is clearly just a relaxation of the original formula. The alternative assignment to $(v_2 = 1)$ results in the subformula $(v_1 = 3)\mathbf{U}(\neg\top)$, which is unsatisfiable. This raises the question of which propositions should be assigned false and which true, and whether this is always possible.

In this section the operator assignFree$(\psi, \mathcal{V}_p)$ is introduced, which assigns true or

false to every proposition $(v = d)$ in ψ where $v \notin \mathcal{V}_p$ such that assignFree$(\psi, \mathcal{V}_p)$ creates a projection of ψ onto $\mathcal{V}_p$. This operator first converts ψ into negation normal form if it is not already, and then replaces every literal which contains a variable $v \notin \mathcal{V}_\psi$ with $\top$. Note here that this means that a proposition inside a negated literal would be set to $\bot$, making the whole literal true. Following this substitution, the formula is simplified according to well known trivial LTL identities (see section 2.2.3). The resulting formula obviously contains no variables which do not appear in $\mathcal{V}_p$, and (provided that the entire formula doesn't simplify to $\top$) preserves some of the semantics of the original formula.

Theorem 1. assignFree$(\psi, \mathcal{V}_p)$ *is a projection of* ψ *onto* $\mathcal{V}_p$*.*

Proof. This property can quite easily be proven inductively on the structure of formulae. Without loss of generality, it can be assumed that ψ is in negation normal form. Otherwise, it can first be transformed into negation normal form and this proof will still be applicable. The base cases are constants and literals, i.e., $\top$, $\bot$, $(v = d)$ and $\neg(v = d)$. It is obvious that the property holds in these cases, as either the formula doesn't change, or changes to $\top$, which is satisfied by any path. The proof for the inductive cases are very straight forward and are evident from the semantics of those operators, so only the proof for the case of two subformulae connected by the until operator, $\psi_1 \mathbf{U} \psi_2$, is included as a schematic proof for the other operators.

Case assignFree$(\psi_1 \mathbf{U} \psi_2, \mathcal{V}_p)$**:**
Assume $p \in S^+$ has $p \models_{\text{IE}} \psi_1 \mathbf{U} \psi_2$. By the semantics of the until operator, there exists a suffix $p[\geq i]$ of p, $p[\geq i] \models_{\text{IE}} \psi_2$ and every suffix $p[\geq j] \models_{\text{IE}} \psi_1$ for $0 \leq j < i$. By the inductive hypothesis $p[\geq i] \models_{\text{IE}}$ assignFree$(\psi_2, \mathcal{V}_p)$ and $p[\geq j] \models_{\text{IE}}$ assignFree$(\psi_1, \mathcal{V}_p)$ for $0 \leq j < i$. It follows immediately by the semantics of the until operator that $p \models_{\text{IE}}$ assignFree$(\psi_1, \mathcal{V}_p) \mathbf{U}$ assignFree$(\psi_2, \mathcal{V}_p)$ From the definition of assignFree, clearly

$$\text{assignFree}(\psi_1 \mathbf{U} \psi_2, \mathcal{V}_p) = \text{assignFree}(\psi_1) \mathbf{U}\, \text{assignFree}(\psi_2),$$

so $p \models_{\text{IE}}$ assignFree$(\psi_1 \mathbf{U} \psi_2, \mathcal{V}_p)$.

In each case (including those not listed here) if $p \in S^+$ has $p \models_{\text{IE}} \psi$ then $p \models_{\text{IE}}$ assignFree$(\psi, \mathcal{V}_p)$, which is a necessary condition for a projection by definition 6. As already observed, all instances of propositions with free variables are eliminated by assignFree$(\psi, \mathcal{V}_p)$. By induction on the structure of a formula, theorem 1 holds. □

For convenience, the projection of each formula in a CNF set Ψ along with on the fly simplification is denoted assignFree$(\Psi, \mathcal{V}_p)$. In accordance with the semantics of a CNF set, if an X-literal is Ψ is projected to $\bot$, it is removed from the associated clause, and an empty clause means the entire Ψ simplifies to $\bot$. Similarly, if an X-literal projects to $\top$, the clause it is in is removed from Ψ, and if Ψ is empty, that represents Ψ simplifying to $\top$.

This section introduced a method to project a formula onto a subset of SAS$^+$ variables, simplifying the formula and allowing LTL constraints to be meaningfully

represented using only those variables. This can be used in conjunction with a projection of the SAS$^+$ problem itself, relaxing both the state space dynamics and LTL constraint.

4.3 Choosing Variables

In order to take advantage of this projection onto a subset $\mathcal{V}_p \subseteq \mathcal{V}$, obviously $\mathcal{V}_p$ must be chosen. In the literature, it is common that a number of projections are chosen, and the problem of choosing the variables for these projections in the context of deterministic unconstrained planning is well studied. Discussion for these projection strategies can be found in [Edelkamp, 2007; Helmert et al., 2007; Haslum et al., 2007] among others. An analysis of strategies for constructing projections given a PLTL constraint is beyond the scope of this thesis, and in experiments for this thesis variables were chosen at random, according to considerations detailed in this section. The algorithm used to choose variables in the experiments for this thesis addresses these considerations and is also presented in this section.

Given that the projection of the SAS$^+$ problem and PLTL constraint $\psi = \mathsf{Pr}\,(\psi) \in z_i$ is intended to estimate the probability of satisfying the given constraint, a natural choice of variables is the variables which appear in propositions in the formula for ψ. The set of these variables are denoted $\mathcal{V}_\psi$. In practice, this set of variables often does not relax the problem sufficiently, and results in the heuristic taking too long to compute for it to be effective. Instead n (not necessarily disjoint) subsets $\mathcal{V}_{\psi,1}, \mathcal{V}_{\psi,2}, \dots \mathcal{V}_{\psi,n}$ are chosen where $\mathcal{V}_{\psi,i} \subseteq \mathcal{V}_\psi$ for all i, and $\bigcup_{0 \leq i \leq n} \mathcal{V}_{\psi,i} = \mathcal{V}_\psi$.

The resulting projections combined account for all the variables so that the role of any one variable in the constraint is captured to at least some extent. The omission of variables not in $\mathcal{V}_\psi$ may relax away details of the SAS$^+$ problem necessary to accurately estimate the probability of satisfying ψ, but identifying which variables not in $\mathcal{V}_\psi$ should be included is once again a problem beyond the scope of this thesis.

The projection detailed above in conjunction with simplification according to trivial LTL identities can lead to projections onto certain variables being trivial (i.e., equal to $\top$). For the projection to be a useful relaxation of the original constraint, it must keep at least some part of the formula. Given a formula ψ, it is possible to determine a set of minimal combinations such that all the variables in some minimal combination must be present in the projection for the projected formula to be non-trivial.

To capture this concept formally, let minvars(ψ) be the set of minimal combinations $\{C_{\psi,1}, C_{\psi,2}, \dots C_{\psi,n}\}$, which are sets of SAS$^+$ variables, such that assignFree($\psi, \mathcal{V}_p$) simplifies to $\top$ if and only if there is no combination $C_{\psi,i} \in$ minvars(ψ) with $C_{\psi,i} \subseteq \mathcal{V}_p$. These combinations are referred to as the necessary combinations of variables for the formula ψ.

It can be assumed without loss of generality that the given formula ψ is in negation normal form and has been simplified according to the trivial LTL identities. To define minvars(ψ), observe that each LTL operator can be associated with a rule which defines the necessary combinations based on the combinations for each direct subformula. minvars(ψ) is formally defined in algorithm 3, and it recursively extracts necessary

combinations under the assumption ψ is in negation normal form.

Algorithm 3 An algorithm for determining minimal combinations of SAS$^+$ variables such that projection is non-trivial if and only if the projection contains at least one minimal combination.

$$
\begin{aligned}
\text{minvars}(\top) &= \text{minvars}(\bot) = \emptyset \\
\text{minvars}((v = d)) &= \text{minvars}(\neg(v = d)) = \{\{v\}\} \\
\text{minvars}(\psi_1 \wedge \psi_2) &= \text{reduce}(\text{minvars}(\psi_1) \cup \text{minvars}(\psi_2)) \\
\text{minvars}(\psi_1 \vee \psi_2) &= \text{reduce}(\{C_{\psi_1,i} \cup C_{\psi_2,j} \mid C_{\psi_1,i} \in \text{minvars}(\psi_1), C_{\psi_2,j} \in \text{minvars}(\psi_2)\}) \\
\text{minvars}(\mathbf{X}\psi) &= \text{minvars}(\psi) \\
\text{minvars}(\psi_1 \mathbf{U} \psi_2) &= \text{minvars}(\psi_2) \\
\text{minvars}(\psi_1 \mathbf{R} \psi_2) &= \text{minvars}(\psi_2)
\end{aligned}
$$

To find the *minimal* combinations, if any combination is a subset of another, the larger set is removed. This is the function of the reduce operator, which is defined as

$$\text{reduce}(\mathcal{S}) = \{C_{\psi,i} \in \mathcal{S} \mid \nexists C_{\psi,j} \in \mathcal{S}, C_{\psi,j} \subset C_{\psi,i}\}.$$

The intuition for the cases of minvars(ψ) can be derived from the trivial LTL identities and a simple analysis of assignFree. It's clear that if ψ has already been simplified, assignFree(ψ) will convert literals to $\top$, and all identities except those for $\neg$ which involve $\top$ will simplify formulae to a subformula (e.g., $\top \mathbf{R} \psi = \psi$) or will simplify the formula to $\top$ (e.g., $\psi \vee \top = \top$). Hence only the identities involving $\top$ need to be considered, specifically only those which simplify the entire formula to $\top$.

For example consider $\psi_1 \wedge \psi_2$. If either subformula evaluates to $\top$, the conjunction will still not become $\top$ unless the other is $\top$ as well. Hence the projection must include sufficient variables that at least one side is non-trivial, so minvars($\psi_1 \wedge \psi_2$) is the union of the minimal projection combinations from either side. On the other hand, in the case of minvars($\psi_1 \vee \psi_2$), if either subformula projects to $\top$, then the disjunction would be immediately simplified to $\top$. In this case, both sides must be non-trivial, so minvars($\psi_1 \vee \psi_2$) specifies that all the variables in at least one necessary combination from each side must be present.

Given a constraint $\psi = \mathsf{Pr}(\psi) \in z_i$, the implementation uses a randomised algorithm to choose a number of subsets of $\mathcal{V}_\psi$ called batches, such that each batch $\mathcal{V}_{\psi,i}$ contains at least one combination in minvars(ψ). This algorithm (shown as algorithm 4) tracks the variables in $\mathcal{V}_\psi$ which have been included in some batch to ensure that every variable in $\mathcal{V}_\psi$ is included in at least one batch.

Initially, starting at line 6, combinations from minvars(ψ) are chosen such that they include a variable which has not already been accounted for. If all the variables are accounted for in this way, the algorithm terminates. It may however be possible for some variables in $\mathcal{V}_\psi$ to not be present in any combination in minvars(ψ). In this case, in the loop at line 10 each such variable is included by choosing a random combination

from minvars(ψ) and adding that combination along with the un-accounted-for variable as a new batch. It is possible that this will create multiple batches where one is a subset of another (e.g., if a batch chosen in the second stage was already chosen in the first stage). This case is dealt with at line 13 where the smaller, pre-existing batch is removed.

Algorithm 4 An algorithm for randomly selecting a number of subsets of $\mathcal{V}_\psi$ such that a projection onto each is non-trivial.

```
1:  procedure ApportionBatches(ψ)
2:      Let ψ be the LTL formula for ψ
3:      p ← ∅                                              ▷ The chosen subsets
4:      r ← V_ψ                                            ▷ The variables not yet in p
5:      while ∃V_{ψ,i} ∈ minvars(ψ) s.t. V_{ψ,i} ∩ r ≠ ∅ do
6:          p_i ← randomly chosen V_{ψ,i} ∈ minvars(ψ) with V_{ψ,i} ∩ r ≠ ∅.
7:          add p_i to p
8:          r ← r \ V_{ψ,i}
9:      end while                                          ▷ r is not necessarily empty
10:     for all v ∈ r do
11:         p_i ← randomly chosen V_{ψ,i} ∈ minvars(ψ)
12:         if p_i ∈ p then
13:             remove p_i from p
14:         end if
15:         add v to p_i
16:         add p_i to p
17:     end for
18: end procedure
```

Output: p is a set of batches. Every $v \in \mathcal{V}_\psi$ is in some $\mathcal{V}_{\psi,i} \in p$.

Using algorithm 4, a number of subsets of $\mathcal{V}$ are chosen, and a projection of both the SAS$^+$ problem and the PLTL constraint onto each of these subsets can be used as a simplified planning problem for the heuristic detailed in later sections.

4.4 Summary

Projection of LTL formula, as a counterpart for projection of SAS$^+$ problems, allows variables to be safely eliminated from formulae in such a way that any run which satisfies an LTL formula will also satisfy it's projection. The primary value in this projection is the simplification of the formula in a structured way, but it also allows LTL constraints to be applied to projections of SAS$^+$ problems. This property is invaluable for use in a planning heuristic that uses SAS$^+$ projections.

The projection of LTL formulas in this thesis relies on the proposition independence assumption, which allows individual propositions in the formula to be individually assigned true or false statically. While this causes some loss of information, it avoids the issue of finding satisfying assignments to free variables, and greatly reduces the number of subformula that can be produced by progression. Projection under the

literal independence assumption can be performed by simply replacing every literal in negation normal form with true, and this operation is defined as assignFree$(\psi, \mathcal{V}_p)$.

When performing this operation, it is possible for formulae to be reduced to the trivially satisfiable formula $\top$ depending on the selection of projection variables. An analysis of the formula can determine which variables are necessary in combination to prevent this, an algorithm is presented which chooses a number of projections at random to be used in conjunction. These projections together contain all the variables in the original formula, so the semantics of each part of the formula will be preserved after projection to some extent.

In the next chapter, a method for overestimating the probability of a LTL constraint being satisfied will be presented, which estimates the probability of subformulae of the constraint individually.

Chapter 5

Probability Estimation by Decomposition

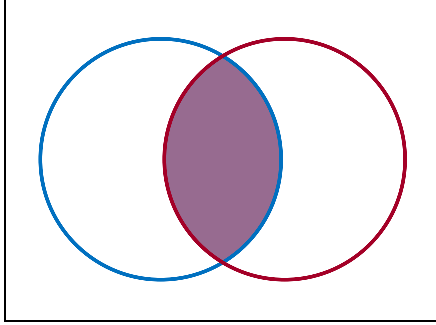
Solving SSPs with MO-PLTL constraints efficiently requires that the restrictions on the plan space imposed by the constraints can be efficiently predicted. This chapter details a method for estimating the probability of reaching the goal of an SSP while satisfying a single PLTL constraint ψ. This method revolves around decomposition of LTL formula, which means in this context breaking formula into sub-formulae which are treated individually. The probability of satisfying a formula can be estimated based on the probability of satisfying the sub-formulae.

Using this concept, a planning model called Concurrent Constrained SSP (CC-SSP) is introduced, representing a decomposition of the formula in every augmented state. The CC-SSP clones agents, resulting in agents being in multiple states simultaneously, representing independent satisfaction of several sub-formulae simultaneously in the same problem. From the solution to a CC-SSP an estimate for the probability of satisfying a given constraint can be derived. CC-SSPs can be formulated as a flow problem, though the problem definition allows duplication of flow. The duplication of flow (equivalently the nature of CC-SSPs that agents are cloned into multiple states at once) causes the estimated probability to be very poor in some common cases, especially where there are loops in the underlying SSP.

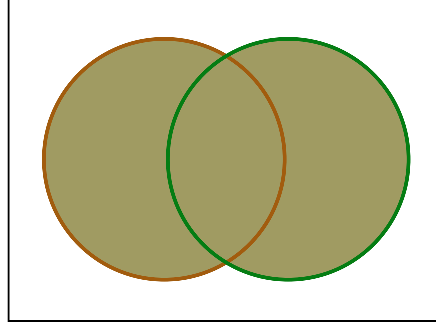
This chapter first defines and explains LTL decomposition in section 5.1, and uses this definition to define Concurrent Constrained SSPs in section 5.2. A linear program representing the flow problem for a CC-SSP is given in section 5.3 and finally examples where this estimation method performs very poorly are presented and explained in section 5.4.

5.1 LTL Decomposition

Augmented states in this thesis are defined as pairs $\langle s, \Psi \rangle$ of an SSP state and a CNF set. This CNF set is a set of sets of X-literals, representing a conjunction of disjunctions, reminiscent of conjunctive normal form in classical logic, except that an X-literal is a temporal formula ϕ beginning with the next operator $\mathbf{X}$. The task to be achieved in this chapter is to estimate the probability of satisfying the formula represented by Ψ



(a) The probability of a conjunction is less than that of any conjunct.



(b) The probability of a disjunction is less than the sum of the probability of the disjuncts.

Figure 5.1: A diagrammatic representation of the probability inequalities used in formula decomposition. While these venn diagrams only have two parts, an extension to more conjuncts or disjuncts is trivial.

from state s under any optimal policy π^* for the MO-PLTL SSP problem, denoted as $\mathsf{Pr}^{\pi^*}(\Psi \mid s)$. If the probability is from the initial state, the state is omitted in the notation, so $\mathsf{Pr}^{\pi^*}(\Psi)$ is the probability of satisfying Ψ and reaching a goal from the initial state s_0.

Given this task, it is natural to relax the problem by calculating the probability of each clause Φ individually, but constraints could contain only a single clause, meaning that no relaxation would take place. Instead a further relaxation is done, and the probability of satisfying each X-literal is estimated individually. To estimate the probability $\mathsf{Pr}^{\pi^*}(\Psi)$ for the whole formula from the probabilities of the X-literals in Ψ, the inequalities are used:

$$\mathsf{Pr}^{\pi^*}(\Psi) = \mathsf{Pr}^{\pi^*}\left(\bigwedge_{\Phi\in\Psi} \Phi\right) \le \min_{\Phi\in\Psi} \mathsf{Pr}^{\pi^*}(\Phi) \le \frac{\sum_{\Phi\in\Psi} \mathsf{Pr}^{\pi^*}(\Phi)}{|\Psi|} \tag{5.1}$$

$$\mathsf{Pr}^{\pi^*}(\Phi) = \mathsf{Pr}^{\pi^*}\left(\bigvee_{\phi\in\Phi} \phi\right) \le \sum_{\phi\in\Phi} \mathsf{Pr}^{\pi^*}(\phi)\,. \tag{5.2}$$

That is, the probability of satisfying a conjunction Ψ is less than or equal to the average probability of satisfying the conjuncts Φ, and the probability of satisfying a disjunction Φ is less than or equal to the sum of the probabilities of satisfying the disjuncts ϕ. Figure 5.1 provides a diagrammatic representation of these inequalities, though the reader may notice that the bound on conjunctions is not as tight as it could be. The probability of satisfying a conjunction is actually less than or equal to the probability of satisfying any conjunct Φ, and hence is upper-bounded by the probability of satisfying the conjunct with the *minimum* probability. However, formulating the bound in this way would lead to a bi-level optimisation problem, which is too computationally expensive to be used in the context of heuristic search. The alternative taken in this

thesis is to use the average, which can be computed as a linear constraint.

Using decomposition, the probability of satisfying a formula ψ from a state s can be estimated as

$$\mathsf{Pr}^{\pi^*}(\psi \mid s) \leq \frac{\sum_{\Phi\in\Psi}\left(\sum_{\phi\in\Phi}\mathsf{Pr}^{\pi^*}(\phi \mid s)\right)}{|\Psi|} \tag{5.3}$$

where $\Psi = \mathrm{CNF}(\psi, s)$. Importantly, because of the inequalities in equations 5.1 and 5.2, this estimate is an upper bound on the probability. The relevance of this upper bound lies in the definition of an admissible heuristic for a PLTL constraint. A heuristic h^{ψ_i} for a PLTL constraint $\psi_i = \mathsf{Pr}(\psi_i) \in z_i$ is defined as a function which estimates the probability of reaching an accepting goal from a product state $s^\times$, and such a heuristic is admissible if, for all product states $s^\times \in \mathrm{S}^\times$, $h^{\psi_i}(s^\times) \geq \mathsf{Pr}^{\pi^*}\left(s_i^\times\right)$ for every optimal policy π^* in the associated product C-SSP. It is sufficient to consider only the mode for constraint ψ_i, which in this thesis will be a CNF set Ψ. Hence, an admissible heuristic in this thesis is a function $h^{\psi_i} : \mathrm{S} \times 2^{2^{\Sigma(\psi_i)}} \to [0, 1]$ which, given an augmented state $\langle s, \Psi\rangle$, upper-bounds the probability of satisfying Ψ from the state s for every optimal policy π^*, i.e., $h^{\psi_i}(\langle s, \Psi\rangle) \geq \mathsf{Pr}^{\pi^*}(\Psi \mid s)$. This is the basis for the heuristics which will be detailed in this chapter and chapter 6, but taken as it stands, it does not relax the problem sufficiently to be used as an effective heuristic.

Consider the formula

$$\psi = (a\mathbf{U}b)\mathbf{U}\mathbf{G}c.$$

where a, b and c are propositions. ψ states that $a\mathbf{U}b$ must hold until c becomes true forever. The conjunctive normal form for this formula in a state where a and c hold would be

$$\mathrm{CNF}(\psi, \{a, c\}) = \{\{\mathbf{X}a\mathbf{U}b\}, \{\mathbf{X}((a\mathbf{U}b)\mathbf{U}\mathbf{G}c), \mathbf{X}\mathbf{G}c\}\}.$$

Note that this contains the X-literal $\mathbf{X}\psi$, so decomposing this formula into the X-literals would include a formula which is just as hard to satisfy as the original. This is resolved by allowing formulae to be decomposed after each progression, relaxing the problem even further. Recurrently decomposing the formula at each state maintains the upper bound property necessary for PLTL heuristics, but makes the estimation process less straightforward.

In the next section a new type of planning problem called a CC-SSP is introduced to perform this estimation. Instead, the probability of satisfying LTL as estimated by decomposing formulae at every step can be represented by a much simpler recursive equation, but the dual representation of CC-SSPs presented in section 5.3 lends itself to integration with PLTL-dual in chapter 6.

5.2 Concurrent Constrained SSPs

To represent the decomposition of formulae at every state, a new type of planning problem called a Concurrent Constrained SSP (CC-SSP) will be introduced, the optimal solution of which will be directly related to an estimate of the probability of satisfying a given formula ψ from an SSP state s. The states of this planning problem are pairs

$\langle s, \phi \rangle$ where s is an SSP state, and ϕ is an X-literal. These pairs are called X-literal states. In a CC-SSP, instead of an agent being in a single state at each step, the agent is in a set of states $\mathcal{P}$. CC-SSPs are used in this thesis to model decomposition of formula at every state, and hence to estimate the probability of satisfying a formula.

Before formalising the concept of a CC-SSP in section 5.2.2, the semantics of redistribution (or 'cloning') of agents is formalised, as this helps motivate the definition of CC-SSPs, and introduces several definitions useful for defining them.

5.2.1 State Redistribution

When an agent takes an action from an X-literal state $\langle s, \phi \rangle$, the X-literal will be progressed and converted to CNF once again. This would put the agent in an augmented state under the semantics of a MO-PLTL SSP problem, but under the semantics of a CC-SSP the agent is redistributed to one or more X-literal states chosen from those generated by decomposing the CNF found when progressing ϕ.

Given a clause Φ, let decompose(s, Φ) be an operator which outputs the set of X-literal states $\langle s, \phi \rangle$ with the same SSP states, and where ϕ is in Φ. That is,

$$\text{decompose}(s, \Phi) \equiv \{\langle s, \phi \rangle \mid \forall \phi \in \Phi\}.$$

Similarly, applied to a CNF set Ψ, decompose(s, Ψ) outputs a set of all X-literal states $\langle s, \phi \rangle$ where ϕ is in some clause in Ψ. That is

$$\text{decompose}(s, \Psi) \equiv \bigcup_{\Phi \in \Psi} \text{decompose}(s, \Phi).$$

In a CC-SSP $\mathcal{C}$, every time agent would enter an augmented state $\langle s, \Psi \rangle$, a coin is flipped by the environment, choosing a clause Φ from Ψ following a uniform distribution (i.e., $\mathsf{Pr}_{\mathcal{C}}(\Phi \mid \Psi) = 1/|\Psi|$), and the agent enters *all* the states in decompose(s, Φ). A given X-literal ϕ may appear in multiple clauses $\Phi \in \Psi$, so the probability that, after redistribution, there is an agent in any given X-literal state $\langle s, \phi \rangle$ is not uniform over the possible X-literal states. Assuming an agent is redistributed from $\langle s, \Psi \rangle$, the probability that after redistribution there is a clone of that agent in a given X-literal state $\langle s, \phi \rangle$ is denoted $\mathsf{Pr}_{\mathcal{C}}(\langle s, \phi \rangle \mid \langle s, \Psi \rangle)$ and can be calculated as

$$\begin{aligned}
\mathsf{Pr}_{\mathcal{C}}(\langle s, \phi \rangle \mid \langle s, \Psi \rangle) &\equiv \sum_{\Phi \in \Psi : \phi \in \Phi} \mathsf{Pr}_{\mathcal{C}}(\Phi \mid \Psi) \\
&= \sum_{\Phi \in \Psi : \phi \in \Phi} \frac{1}{|\Psi|} \\
&= \frac{|\{\Phi \in \Psi \mid \phi \in \Phi\}|}{|\Psi|}
\end{aligned}$$

As this probability is not dependent on s, $\mathsf{Pr}(\phi \mid \Psi)$ will be used interchangeably with $\mathsf{Pr}(\langle s, \phi \rangle \mid \langle s, \Psi \rangle)$.

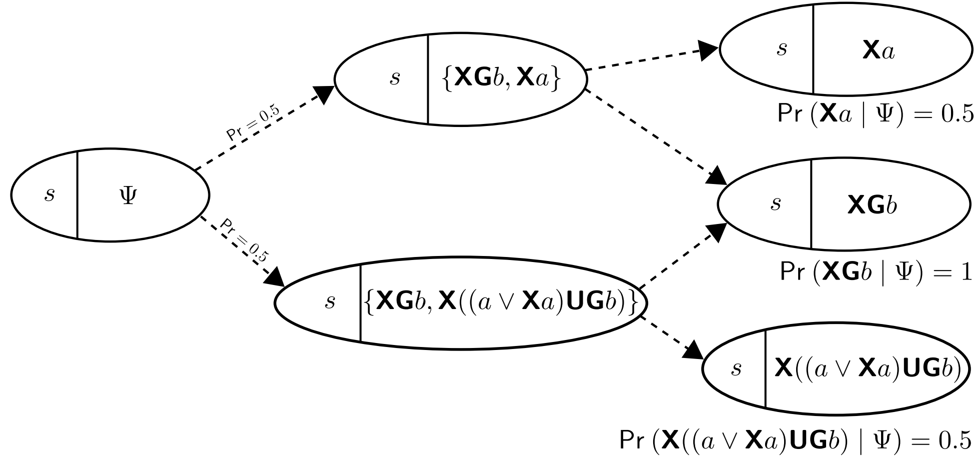


Figure 5.2: Redistribution of agents. Note how the probability of each X-literal state is dependent on the number of clauses it is in.

An example redistribution can be seen in figure 5.2 for the CNF set

$$\Psi = \{\{\mathbf{XG}b, \mathbf{X}a\}, \{\mathbf{XG}b, \mathbf{X}((a \vee \mathbf{X}a)\mathbf{UG}b)\}\}.$$

where a and b are arbitrary propositions. An agent that would enter $\langle s, \Psi \rangle$ is redirected to either $\langle s, \{\mathbf{XG}b, \mathbf{X}a\} \rangle$ or $\langle s, \{\mathbf{XG}b, \mathbf{X}((a \vee \mathbf{X}a)\mathbf{UG}b)\} \rangle$ with probability $1/|\Psi| = 0.5$. In each case, the agent enters both X-literals in the chosen clause, meaning that there are, in either case, twice as many agents being simulated concurrently. This can be thought of as an agent for each item in the disjunction, concurrently (and independently) attempting to satisfy that disjunct. It is clear also from figure 5.2 how the probability of there being an agent in each X-literal state is dependent on the number of clauses that X-literal appears in. Because $\mathbf{XG}b$ appears in both clauses, $\mathsf{Pr}\left(\mathbf{XG}b \mid \Psi\right)$ is 1, where the probability of the other clauses is 0.5.

To motivate this definition, it can be related back to equation 5.3. Consider the expected number of agents reaching a goal in a CC-SSP $\mathcal{C}$ from those distributed from the augmented state $\langle s, \Psi \rangle$ under a policy π. The expected number of agents reaching a goal from an augmented state $\langle s, \Psi \rangle$ under a policy π is (using notation which will be formalised later) denoted $\text{Egoals}(\langle s, \Psi \rangle, \pi)$. This expectation can be expressed in terms of the expected number of agents reaching a goal from each state in $\text{decompose}(s, \Psi)$ under π, denoted $\text{Egoals}(\langle s, \phi \rangle, \pi)$, and is expressed as follows

$$\begin{aligned}
\text{Egoals}(\langle s, \Psi \rangle, \pi) &= \sum_{\langle s, \phi \rangle \in \text{decompose}(s, \Psi)} \mathsf{Pr}\left(\phi \mid \Psi\right) \cdot \text{Egoals}(\langle s, \phi \rangle, \pi) \\
&= \sum_{\langle s, \phi \rangle \in \text{decompose}(s, \Psi)} \frac{|\{\Phi \in \Psi \mid \phi \in \Phi\}|}{|\Psi|} \cdot \text{Egoals}(\langle s, \phi \rangle, \pi) \\
&= \frac{\sum_{\Phi \in \Psi} \left(\sum_{\phi \in \Phi} \text{Egoals}(\langle s, \phi \rangle, \pi) \right)}{|\Psi|} \qquad (5.4)
\end{aligned}$$

To take the step ending at equation 5.4, observe that each X-literal ϕ is counted once for each clause Φ it is in, and this is equivalent to counting each X-literal in each clause. Note the strong similarity between equation 5.3 and equation 5.4 achieved by substituting $\mathsf{Pr}^{\pi^*}(\cdot \mid s)$ for Egoals$(\langle s, \cdot \rangle, \pi)$. Of course, as of yet CC-SSPs have not been formally defined, and hence Egoals$(\langle s, \cdot \rangle, \pi)$ has no meaning, so this is a superficial relationship, but given an appropriate definition, the optimal solution to a CC-SSP finds an estimate for the probability of satisfying some CNF set Ψ from a state s in an SSP.

5.2.2 CC-SSP Definition

In this section, using the definitions from section 5.2.1, CC-SSPs are formally defined.

Definition 7. A Concurrent Constrained SSP is a tuple $\mathcal{C} \equiv \langle \hat{\mathrm{S}}, \langle s_0, \Psi_0 \rangle, \mathrm{G}, \mathrm{A}, \mathrm{T} \rangle$, where $\hat{\mathrm{S}}$ is a set of augmented states, $\langle s_0, \Psi_0 \rangle \in \hat{\mathrm{S}}$ is an augmented state representing the initial states, G is a set of absorbing goal states. Note that this definition does not specify the X-literal states which were described in section 5.2.1. The set of all X-literal states in a CC-SSP is denoted $\tilde{\mathrm{S}}$ and is derived from $\hat{\mathrm{S}}$ below. A is set of actions defined for the CC-SSP and $\mathrm{A}(s)$ is a set of actions available in any X-literal state $\langle s, \phi \rangle \in \tilde{\mathrm{S}}$, and the transition function $\mathrm{T}(s' \mid s, \alpha)$ is the probability that agent taking an action $\alpha \in \mathrm{A}(s)$ when in any X-literal state $\langle s, \phi \rangle \in \tilde{\mathrm{S}}$ will transition to s', and be redistributed to the X-literal states in decompose$(s', \mathrm{CNF}(\text{un-}\mathbf{X}(\phi), s'))$. A is extended with a special action α_{die} which represents an agent's path being abandoned. $\mathrm{T}(s' \mid s, \alpha_{die})$ is not defined for any states s, s'.

Note that any CC-SSP $\mathcal{C}$ has a close relationship with an underlying SSP $\mathcal{S}_{\mathcal{C}}$ which can be derived from the tuple $\mathcal{C} = \langle \hat{\mathrm{S}}, \langle s_0, \Psi_0 \rangle, \mathrm{G}, \mathrm{A}, \mathrm{T} \rangle$ as

$$\mathcal{S}_{\mathcal{C}} = \langle \mathrm{S}, s_0, \mathrm{G}, \mathrm{A} \setminus \{\alpha_{die}\}, \mathrm{T}, C \rangle$$

where $\mathrm{S} = \{s \mid \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}}\}$ and C is a function which always returns 1, as there is no cost function in a CC-SSP. A CC-SSP $\mathcal{C}$ must satisfy the condition that the underlying SSP $\mathcal{S}_{\mathcal{C}}$ has a proper policy, i.e., that there is a policy which, from any state, eventually reaches a state in G with probability 1. CC-SSPs are used in this thesis to estimate the probability of satisfying formula while reaching a goal in the underlying SSP $\mathcal{S}_{\mathcal{C}}$.

Several sets necessary to define the semantics of a CC-SSP $\mathcal{C} = \langle \hat{\mathrm{S}}, \langle s_0, \Psi_0 \rangle, \mathrm{G}, \mathrm{A}, \mathrm{T} \rangle$ are derived from the tuple. The set of X-literal states over which the search is defined is denoted $\tilde{\mathrm{S}}$ and is the combination of the X-literal states from decomposing all the augmented states in $\hat{\mathrm{S}}$.

$$\tilde{\mathrm{S}} \equiv \bigcup_{\langle s, \Psi \rangle \in \hat{\mathrm{S}}} \text{decompose}(s, \Psi)$$

To define the objective of a CC-SSP, two sets of states are derived, $\hat{\mathrm{F}} \subset \hat{\mathrm{S}}$ is the set of absorbing augmented states, and $\hat{\mathrm{G}} \subseteq \hat{\mathrm{F}}$ is a set of augmented states which are goals. $\hat{\mathrm{F}}$ is defined as the set of augmented states $\langle s, \Psi \rangle \in \hat{\mathrm{S}}$ where the probability of satisfying Ψ from while reaching a goal from s in $\mathcal{S}_{\mathcal{C}}$ is trivially 1 or 0. Given an augmented state $\langle s, \Psi \rangle$ there are four cases in which this happens:

1. If $\Psi = \emptyset$, representing $\top$, then Ψ is trivially satisfiable. As it is assumed that there is a proper policy from any state in the underlying SSP $\mathcal{S_C}$, the probability of reaching a state $s' \in \mathrm{G}$ in $\mathcal{S_C}$ is 1, and any path to that goal state would satisfy the LTL formula $\top$. Thus there is no point considering states beyond this augmented state, and we treat it like a goal state.

2. If $s \in \mathrm{G}$ and idle(s, Ψ) returns $\top$, then a goal has been reached which satisfies Ψ, meaning the probability is 1.

3. If $\Psi = \{\emptyset\}$, representing $\bot$, then Ψ is trivially unsatisfiable. Similarly to case 1, even if s is not a goal, there can be no path to a goal in $\mathcal{S_C}$ which satisfies the LTL formula $\bot$, and so the probability of reaching an accepting goal state from this state is 0.

4. If $s \in \mathrm{G}$ but idle(s, Ψ) returns $\bot$, then Ψ is not satisfied at this goal, so the probability is 0.

$\hat{\mathrm{G}}$ is the set of augmented states $\langle s, \Psi \rangle$ in $\hat{\mathrm{F}}$ which satisfy condition 1 or 2. These are the cases where the probability that Ψ is satisfied when reaching a goal from s in $\mathcal{S_C}$ is trivially 1. Formally

$$\hat{\mathrm{F}} \equiv \{\langle s, \Psi \rangle \mid \langle s, \Psi \rangle \in \hat{\mathrm{S}}, (\Psi = \top) \vee (\Psi = \bot) \vee (s \in \mathrm{G})\}$$
$$\hat{\mathrm{G}} \equiv \{\langle s, \Psi \rangle \mid \langle s, \Psi \rangle \in \hat{\mathrm{S}}, (\Psi = \top) \vee (\mathrm{idle}(s, \Psi) = \top \wedge s \in \mathrm{G})\}$$

Including states which are not in G as "goal" states is not the norm, but it is done in the context of CC-SSPs for two reasons. The first reason is that decompose(s, Ψ) is the empty set when $\Psi = \{\emptyset\}$ and when $\Psi = \emptyset$, meaning that the semantics for actions would have to be extended to include states of the form $\langle s, \top \rangle$ and $\langle s, \bot \rangle$. The second is that pruning the state space beyond augmented states $\langle s, \Psi \rangle$ where $\Psi = \top$ or $\Psi = \bot$ can sometimes greatly reduce the size of the CC-SSP, meaning that solving it is much easier.

5.2.3 Formal Concepts for CC-SSPs

A **path** through a CC-SSP is a sequence of X-literal states $\langle s_1, \phi_1 \rangle \langle s_2, \phi_2 \rangle \ldots$ with $\langle s_i, \phi_i \rangle \in \tilde{\mathrm{S}}$ that is either infinite, or is terminated by an augmented state $\langle s_n, \Psi_n \rangle \in \hat{\mathrm{F}}$. A path terminated in this way is called a complete path, and if $\langle s_n, \Psi_n \rangle \in \hat{\mathrm{G}}$, the path is also a goal path. Let $\tilde{\mathrm{S}}^{+}$ be the set of all incomplete paths through a CC-SSP, $\tilde{\mathrm{S}}^{+\hat{\mathrm{F}}}$ be the set of all complete paths through a CC-SSP and $\tilde{\mathrm{S}}^{+\hat{\mathrm{G}}} \subseteq \tilde{\mathrm{S}}^{+\hat{\mathrm{F}}}$ be the set of goal paths.

Intuitively, a **run** R of a CC-SSP is a sequence $\mathcal{P}_1 \overset{\mathcal{A}_1}{\to} \mathcal{P}_2 \overset{\mathcal{A}_2}{\to} \ldots$ of sets of paths of increasing length, where each path represents the history of an agent up to that step, and each set is annotated with a function representing which actions were taken from those histories. Each set of paths $\mathcal{P}_i$ has $\mathcal{P}_i \subset \tilde{\mathrm{S}}^{+} \cup \tilde{\mathrm{S}}^{+\hat{\mathrm{F}}}$, each path in $\mathcal{P}_i$ has length i, and by a slight abuse of notation, $\mathcal{P}_i \in R$ denotes that $\mathcal{P}_i$ is one of the sets of paths in R. Each $\mathcal{A}_i$ is a function $\mathcal{A}_i : \mathcal{P}_i \cap \tilde{\mathrm{S}}^{+} \to \mathrm{A}$ which annotates, for all incomplete paths in

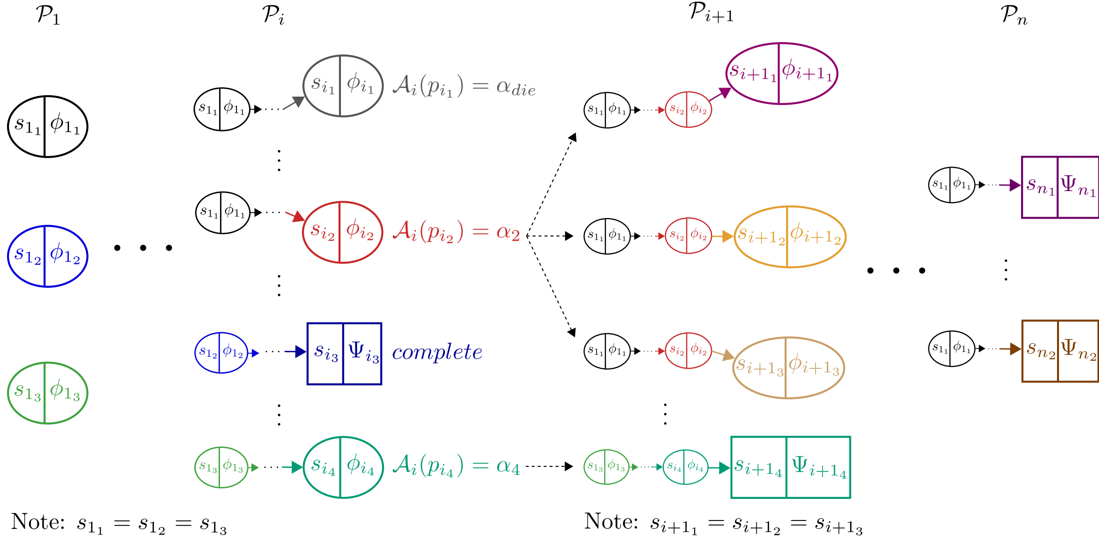


Figure 5.3: The paths present in a run are extended and cloned in each $\mathcal{P}_i$ until they are completed or abandoned.

$\mathcal{P}_i$ which action was taken. Some paths will be terminated by reaching an augmented state in $\hat{\mathrm{F}}$ or will be abandoned using the special action α_{die}. For convenience, the set of paths that are extended in the next set is denoted

$$\text{alive}(\mathcal{P}_i, \mathcal{A}_i) = \{p \mid p \in \mathcal{P}_i \cap \tilde{\mathrm{S}}^+, \mathcal{A}_i(p) \neq \alpha_{die}\}$$

A run can be finite or infinite, and in the case that it is finite, the final set $\mathcal{P}_n$ and annotation $\mathcal{A}_n$ have $\text{alive}(\mathcal{P}_n, \mathcal{A}_n) = \emptyset$. The length of a run is $\ell(R)$, which is defined as the maximum i such that $\mathcal{P}_i \in R$, and can be infinite.

Each set $\mathcal{P}_i$ of a run is the set of paths representing the first i steps of all agents which complete at least i steps. At each successive set $\mathcal{P}_{i+1}$, the incomplete paths from $\mathcal{P}_i$ are extended by one state. In the case that an agent would be redistributed to multiple X-literal states, the associated path p in $\mathcal{P}_i$ is cloned for each new state, so that $\mathcal{P}_{i+1}$ contains multiple paths which are p followed by one new state from the redistribution. In figure 5.3, this is shown schematically, with each path in $\mathcal{P}_i$ being extended, abandoned or completed at each step. In figure 5.3 note that the run is finite, with the final set $\mathcal{P}_n$ containing only complete paths, meaning the domain of $\mathcal{A}_n$ is empty. The progression to a successive set of paths $\mathcal{P}_{i+1}$ from $\mathcal{P}_i$ is formalised as follows.

For all i such that $\text{alive}(\mathcal{P}_i, \mathcal{A}_i) \neq \emptyset$, $\mathcal{P}_{i+1}$ exists. For each incomplete path $p \in \text{alive}(\mathcal{P}_i, \mathcal{A}_i)$, let $\mathcal{P}_{i+1,p} \subseteq \mathcal{P}_{i+1}$ be the set of paths in $\mathcal{P}_{i+1}$ prefixed by p. $\mathcal{P}_{i+1}$ contains only the paths in these sets, i.e.,

$$\mathcal{P}_{i+1} = \bigcup_{p \in \text{alive}(\mathcal{P}_i, \mathcal{A}_i)} \mathcal{P}_{i+1,p} \,.$$

Every path $p' \in \mathcal{P}_{i+1,p}$ is equal to p extended by one state. For every path to action mapping $p = \langle s_1, \phi_1 \rangle \dots \langle s_i, \phi_i \rangle \mapsto \alpha$ in $\mathcal{A}_i$, there exists a state s_{i+1} with $\mathrm{T}\left(s_{i+1} \mid s_i, \alpha\right) > 0$, such that either

- transitioning to s_{i+1} reaches an absorbing augmented state, i.e.,

  $$\langle s_{i+1}, \mathrm{CNF}(\text{un-}\mathbf{X}(\phi_i), s_{i+1}) \rangle \in \hat{\mathrm{F}}$$

  in which case $\mathcal{P}_{i+1,p} = \{p \langle s_{i+1}, \mathrm{CNF}(\text{un-}\mathbf{X}(\phi_i), s_{i+1}) \rangle\}$; or

- there exists a clause $\Phi \in \mathrm{CNF}(\text{un-}\mathbf{X}(\phi_i), s_{i+1})$ such that

  $$\mathcal{P}_{i+1,p} = \{p \langle s_{i+1}, \phi_{i+1} \rangle \mid \phi_{i+1} \in \Phi\},$$

  which effectively clones p for each X-literal in the clause Φ.

The second case encapsulates the redistribution of agents, whereas the first case captures the completion of paths. Abandoned paths, those paths for which $\mathcal{A}_i(p) = \alpha_{die}$, are not included or extended in $\mathcal{P}_{i+1}$.

A run starts at a specified state. That is, if a run starts at an X-literal state $\langle s_1, \phi_1 \rangle$, then $\mathcal{P}_1$ is the singleton set $\{\langle s_1, \phi_1 \rangle\}$. Otherwise if the run starts an augmented state $\langle s_1, \Psi_1 \rangle$ then $\mathcal{P}_1$ is decompose(s_1, Φ) for some $\Phi \in \Psi_1$ by redistribution.[1]

A **policy** for a CC-SSP is defined as a function $\pi : \tilde{\mathrm{S}}^+ \times \mathrm{A} \to [0,1]$ which specifies a probability $\pi(p, \alpha)$ of each action being taken from the end of a given path. Note here that all agents follow the same policy independently, and so the policy is dependent on a path p, rather than being dependent on a set of paths $\mathcal{P}_i$. Stationary policies, that is policies where $\pi(p, \alpha)$ is dependent only on the final state in p, are sufficient to optimize CC-SSPs. As such, only stationary policies will be considered in this thesis, and $\pi(\langle s, \phi \rangle, \alpha)$ is used to denote the probability $\pi(p, \alpha)$ where p ends with $\langle s, \phi \rangle$.

The probability of a run R in a CC-SSP $\mathcal{C}$ given a policy π is denoted $\mathsf{Pr}_{\mathcal{C}}\left(R \mid \pi\right)$, and defined

$$\mathsf{Pr}_{\mathcal{C}}\left(R \mid \pi\right) \equiv \mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_1\right) \prod_{i=1}^{\ell(R)} \mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1} \mid \mathcal{P}_i, \mathcal{A}_i\right) \cdot \mathsf{Pr}_{\mathcal{C}}\left(\mathcal{A}_i \mid \mathcal{P}_i, \pi\right).$$

The probability $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_1\right)$ is based on the initial state of the run, if it starts from an X-literal state $\langle s, \phi \rangle$, $\mathcal{P}_1$ contains only $\langle s, \phi \rangle$ and $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_1\right) = 1$, whereas if the run starts from an augmented state $\langle s, \Psi \rangle$ then $\mathcal{P}_1$ is equal to decompose(s, Φ) for some $\Phi \in \Psi$, and $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_1\right) = \mathsf{Pr}\left(\Phi \mid \Psi\right)$. The probability $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{A}_i \mid \mathcal{P}_i, \pi\right)$ of a given action mapping being chosen under the policy π is

$$\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{A}_i \mid \mathcal{P}_i, \pi\right) = \prod_{p \in \mathcal{P}_i} \pi(\mathrm{last}(p), \mathcal{A}_i(p))$$

Finally, the probability of each transition $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1} \mid \mathcal{P}_i, \mathcal{A}_i\right)$ is found by reconstructing

[1] A subscript 1 is used for the first state in a run to distinguish from the initial state of a CC-SSP, $\langle s_0, \Psi_0 \rangle$. In the case that a run starts from the initial state, we have that $s_0 = s_1$ and $\Psi_0 = \Psi_1$.

which state and clause resulted in the extensions in $\mathcal{P}_{i+1}$. For each incomplete path $p = \langle s_1, \phi_1 \rangle \dots \langle s_i, \phi_i \rangle \in \text{alive}(\mathcal{P}_i, \mathcal{A}_i)$, consider the probability that $\mathcal{A}_i(p)$ resulted in the set $\mathcal{P}_{i+1,p}$ of paths in $\mathcal{P}_{i+1}$ prefixed by p. This probability is denoted $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1,p} \mid \mathcal{A}_i(p)\right)$, and there are two cases for $\mathcal{P}_{i+1,p}$:

- $\mathcal{P}_{i+1,p}$ is a singleton set containing a completed path $p\,\langle s_{i+1}, \Psi_{i+1} \rangle$, in which case

  $$\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1,p} \mid \mathcal{A}_i(p)\right) = \mathrm{T}\left(s_{i+1} \mid s_i, \mathcal{A}_i(p)\right)$$

  as the probability is simply dependent on the transition to the new state.

- $\mathcal{P}_{i+1,p}$ is a set containing incomplete paths $p\,\langle s_{i+1}, \phi_{i+1} \rangle$ for all $\phi_{i+1} \in \Phi$ for some $\Phi \in \text{CNF}(\text{un-}\mathbf{X}(\phi_i), s_{i+1})$. In which case

  $$\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1,p} \mid \mathcal{A}_i(p)\right) = \mathrm{T}\left(s_{i+1} \mid s_i, \mathcal{A}_i(p)\right) \cdot \mathsf{Pr}_{\mathcal{C}}\left(\Phi \mid \text{CNF}(\text{un-}\mathbf{X}(\phi_i), s_{i+1})\right).$$

  That is, the probability is that of both the transition to s_{i+1} and the choice of Φ, which are independent events.

Given this definition, the probability of a transition from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ given $\mathcal{A}_i$ is

$$\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1} \mid \mathcal{P}_i, \mathcal{A}_i\right) = \prod_{p \in \text{alive}(\mathcal{P}_i, \mathcal{A}_i)} \mathsf{Pr}_{\mathcal{C}}\left(\mathcal{P}_{i+1,p} \mid \mathcal{A}_i(p)\right),$$

being the product of the probabilities of each independent path extension forming $\mathcal{P}_{i+1}$.

A run is *possible* under a policy π if $\mathsf{Pr}_{\mathcal{C}}\left(\mathcal{A}_i \mid \pi\right) > 0$ for all $\mathcal{A}_i$, that is, if every action mapping could possibly have been chosen while following the policy π. The set of all runs starting at an X-literal state $\langle s, \phi \rangle$ possible under a policy π is denoted $\text{runs}_{\mathcal{C}}(\langle s, \phi \rangle, \pi)$, and similarly the set of all runs starting at an augmented state $\langle s, \Psi \rangle$ possible under a policy π is denoted $\text{runs}_{\mathcal{C}}(\langle s, \Psi \rangle, \pi)$.

The number of agents reaching a goal in a given run R is the number of paths in R in $\tilde{\mathrm{S}}^{+\hat{\mathrm{G}}}$, that is, the number that are terminated by reaching a state in $\hat{\mathrm{G}}$. This amount is denoted

$$\#\text{goals}(R) \equiv \sum_{\mathcal{P}_i \in R} |\tilde{\mathrm{S}}^{+\hat{\mathrm{G}}} \cap \mathcal{P}_i|$$

The expected number of agents which reach a goal from a given X-literal state $\langle s, \phi \rangle$ while following policy π is

$$\text{Egoals}(\langle s, \phi \rangle, \pi) \equiv \mathbb{E}_{R \in \text{runs}_{\mathcal{C}}(\langle s, \phi \rangle, \pi)}\left[\#\text{goals}(R)\right]$$

and similarly for the number of agents reaching a goal state from a given augmented state $\langle s, \Psi \rangle$,

$$\text{Egoals}(\langle s, \Psi \rangle, \pi) \equiv \mathbb{E}_{R \in \text{runs}_{\mathcal{C}}(\langle s, \Psi \rangle, \pi)}\left[\#\text{goals}(R)\right].$$

The objective of a CC-SSP is to maximize the expected number of agents that reach a goal state $\langle s, \Psi \rangle \in \hat{\mathrm{G}}$ from the initial state $\langle s_0, \Psi_0 \rangle$. There exist policies π for some CC-SSPs such that $\text{Egoals}(\langle s, \Psi \rangle, \pi)$ is infinite. It is natural to limit policies

such that $\mathrm{Egoals}(\langle s, \Psi \rangle, \pi)$ is finite, and as this is to be used as probability estimate, 1 is a reasonable bound. A *bounded* policy π for a CC-SSP $\mathcal{C}$ is a policy which has $\mathrm{Egoals}(\langle s_0, \Psi_0 \rangle, \pi) \leq 1$. Let the set of all bounded policies for $\mathcal{C}$ be $\Pi_{\mathcal{C}}$. An *optimal bounded* policy π^* for a CC-SSP $\mathcal{C}$ is a policy which has

$$\mathrm{Egoals}(\langle s_0, \Psi_0 \rangle, \pi^*) \geq \mathrm{Egoals}(\langle s_0, \Psi_0 \rangle, \pi)$$

for all $\pi \in \Pi_{\mathcal{C}}$.

Requiring a bounded policy does not reduce the maximum expected agents reaching goals below what it would be otherwise, with the obvious exception of the bound to 1. That is, if there exists a policy π for which $\mathrm{Egoals}(\langle s_0, \Psi_0 \rangle, \pi) > 1$, then an optimal bounded policy π^* can be constructed by removing any infinite loops in the policy (by having agents take the α_{die} action during the loop with some small probability) and making agents take the α_{die} action in the initial states with some probability such that $\mathrm{Egoals}(\langle s_0, \Psi_0 \rangle, \pi^*)$ is reduced to exactly 1.

5.2.4 CC-SSPs as Heuristic Estimation

Given an SSP $\mathcal{S} = \langle \mathrm{S}, s_0, \mathrm{G}, \mathrm{A}, \mathrm{T}, C \rangle$, a state $s \in \mathrm{S}$ and an LTL CNF set Ψ, the probability $\mathsf{Pr}^{\pi^*}(\Psi \mid s)$ that Ψ will be satisfied under any optimal policy π^* for $\mathcal{S}$ is upper bounded by the optimal expected number of agents reaching goals in an associated CC-SSP $\mathcal{C}$. The associated CC-SSP is $\mathcal{C}_s^{\langle s, \Psi \rangle} = \langle \hat{\mathrm{S}}, \langle s, \Psi \rangle, \mathrm{G}, \mathrm{A}, \mathrm{T} \rangle$, where $\hat{\mathrm{S}}$ is the set of augmented states $\mathrm{S} \times 2^{2^{\Sigma(\Psi))}}$.

This upper bounding property is clear from the way agent's paths are defined, and was demonstrated earlier by equation 5.4. An agent entering the state $\langle s, \Psi \rangle$ need only satisfy some clause $\Phi \in \Psi$, and then an agent is placed in an X-literal state for each $\phi \in \Phi$. These agents in turn need only satisfy ϕ, and as agents can independently choose the best actions for satisfying each X-literal, the expected number of agents that reach goals from each of these is at least as much as the probability that ϕ could be satisfied from s.

Let $h_{\mathcal{S}}^{\mathrm{dec}}$ be a heuristic function for a MO-PLTL SSP problem $\mathcal{S}$ called the decomposition heuristic, where

$$h_{\mathcal{S}}^{\mathrm{dec}}(\langle s, \Psi \rangle) \equiv \max_{\pi \in \Pi_{\mathcal{C}}} \mathrm{Egoals}(\langle s, \Psi \rangle, \pi)$$

over the related CC-SSP $\mathcal{C}_s^{\langle s, \Psi \rangle}$. Notice that the underlying SSP for $\mathcal{C}_s^{\langle s, \Psi \rangle}$ is $\mathcal{S}$, and so generating the state space $\hat{\mathrm{S}}$ is at least as complex as generating the full state space S. On the other hand, this relaxation reduces the number of LTL formula considered in the state space, as the number of X-literals derivable from a formula is much less than the number of possible CNF set formula constructed from the same formula.

Firstly, as $h_{\mathcal{S}}^{\mathrm{dec}}$ estimates only the probability for a single PLTL constraint ψ_i, $\hat{\mathrm{S}}$ is much smaller than the state space for the MO-PLTL SSP problem, which has a state space $\mathrm{S} \times 2^{2^{\Sigma(\psi_1)}} \times \cdots \times 2^{2^{\Sigma(\psi_n)}}$. As well as this, while generating the CC-SSP requires all the states in $\hat{\mathrm{S}}$ to be considered, solving a CC-SSP requires only $\tilde{\mathrm{S}}$, which is much

smaller, as $\tilde{\mathrm{S}} = \mathrm{S} \times \Sigma(\psi_i)$.

It was noted above that the underlying SSP for the CC-SSP used in $h^{\mathrm{dec}}_{\mathcal{S}}$ is $\mathcal{S}$, so in the worst case the full state space of $\mathcal{S}$ is generated to compute $h^{\mathrm{dec}}_{\mathcal{S}}$. For any practical SSP, this is intractable, but the decomposition heuristic can benefit from SAS$^+$ and LTL projection as described in chapter 4 to make the state space feasible to compute. Let $h^{\text{s-dec}}_{\mathcal{S},\psi}$ be the decomposition splitting heuristic for an SSP $\mathcal{S}$ constrained by the PLTL constraint ψ. This heuristic maintains a set of batches $\mathcal{V}_{\psi,1}, \mathcal{V}_{\psi,2} \ldots \mathcal{V}_{\psi,n}$ of the set of SAS$^+$ variables $\mathcal{V}$ as returned by algorithm 4 or any other variable grouping algorithm. The heuristic value for a state is found by finding the optimal solution to a CC-SSP for each batch, and the minimum is the result. Formally,

$$h^{\text{s-dec}}_{\mathcal{S},\psi}(\langle s, \Psi \rangle) \equiv \min_i h^{\mathrm{dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}(\langle \mathrm{proj}(s, \mathcal{V}_{\psi,i}), \mathrm{assignFree}(\Psi, \mathcal{V}_{\psi,i}) \rangle)$$

In practice, these heuristics can be used for each state during a search, and are computed using a linear program in the occupation measure dual space. The constraints used to compute $h^{\mathrm{dec}}_{\mathcal{S}}$ can be reused for each state, only changing the initial state of the CC-SSP, so the time necessary to generate the constraints is amortised over all the uses of $h^{\mathrm{dec}}_{\mathcal{S}}$. Furthermore, $\tilde{\mathrm{S}}$ can be generated on-the-fly, as only the subset of $\tilde{\mathrm{S}}$ which is reachable from a given state $\langle s, \Psi \rangle$ is necessary to compute $h^{\mathrm{dec}}_{\mathcal{S}}(\langle s, \Psi \rangle)$.

5.3 LP Formulation

The heuristic value $h^{\mathrm{dec}}_{\mathcal{S}}(\langle s, \Psi \rangle)$ can be calculated similarly to the optimal solution for an SSP. In this thesis, this is done in dual space, by finding the optimal occupation measures. This models the CC-SSP as something like a flow network, where flow entering an X-literal state is representative of agents entering the same X-literal state in expectation. One of the rules, for example, is that agents leaving an X-literal state (other than one of the initial state) must have entered this state, so the flow leaving an X-literal state must be less than or equal to the flow entering it.

The primary advantage of using the dual representation is in the integration of this heuristic with the state-of-the-art planner PLTL-dual, and so will be detailed and explained in the following chapter. For the sake of illustration, the formulation $h^{\mathrm{dec}}_{\mathcal{S}}$ is provided in this section and extended in chapter 6.

5.3.1 Occupation Measures

The linear program for $h^{\mathrm{dec}}_{\mathcal{S}}$ is defined over the occupation measure variables $x_{s,\phi,\alpha}$, where $x_{s,\phi,\alpha}$ represents the expected number of times that action $\alpha \in \mathrm{A}$ is executed in the X-literal state $\langle s, \phi \rangle$ by all agents put together. This is quite a natural representation for CC-SSPs, as the objective is the expected number of agents entering goals, which can be defined quite easily in terms of the expected number times actions which can terminate at goals are executed. Representing the paths of all the agents using an LP also means that the behaviour of all agents is calculated simultaneously.

Formally, an occupation measure represents the expectation

$$x_{s,\phi,\alpha} = \mathbb{E}_{R\in\text{runs}_{\mathcal{C}}(\langle s_0,\Psi_0\rangle,\pi)} \sum_{i=1}^{\ell(R)} |\{p \in \mathcal{P}_i | \, \text{last}(p) = \langle s, \phi\rangle, \mathcal{A}_i(p) = \alpha\}|, \tag{5.5}$$

and we denote the set of all occupation measures

$$\mathrm{X}_{\mathcal{C}} \equiv \{x_{s,\phi,\alpha} \mid \langle s, \phi\rangle \in \tilde{\mathrm{S}}, \alpha \in \mathrm{A}(s) \setminus \{\alpha_{die}\}\}$$

Though the linear program in section 5.3.2 is defined in terms of these occupation measures, the outcome can be interpreted without inferring π from this equivalence.

It is worth noting that there exist bounded policies for which $x_{s,\phi,\alpha}$ would be infinite, which makes finding the associated solution to an LP very impractical. This can occur, for example, when a policy specifies that an agent must loop between two states indefinitely. An infinite loop can even emerge in an optimal bounded policy, as not all agents must reach goals for Egoals(s_0, π) to reach 1, and so unnecessary agents could get trapped in infinite loops. However, it is never necessary to have an occupation measure be infinite, as any such infinite loops can be eliminated by modifying the policy so an agent will take the α_{die} action with some small probability each time around the loop.

Using occupation measures allows the problem to be thought of as a flow problem, with a source (the initial state) and a sink (the absorbing states). In this line of thinking, occupation measures represent expected flow between states, and the α_{die} action allows flow to leak out of the network without reaching the absorbing states. This technique was used to solve SSPs with dead ends in [Trevizan et al., 2017b], with an implicit "give-up" action.

5.3.2 Linear Program

The linear program (LP2) in this section can be used to calculate $h^{\text{dec}}_{\mathcal{S}}$, and can be reused for each state that $h^{\text{dec}}_{\mathcal{S}}$ is evaluated on, only needing to change the source state. Several functions are presented in advance to clarify the behaviour of the network and the purpose of each constraint is described in detail.

The functions to be used in LP2 capture the movement of flow (i.e., movement of agents in expectation) from one X-literal state to the next by actions. They are defined as follows:

$$\text{in}(s,\Psi) \equiv \sum_{\substack{\langle s',\phi\rangle\in\tilde{\mathrm{S}},\alpha\in\mathrm{A}(s')\setminus\{\alpha_{die}\}: \\ \mathrm{CNF}(\text{un-}\mathbf{X}(\phi),s)=\Psi}} x_{s',\phi,\alpha}\mathrm{T}\left(s \mid s',\alpha\right) \qquad \forall\langle s,\Psi\rangle \in \hat{\mathrm{S}}$$

$$\text{out}(s,\phi) \equiv \sum_{\alpha\in\mathrm{A}(s)\setminus\{\alpha_{die}\}} x_{s,\phi,\alpha} \qquad \forall\langle s,\phi\rangle \in \tilde{\mathrm{S}}$$

$$\text{receive}(s,\phi) \equiv \sum_{\substack{\Psi:\exists\Phi\in\Psi, \\ \phi\in\Phi}} \mathsf{Pr}_{\mathcal{C}}\left(\phi \mid \Psi\right)\text{in}(s,\Psi) \qquad \forall\langle s,\phi\rangle \in \tilde{\mathrm{S}}$$

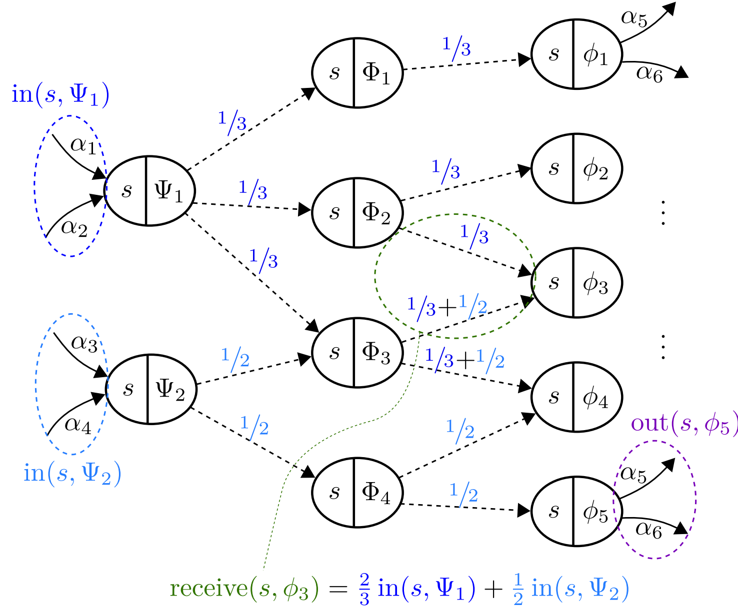


Figure 5.4: The flow being redistributed through a given X-literal state, in a process analogous to redistribution of agents in a CC-SSP.

These functions are depicted in figure 5.4, showing how they capture the movement and redistribution of agents through the flow network. The functions are described as follows:

- $\text{in}(s, \Psi)$ represents the flow passing through the augmented state $\langle s, \Psi \rangle$, defined as the expected flow from actions which can end at $\langle s, \Psi \rangle$ from X-literal states. This can be seen in the left side of figure 5.4, where actions which have some probability of taking an agent to $\langle s, \Psi \rangle$ are summed up. Note that $\text{in}(s, \Psi)$ is not used directly in LP2, and augmented states are not directly represented either. This is analogous to the fact that, in CC-SSPs, agents move only between X-literal states. The abandon action α_{die} is not counted as taking this action does not move an agent to another state. It will instead be represented in the constraints of LP2.

- Similarly, $\text{out}(s, \phi)$ is the flow leaving the X-literal state $\langle s, \phi \rangle$ along all the actions available in that state. This can be seen on the right hand side of figure 5.4, and is the sum of the occupation measures leaving $\langle s, \phi \rangle$.

- Finally, $\text{receive}(s, \phi)$ represents the flow reaching the X-literal state $\langle s, \phi \rangle$, after being redistributed from augmented states. It is defined as the flow entering each augmented state which can redistribute to $\langle s, \phi \rangle$, multiplied by the probability that the augmented state redistributes to this one. This a natural definition of this expectation. Figure 5.4 depicts this in the middle section, with several augmented states feeding into one X-literal state. An important note is that, as the sum $\sum_{\langle s, \phi \rangle \in \text{decompose}(\langle s, \Psi \rangle)} \mathsf{Pr}_{\mathcal{C}}(\phi \mid \Psi)$ can be greater than 1, more flow can be

received by the children of a given augmented state than reaches that augmented state. This is referred to as "flow duplication" and is analogous to the cloning of agents in CC-SSPs.

Note that as the flow into a given X-literal state is dependent on $\mathsf{Pr}\left(\phi \mid \Psi\right)$, it can be considered without specifically enumerating the clauses $\Phi \in \Psi$. Diagrams of flow networks representing CC-SSPs in this thesis therefore omit the states associated with disjunctions that can be seen in the centre of figure 5.4.

Using these functions, an LP can be defined to compute the value of the heuristic $h^{\text{dec}}_{\mathcal{S}}(\langle s_0, \Psi_0\rangle)$, and is defined below as LP2. As mentioned before, this LP can be constructed once when $h^{\text{dec}}_{\mathcal{S}}(\langle s_0, \Psi_0\rangle)$ is first computed, and only the subset of $\tilde{\mathrm{S}}$ (and the associated subset of $\mathrm{X}_{\mathcal{C}}$) reachable from $\langle s_0, \Psi_0\rangle$ is necessary. To compute $h^{\text{dec}}_{\mathcal{S}}(\langle s_0', \Psi_0'\rangle)$[2] for any state in this reachable subset, LP2 can be reused, changing the source constraint (C8) to the new source state. If $h^{\text{dec}}_{\mathcal{S}}(\langle s_0', \Psi_0'\rangle)$ is computed for some $\langle s_0', \Psi_0'\rangle$ not in the generated subset of $\tilde{\mathrm{S}}$, the LP is be extended to include all X-literal states (and associated occupation measures) reachable from this new initial state. LP2 is defined as follows:

$$
\begin{aligned}
\max \quad & sink_{\text{acc}} && && \text{(LP2)}\\
\text{s.t.} \quad & x_{s,\phi,\alpha} \geq 0 && \forall x_{s,\phi,\alpha} \in \mathrm{X}_{\mathcal{C}} && \text{(C5)}\\
& sink_{\text{acc}} \leq 1 && && \text{(C6)}\\
& sink_{\text{acc}} = \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{G}}} \text{in}(s,\Psi) && && \text{(C7)}\\
& \text{out}(s_0,\phi) - \text{receive}(s_0,\phi) \leq \mathsf{Pr}_{\mathcal{C}}\left(\phi \mid \Psi_0\right) && \forall \langle s,\phi\rangle \in \text{decompose}(s_0,\Psi_0) && \text{(C8)}\\
& \text{out}(s,\phi) - \text{receive}(s,\phi) \leq 0 && \forall \langle s,\phi\rangle \in \tilde{\mathrm{S}} \setminus \text{decompose}(s_0,\Psi_0) && \text{(C9)}
\end{aligned}
$$

The heuristic estimate $h^{\text{dec}}_{\mathcal{S}}(\langle s_0, \Psi_0\rangle)$ is the objective $sink_{\text{acc}}$, which represents the expected number of agents reaching goal states. LP2 behaves a lot like a flow network, but with flow duplication as described above. The purpose of each constraint can be categorised as:

Objective Upper Bound (C6). To restrict LP2 to finding only bounded policies, the objective is upper bounded to 1. Obviously, as LP2 is used to estimate a probability, limiting the estimate to be no greater than 1 is a very natural approach.

Objective Definition (C7). To aid with clarity, the objective is denoted $sink_{\text{acc}}$ and is defined to be equal to the expected number of agents reaching absorbing goal states, i.e., states in $\hat{\mathrm{G}}$.

Flow Constraints (C8 - C9). The flow out of an X-literal state must be less than or equal to the flow leaving that state. The abandon action α_{die} is not directly represented, and instead an inequality is used. (C9) is equivalent to $\text{out}(s,\phi) + x_{s,\phi,\alpha_{die}} - \text{receive}(s,\phi) = 0$, but requires one less variable to represent the same relationship, similarly for (C8). Constraint (C8) represents the flow into the source states decom-

[2] Here $\langle s_0', \Psi_0'\rangle$ is used to denote a new state for which the heuristic is computed, but the subscript 0 is used because the heuristic is computed using a CC-SSP for which this state is the initial state.

posed from $\langle s_0, \Psi_0 \rangle$, allowing $\mathsf{Pr}_{\mathcal{C}}\left(\phi \mid \Psi_0\right)$ into each such state as well as allowing flow through the state.

As LP2 calculates $h^{\text{dec}}_{\mathcal{S}}(\langle s_0, \Psi_0 \rangle)$ directly, it is not necessary to extract a policy from the solution found, but if a policy was necessary, it can be computed as $\pi(\langle s, \phi \rangle, \alpha) = \frac{x_{s,\phi,\alpha}}{\text{receive}(s,\phi)}$ for $\alpha \in \mathrm{A}(s) \setminus \alpha_{die}$, and the probability of α_{die} can be found from the implicit leaking of flow; $\pi(\langle s, \phi \rangle, \alpha_{die}) = 1 - \frac{\text{out}(s,\phi)}{\text{receive}(s,\phi)}$.

5.4 Limitations

This approach is not without limitations, the primary limitations are sourced from the cloning of agents, or equivalently the duplication of flow. One such limitation is that loops in the underlying SSP can be greatly abused in CC-SSPs, leading to gross overestimates. As discussed earlier, $h^{\text{dec}}_{\mathcal{S}}$ also expands the entire state space, which can be prohibitively large.

To construct LP2, the entire state space S is expanded, and the linear program contains a constraint for each state in $\tilde{\mathrm{S}}$, which can be larger than S. For any practical problem, S is already exponential, and if it weren't there would be no need for heuristic search. This limitation is averted in the presented implementation in two ways:

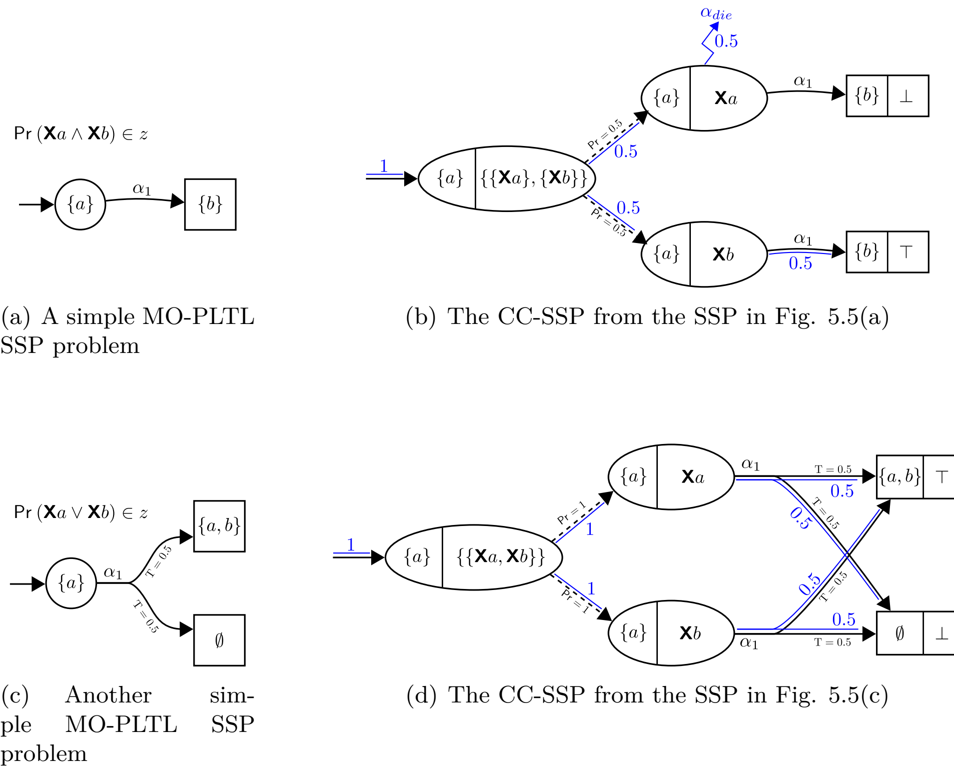
- Expansion of $\tilde{\mathrm{S}}$ stops when reaching a non-goal state s paired with a trivial LTL formula $\top$ or $\bot$, meaning that for simple constraints, large amounts of $\tilde{\mathrm{S}}$ (or even the entire state space) are pruned before computation of LP2.

- Instead of computing $h^{\text{dec}}_{\mathcal{S}}$ for the full underlying SSP $\mathcal{S}$, it can be projected onto a set of batches $\mathcal{V}_{\psi,1}, \mathcal{V}_{\psi,2} \ldots \mathcal{V}_{\psi,n}$, and $h^{\text{dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}$ can be computed for each batch. This is $h^{\text{s-dec}}_{\mathcal{S},\psi}$, and the state space $\tilde{\mathrm{S}}$ of the CC-SSP associated with each heuristic will be drastically smaller, given a reasonable choice of batches. However projecting onto these batches makes the $h^{\text{s-dec}}_{\mathcal{S},\psi}$ much less informative than $h^{\text{dec}}_{\mathcal{S}}$, leading it to return an estimate of 1 in most cases, except when the formula is trivially unsatsifiable.

5.4.1 Innate Limitations of Decomposition

Aside from the issue with intractable state spaces, estimating by decomposing LTL formulae causes drastic overestimates in some cases by its fundamental nature. Decomposition assumes the independence of the X-literals in the constraint, and when they are dependent, the estimate found by h^{dec}_{s} is quite inaccurate. Two similar trivial examples demonstrate this, both consisting of only one action leading from the initial state to goal states.

In figure 5.5, two very similar SSPs are presented, along with their associated CC-SSPs. The probability that $\mathbf{X}a \wedge \mathbf{X}b$ is satisfied from the initial state of the SSP in figure 5.5(a) is zero, as there is only one possible outcome, which satisfies only $\mathbf{X}a$. It can be seen in the associated CC-SSP in figure 5.5(b) that there is a 50% chance that an agent would be redistributed to the state $\langle \{b\}, \mathbf{X}a \rangle$, from which it can satisfy



(a) A simple MO-PLTL SSP problem

(b) The CC-SSP from the SSP in Fig. 5.5(a)

(c) Another simple MO-PLTL SSP problem

(d) The CC-SSP from the SSP in Fig. 5.5(c)

Figure 5.5: These SSPs (and the assocated CC-SSPs to the right) result in significant overestimates from $h_{\mathcal{S}}^{\text{dec}}$ despite how trivial they are.

the X-literal. The occupation measures and flow redistribution for a solution which achieves a 50% estimate are shown in blue in figure 5.5(b). The estimate returned by h^{dec} from the initial state would therefore be 0.5, which is a significant overestimate from 0. This type of overestimate is, however, not an issue for constraints which require that an LTL constraint be satisfied with probability 1. Averaging over 'and' operators alone will always lower the probability estimate if any conjunct has a probability less than 1.

On the other hand, when adding the expected number of agents reaching goals due to 'or' operators, the estimate can reach 1 quite easily. The SSP presented in 5.5(c) demonstrates this, as the probability of satisfying the constraint $\mathbf{X}a \vee \mathbf{X}b$ is obviously 50%. On the other hand, the CC-SSP in figure 5.5(d) essentially adds the probability of satisfying each disjunct independently and as the two disjuncts are not independent, this is a significant overestimate. The end result is an estimate of 1 agent reaching the goal in expectation, equivalent to an estimation of a 100% probability, achieved by the policy shown with occupation measures and flow redistribution shown in blue.

Decomposition is a reasonable relaxation of the problem, and significant overestimates in some situations are to be expected by any heuristic. A heuristic which estimates accurately in all situations likely would take too long to compute to be effective. However, the next section demonstrates how loops in the underlying SSP lead to these properties being abused by repeating the decomposition indefinitely.

5.4.2 Loops

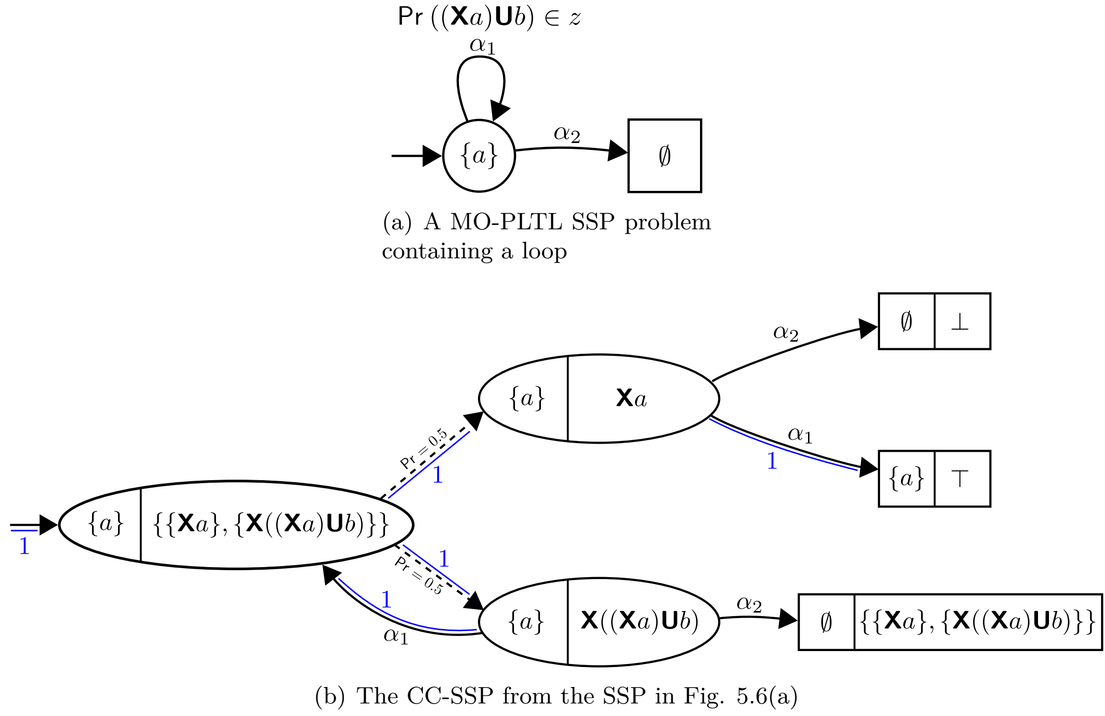
If there is a loop in the underlying SSP, such that an agent can return to the same state with probability 1, then a policy can abuse this by having an agent continue around this loop (almost) indefinitely to repeat the decomposition at one or more of the loop states forever. Two simple cases of loop abuse are presented, one for 'and' decomposition and the other for 'or' decomposition.

The case for 'and' is presented in figure 5.6, where the probability of satisfying the formula $(\mathbf{X}a)\mathbf{U}b$ is obviously 0, as b is not true in any state. Despite this, the CC-SSP presented in figure 5.6(b) has a bounded policy which has 1 agent reaching the only goal state $\langle\{a\},\top\rangle$ in expectation. This policy has agents loop back by action α_1 from the state $\langle\{a\},\mathbf{X}((\mathbf{X}a)\mathbf{U}b)\rangle$ until by chance they are redirected to the state $\langle\{a\},\mathbf{X}a\rangle$. Being redirected to $\langle\{a\},\mathbf{X}a\rangle$ will happen eventually with probability 1. From this state, the action α_1 satisfies $\mathbf{X}a$, ending in the state $\langle\{a\},\top\rangle$. This policy is:

$$\pi(\langle\{a\},\mathbf{X}((\mathbf{X}a)\mathbf{U}b),\alpha_1\rangle) = 1$$
$$\pi(\langle\{a\},\mathbf{X}a\rangle\,,\alpha_1) = 1$$

and the occupation measures for this policy are shown in blue in figure 5.6(b). In this way, the right hand side of the until operator is completely bypassed, even though the semantics of until dictate that the right hand side must eventually become true.

The case for 'or' is presented in figure 5.7, where the probability of satisfying the formula $a\mathbf{U}(b\mathbf{U}c)$ is trivially p, as the constraint can only become true for a path that



(a) A MO-PLTL SSP problem containing a loop

(b) The CC-SSP from the SSP in Fig. 5.6(a)

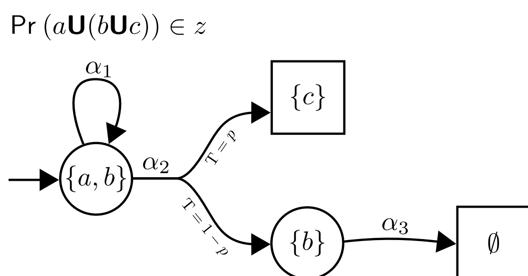
Figure 5.6: A loop in the underlying SSP allows ‘and’ decomposition to be abused to ignore part of the constraint.

reaches the state $\{c\}$. However, for any $p > 0$, these exists a policy for the CC-SSP in figure 5.7(b) which achieves an 1 agent reaching $\{c\}$ in expectation. This policy has an agent go round the loop a sufficiently large number of times in expectation, (at least $\frac{1}{p}$ times) and has all agents entering the state $\langle\{a\}, \mathbf{X}(b\mathbf{U}c)\rangle$ take action α_2. Having agents go around the loop many times in expectation can be achieved with a stationary policy:

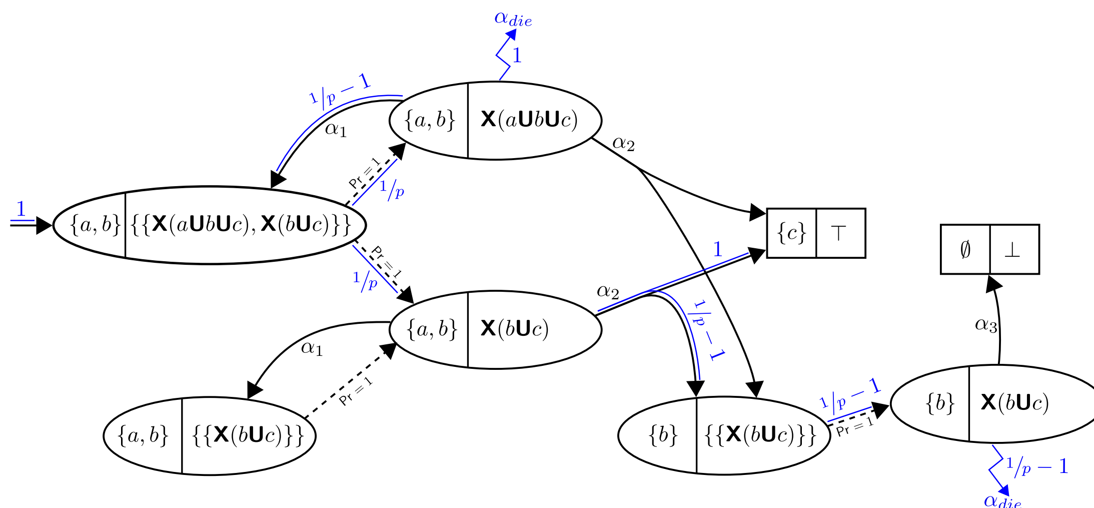
$$\pi(\langle\{a,b\}, \mathbf{X}(a\mathbf{U}(b\mathbf{U}c))\rangle, \alpha_1) = 1 - p$$
$$\pi(\langle\{a,b\}, \mathbf{X}(a\mathbf{U}(b\mathbf{U}c))\rangle, \alpha_{die}) = p$$
$$\pi(\langle\{a,b\}, \mathbf{X}(b\mathbf{U}c)\rangle, \alpha_2) = 1$$
$$\pi(\langle\{b\}, \mathbf{X}(b\mathbf{U}c)\rangle, \alpha_{die}) = 1$$

essentially creating a sufficient number of agents to overcome any low probability of reaching the goal. The occupation measures for this policy are shown in blue in figure 5.7(b). Worryingly, because the linear program considers all the states expanded previously, infinite flow generation can occur even if the loop is *unreachable* from the initial state.

Consider the case where the same linear program is being used later to compute $h^{\text{dec}}_{\mathcal{S}}(\langle\{b\}, \{\{\mathbf{X}(b\mathbf{U}c)\}\}\rangle)$, in this case the only reachable states are $\langle\{b\}, \mathbf{X}(b\mathbf{U}c)\rangle$ and $\langle\emptyset, \bot\rangle$, so obviously the estimate should be 0, however the linear program as it was



(a) A MO-PLTL SSP problem containing a loop



(b) The CC-SSP from the SSP in Fig. 5.7(a)

Figure 5.7: A loop in the underlying SSP allows 'or' decomposition to be abused to create any number of agents necessary.

presented in this chapter allows the unreachable loop from $\{a, b\}$ to itself to have any amount of flow going around it. This puts flow into $\langle\{a\}, \mathbf{X}(b\mathbf{U}c)\rangle$, and this flow can reach the goal state $\langle\{c\}, \top\rangle$, even though this goal state is unreachable from the initial state. Consider for example the following assignment to the occupations measures:

$$x_{\{a\},\mathbf{X}(a\mathbf{U}(b\mathbf{U}c)),\alpha_1} = \frac{1}{p}$$
$$x_{\{a\},\mathbf{X}(b\mathbf{U}c),\alpha_2} = \frac{1}{p}$$

where the $\frac{1}{p}$ units of flow entering $\langle\{a, b\}, \mathbf{X}(b\mathbf{U}c)\rangle$ are leaked immediately. Note how the flow entering the X-literal state $\langle\{a\}, \mathbf{X}(a\mathbf{U}(b\mathbf{U}c))\rangle$ is equal to the flow leaving it. The term flow is used here instead of describing the movement of agents, as no run from the initial state $\langle\{a, b\}, \{\{\mathbf{X}(b\mathbf{U}c)\}\}\rangle$ can replicate these occupation measures, and yet they form a valid solution to LP2. This particular issue can be resolved by not reusing the linear program, but a similar issue exists where a reachable loop spontaneously generates flow, allowing most of a CC-SSP to be skipped, even though the loop was reachable from the initial state, demonstrated schematically in figure 5.8.

Clearly LP2 is not a faithful formulation of a CC-SSP, but sufficiently emulates one outside of this issue of loops. Spontaneous generation of flow from loops can be averted by extending the LP, and a similar issue has been studied and resolved in the research of the Travelling Salesman Problem, where loops on their own are referred to as subtours. A discussion of two of these approaches can be found in [Desrochers and Laporte, 1991], but in summary the techniques add extra constraints and sometimes extra variables, one restricts that any two states must be connected (equivalently in the case of this thesis that states which are included in the policy must be connected), and the other enforces an ordering such that no loops can occur apart from at the starting point. Neither of these approaches are directly applicable, but adding further constraints and variables can prevent spontaneous flow generation.

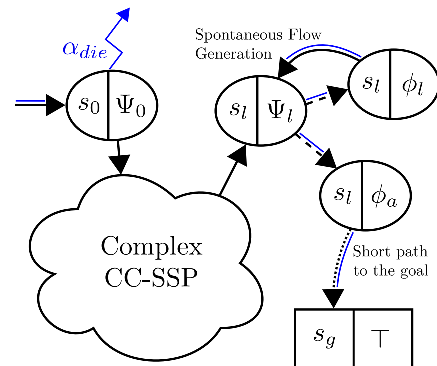


Figure 5.8: A loop which an agent can travel around with probability 1 can spontaneously generate flow, allowing portions of a CC-SSP to be skipped.

5.5 Summary

Estimating the probability of satisfying an LTL formula while reaching the goal of an SSP is an important task for improving planning under PLTL constraints, and a fast estimation can be used as a heuristic to guide a planner towards a valid plan. Decomposition of formulae in CNF is proposed as a method to estimate this probability,

where the probability of satisfying each X-literal in the CNF formula is individually computed. The probability of satisfying the full CNF formula can then be estimated by arithmetic on the probabilities of the individual X-literals. It is not sufficient to perform a single decomposition, and instead decomposition is performed after each formula progression. To simulate this, a planning problem called a CC-SSP is introduced which simulates decomposition at each step by cloning and redistributing agents between decomposed states.

The novel heuristic $h^{\mathrm{dec}}_{\mathcal{S}}$ generates a CC-SSP from the SSP $\mathcal{S}$ and an LTL formula, and estimates the probability of satisfying the given LTL formula in the SSP as the optimal objective value of the CC-SSP. CC-SSPs benefit from having a smaller state space in terms of the LTL formula compared to an MO-PLTL SSP problem, but the state space can still be prohibitively large for a heuristic. The heuristic $h^{\mathrm{s\text{-}dec}}_{\mathcal{S},\Psi}(s)$ chooses the minimum value $h^{\mathrm{dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}(s)$ over a number of projections of the SSP $\mathcal{S}$, where the state space of each is significantly smaller than the CC-SSP for $h^{\mathrm{dec}}_{\mathcal{S}}$.

Policies in CC-SSPs are evaluated by how many agents reach the goal states in expectation, which lends them to an occupation measure representation, and to this end a linear program is presented which simulates a given CC-SSP through the occupation measures. Using a linear program has some advantages, but introduces issues with spontaneous flow generation. These advantages will be exploited in chapter 6, in which $h^{\mathrm{dec}}_{\mathcal{S}}$ will be adapted for integration with the state of the art planner PLTL-dual. By happy coincidence, this adaptation also resolves the spontaneous flow generation problem.

Chapter 6

Decomposition Heuristic for Planning

The ultimate goal of this thesis is to introduce an effective heuristic for MO-PLTL SSP problems which when used in the state of the art planner PLTL-dual will be sufficiently informative and efficient that it can be used to improve scalability of PLTL-dual. Two admissible heuristics which upper-bound the probability of satisfying an LTL formula from a state were presented in chapter 5, but PLTL-dual improves the usage of heuristics in several ways.

To integrate a heuristic into PLTL-dual, it must interface correctly with the planner and other heuristics. Specifically it must be defined for a weighted distribution of initial states, and it must allow for tying constraints. These are described in section 6.1.

To make it more informative and allow for tying constraints, an extension for h_S^{dec} to trace the actions taken by agents which reach goals is described in section 6.2, the resulting heuristic for PLTL-dual is presented in section 6.3, including an extension of the linear program defined in chapter 5 to compute this new heuristic.

6.1 Interfacing with PLTL-dual

Heuristics in PLTL-dual must meet a few extra requirements beyond simply estimating the probability of satisfying a constraint from a state. PLTL-dual iteratively extends the known state space, and uses a linear program to find the optimal solution to the partial problem defined by the known state space, taking into account the heuristic value of each fringe state. Unlike other heuristic search algorithms, the heuristic for each fringe state is calculated in tandem with solving the partial problem, so the heuristic must be defined as a linear program which can be included into the linear program for PLTL-dual.

6.1.1 Weighted Initial States

One of the ways heuristics in PLTL-dual can be more efficient is that they are computed once over all the reachable states in the fringe, rather than being each computed individually. PLTL-dual determines the expected movement of an agent (flow) through

the known state space and agents either reach known goal states, or reach the fringe, from which heuristics are computed. To be usable in PLTL-dual, a heuristic must be defined as a linear program over a set of initial states $\hat{\mathrm{S}}_{\mathrm{init}}$, as the solver may find a policy which reaches multiple states on the fringe of the known state space. For each of these states there is some probability that an agent reaches that fringe state, and these probabilities may not add up to one, as there may be a probability that agents reach goals inside the known state space. Let $\mathbf{P}_{s,\Psi}$ be the probability that an agent reaches a fringe state $\langle s, \Psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}$, referred to the weight on $\langle s, \Psi\rangle$ or equivalently the flow into $\langle s, \Psi\rangle$. This weighting $\mathbf{P}$ is distinct from a probability distribution, as

$$0 \leq \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}} \mathbf{P}_{s,\Psi} \leq 1\,.$$

The case where $\sum_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}} \mathbf{P}_{s,\Psi} = 0$ is important as it implies that no agents reach the fringe of the known state space, so PLTL-dual has found a closed policy and can terminate, and the heuristic itself need not be computed.

Computing a heuristic for a weighted initial state is semantically different from computing a heuristic for a single initial state. Consider a heuristic h^{ψ} for a PLTL constraint ψ defined over a weighted set of initial states $\hat{\mathrm{S}}_{\mathrm{init}}$. Under the assumption that $\mathbf{P}_{s,\Psi}$ agents reach each state $\langle s, \psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}$ in expectation, h^{ψ} estimates the expected number of these agents which satisfy each formula ψ from the associated state s such that $\langle s, \psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}$. Such a heuristic h^{ψ} is admissible if this is an overestimate.

Extending $h^{\mathrm{dec}}_{\mathcal{S}}$ to have a weighted initial state is quite straight forward. Given a set of initial augmented states $\hat{\mathrm{S}}_{\mathrm{init}} \subset \hat{\mathrm{S}}$ and a weighting $\mathbf{P}$ over the states $\langle s, \Psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}$,

$$h^{\mathrm{dec}}_{\mathcal{S}}(\hat{\mathrm{S}}_{\mathrm{init}}, \mathbf{P}) \equiv \max_{\pi \in \Pi_{\mathcal{C}}} \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}} (\mathbf{P}_{s,\Psi} \times \mathrm{Egoals}(\langle s, \Psi\rangle, \pi))\,.$$

Equivalently, one can imagine that instead of an agent starting a specific augmented state in the CC-SSP, the initial state $\langle s, \Psi\rangle$ is chosen at random with probability $\mathbf{P}_{s,\Psi}$, and with probability $1 - \left(\sum_{\langle s,\psi\rangle \in \hat{\mathrm{S}}_{\mathrm{init}}} \mathbf{P}_{s,\Psi}\right)$, no initial state is chosen and the agent fails to reach a goal.

6.1.2 Tying Constraints

Computing all the heuristics simultaneously allows them to depend on each other, and this dependence is captured in PLTL-dual with tying constraints. Tying constraints are typically of the form

$$\sum_{s \in \mathrm{S}_{h_1}} x^{h_1}_{s,\alpha} = \sum_{s \in \mathrm{S}_{h_2}} x^{h_2}_{s,\alpha}$$

where h_1 and h_2 are two different heuristics, S_{h_i} is the state space used to compute heuristic h_i, and $x^{h_i}_{s,\alpha}$ represents an occupation measure used to compute h_i. Tying constraints exist between one heuristic and all other heuristics (i.e., tying h_1 to h_2, h_1 to h_3, h_1 to h_4 and so on), and exist for each action $\alpha \in \mathrm{A}$. Tying constraints intuitively require that all actions deemed necessary by some heuristic are performed

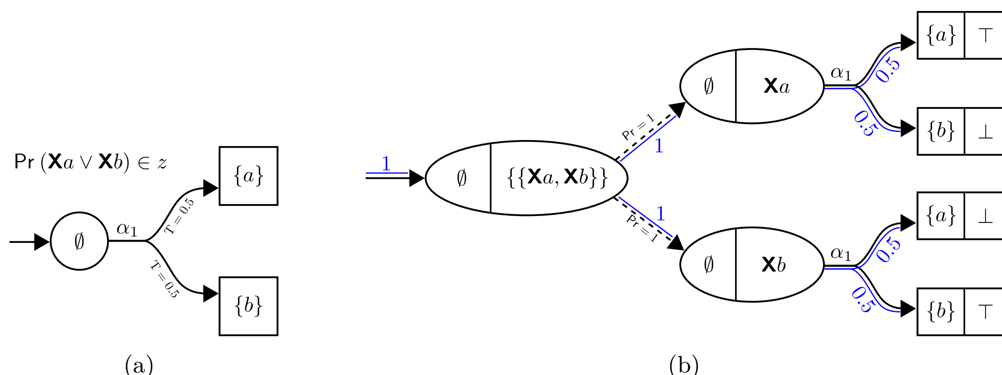


Figure 6.1: This MO-PLTL SSP problem and its associated CC-SSP demonstrate that tying constraints cannot be applied naïvely to $h_{\mathcal{S}}^{\text{dec}}$.

in other heuristics, potentially influencing them, but different heuristics can have these actions performed in different orders.

As an example of tying constraints, consider the projection heuristics for a problem where an agent must push an object between two rooms, `r1` and `r2`. The agent can be in either room; `pos` = `r1` or `pos` = `r2`, and similarly for the object position, `obj-pos`. Let the goal be the assignment (`obj-pos` = `r1`), and the initial state be (`obj-pos` = `r2`), (`pos` = `r1`). The projection onto `pos` is disjoint from the goal, so the artificial goal action α_g can be taken anywhere, making this projection trivial. However, the projection onto `obj-pos` requires that the action '`push r2 r1`' is taken, to move the object to `r1`. The tying constraints would require that this action is taken in the other projection. Suppose '`push r2 r1`' had the precondition `pos` = `r2` and the effect `pos` = `r1`. To make this possible in the projection onto `pos`, that projection would have to first do the action '`move r1 r2`'. This in turn makes the projection onto `obj-pos` perform the '`move r1 r2`' action, but because the action has effects and preconditions disjoint from `obj-pos`, it can do this action at any time in the plan. Obviously, for more complicated problems, the combined projection heuristics will not find the full plan, but only a relaxed one.

For PLTL constraint heuristics, tying constraints provide much more information to the search than the probability estimate from the heuristic, meaning that including tying constraints in $h_{\mathcal{S}}^{\text{dec}}$ is vital for its efficacy. Tying constraints for $h_{\mathcal{S}}^{\text{dec}}$ are not possible without some modification, as the occupation measures from multiple agents add up to more uses of a given action than is necessary in the actual solution, and sometimes more than is possible. Consider the SSP in figure 6.1(a), and the constraint $\mathsf{Pr}\left(\mathbf{X}a \vee \mathbf{X}b\right) = 1$. Obviously, there is only one policy, which is to take the action α_1 from the state $\emptyset$ with a 0.5 chance of reaching each goal state, either of which satisfies the formula. In the CC-SSP for this problem, an agent starts in each of $\langle\emptyset, \mathbf{X}a\rangle$ and $\langle\emptyset, \mathbf{X}b\rangle$, and the only policy $\pi \in \Pi_{\mathcal{C}}$ with $\text{Egoals}(\langle\emptyset, \{\{\mathbf{X}a, \mathbf{X}b\}\}\rangle, \pi) = 1$ is shown in the form of occupation measures and flow redistribution in blue in figure 6.1(b). Under this policy, the action α_1 is taken once by each agent no matter what, so the expected

number of times it is taken is twice. With a naïve inclusion of tying constraints, this would force the cost heuristics to take action α_1 twice in expectation, which is not only impossible, but if it were possible it would make those heuristics overestimate the true cost of reaching the goal, meaning they are inadmissible.

Clearly tying constraints cannot be applied directly to h_S^{dec} without some further modification. Observe in figure 6.1(b) that if only the actions taken by agents that eventually reach goals (i.e., half the agents leaving each $\langle\emptyset,\cdot\rangle$ X-literal state) are counted, then action α_1 is taken only once in expectation. This concept is called tracing accepting flow, and introduces several improvements to h_S^{dec}.

6.2 Tracing Accepting Flow

In a CC-SSP, it is conceivable that under a policy π, with a small probability ϵ, *one* agent reaches an X-literal state $\langle s,\phi\rangle$ from which many of its clones reach goal states. In this case, if $\text{Egoals}(\langle s,\phi\rangle) \geq \frac{1}{\epsilon}$, then this small probability alone is sufficient to push the probability estimate up to 1. Similarly, when there is a loop in the CC-SSP near a goal, spontaneous flow generation can skip the majority of the CC-SSP, even when no flow from the initial state reaches this loop. These are obviously flaws in the design of h_S^{dec}, and it would increase the informativeness of h_S^{dec} if they were resolved in some way.

It is necessary to allow that multiple clones of a single agent can reach a goal to maintain the admissibility of h_S^{dec}; recall figure 6.1(b), in which it is possible for two agents to reach the goals $\langle\{a\},\top\rangle$ and $\langle\{b\},\top\rangle$ in the same run, and if this were prevented in any way, $\text{Egoals}(\langle\emptyset,\{\mathbf{XF}a,\mathbf{XF}b\}\rangle,\pi)$ would be less than 1. However, a natural restriction is that as Egoals represents a recursive probability estimate, it should be bounded to 1 recursively.

An **internally bounded** policy π is a policy such that $\text{Egoals}(\langle s,\Psi\rangle,\pi) \leq 1$ for all $\langle s,\Psi\rangle \in \hat{\text{S}}$ and $\text{Egoals}(\langle s,\phi\rangle,\pi) \leq 1$ for all $\langle s,\phi\rangle \in \tilde{\text{S}}$. As an intuitive example, if under some internally bounded policy π, 0.5 agents reach some X-literal state $\langle s,\phi\rangle$ in expectation, then at most 0.5 agents can reach goals in expectation having gone through $\langle s,\phi\rangle$.

The concept of *tracing accepting flow* is introduced to compute internally bounded policies in the occupation measure dual space. The idea behind an accepting flow trace is that the flow reaching the goal states of a CC-SSP should be traced backwards to the source state, and that this trace should not allow any flow duplication. This trace is defined by labelling portions of the occupation measures in the flow network as *accepting flow*, and as a trace, the amount of accepting flow entering a state should be equal to the accepting flow leaving it.

To put the above example in terms of accepting flow, if 0.5 units of accepting flow are traced from goal states back to some X-literal state $\langle s,\phi\rangle$, then exactly 0.5 units of accepting flow can be traced from $\langle s,\phi\rangle$ to the source state. If there was no such trace to the source, then the policy π encoded by the occupation measures, is not an internally bounded policy, as $\text{Egoals}(\langle s,\phi\rangle,\pi) > 1$.

Tracing accepting flow not only increases the informativeness of the estimate derived

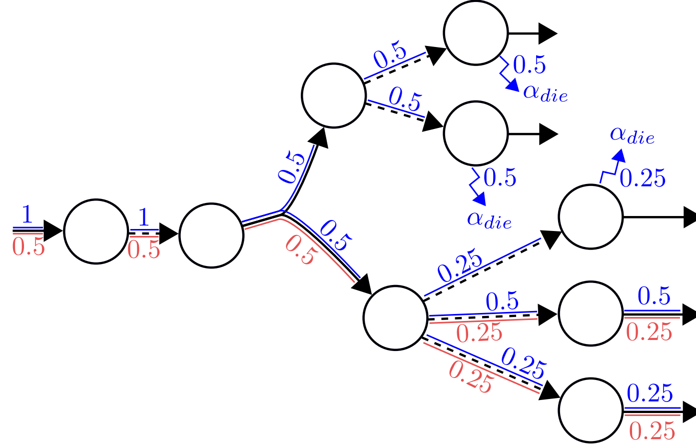


Figure 6.2: An example trace of accepting flow through part of a CC-SSP. Flow is labelled in blue, and accepting flow labelled in red.

from a CC-SSP, but also provides an expression that can be used for tying constraints, and resolves the issues of unreachable loops and spontaneous flow generation identified in section 5.4.2.

Figure 6.2 shows an example trace of accepting flow through part of a CC-SSP, with accepting flow labelled in red and flow labelled in blue. The action and state labels have been omitted for clarity. Observe how the accepting flow is conserved between entering and leaving a state, allowing the flow at the end to be traced backwards towards the source.

6.2.1 Determinisation

To label specific parts of the flow through the network representing a CC-SSP as accepting flow, the occupation measures are split up into components in a way akin to the all-outcomes determinisation for probabilistic planning. This is necessary as the flow along a probabilistic action α may lead to multiple different states, where only agents directed to one such specific state go on to reach goal states. Similarly, in CC-SSPs, agents can be cloned into multiple X-literal states, and it is possible that only some of these clones will go on to reach goal states, or that if they did it would not be under an internally bounded policy. As such, tracing accepting flow requires that each possible outcome can be traced independently.

To this end, accepting flow is labelled by two sets of variables. Formally, given a CC-SSP $\mathcal{C} = \langle \hat{\mathrm{S}}, \langle s_0, \Psi_0 \rangle, \mathrm{G}, \mathrm{A}, \mathrm{T} \rangle$ with X-literal state space $\tilde{\mathrm{S}}$ and underlying state space S, these sets of variables are

$$\tilde{\mathrm{Y}}_{\mathcal{C}} \equiv \{y_{s,\phi,\alpha,s'} \mid \langle s, \phi \rangle \in \tilde{\mathrm{S}}, \alpha \in \mathrm{A}(s) \setminus \{\alpha_{die}\}, s' \in \mathrm{S} \text{ s.t. } \mathrm{T}\left(s' \mid s, \alpha\right) > 0\}$$
$$\hat{\mathrm{Y}}_{\mathcal{C}} \equiv \{y_{s,\Psi,\phi} \mid \langle s, \Psi \rangle \in \hat{\mathrm{S}}, \phi \text{ s.t. } \langle s, \phi \rangle \in \mathrm{decompose}(s, \Psi)\}.$$

$\tilde{\mathrm{Y}}_{\mathcal{C}}$ is the set of variables $y_{s,\phi,\alpha,s'}$ labelling the accepting flow along actions from X-literal states, and, for each action and state, there is a variable for each outcome of

that action. A variable $y_{s,\phi,\alpha,s'} \in \tilde{\Upsilon}_{\mathcal{C}}$ labels what quantity of the flow from $\langle s, \phi \rangle$ to $\langle s', \mathrm{CNF}(\text{un-}\mathbf{X}(\phi), s') \rangle$ by the action α is accepting flow. On the left hand side of figure 6.2, accepting flow only travels down one of the outcomes of the probabilistic action, as none of the flow reaching the other outcome ever reaches accepting states.

Similarly, $\hat{\Upsilon}_{\mathcal{C}}$ is the set of variables $y_{s,\Psi,\phi}$ which label accepting flow from augmented states to X-literal states. A variable $y_{s,\Psi,\phi} \in \hat{\Upsilon}_{\mathcal{C}}$ labels what quantity of the flow redistributed from $\langle s, \Psi \rangle$ to $\langle s, \phi \rangle$ is accepting flow. That is, all the flow which would enter $\langle s, \Psi \rangle$ is redistributed (and duplicated) into states in decompose(s, Ψ), and from some of those states there may be accepting flow. The variables in $\hat{\Upsilon}_{\mathcal{C}}$ trace accepting flow backwards by specifying which augmented states the flow into a given X-literal state comes from. On the lower right side of figure 6.2, the flow into only some of the redistribution states is labelled as accepting flow, as not all states have flow going to goals.

6.2.2 Upper Bounding

The variables in $\tilde{\Upsilon}_{\mathcal{C}}$ and $\hat{\Upsilon}_{\mathcal{C}}$ represent portions of the flow represented by the occupation measures in $\mathrm{X}_{\mathcal{C}}$. Because of this, they are obviously upper bounded by their associated occupation measures. For the variables in $\tilde{\Upsilon}_{\mathcal{C}}$, this is a straightforward upper bound proportional to the probability of the given outcome:

$$y_{s,\phi,\alpha,s'} \leq x_{s,\phi,\alpha} \cdot \mathrm{T}\left(s' \mid s, \alpha\right),$$

however there are no occupation measures directly associated with the labels in $\hat{\Upsilon}_{\mathcal{C}}$, but the amount of flow being redistributed from any augmented state to a specific X-literal state can be defined in terms of $\mathrm{in}(s, \Psi)$, which is a function of the occupation measures representing actions that would end at $\langle s, \Psi \rangle$, defined in section 5.3.2. Using $\mathrm{in}(s, \Psi)$, the upper bound for these variables is

$$y_{s,\Psi,\phi} \leq \mathrm{in}(s, \Psi) \cdot \mathsf{Pr}\left(\phi \mid \Psi\right).$$

This upper bounding property is can be observed in several places for the accepting flow shown in figure 6.2, primarily on the left hand side, where the accepting flow for the whole system is upper bounded by the flow into one outcome of a probabilistic action.

6.2.3 Spontaneous Flow Generation

One of the primary limitations of h_S^{dec} was that of spontaneous flow generation. Under an internally bounded policy, spontaneous flow generation technically can occur, but any flow that reaches goals must be traced back to the source state, so spontaneous flow generation cannot be responsible for flow reaching goals. This rules out the case described in section 5.4.2 (see figure 5.8) in which flow is immediately leaked upon entering the network and a loop near a goal state creates sufficient flow to have 1 unit of flow entering goal states. Similarly, the case described in the same section where

an unreachable loop generates flow to reach a goal state unreachable from the source state cannot occur.

6.2.4 Tying Constraints

Tying constraints between the projection heuristics in PLTL-dual account for the movement of the agent whether or not it satisfies the LTL formula. On the other hand, accepting flow considers only the movement of agents which eventually reach goal states of a CC-SSP, which are associated with the satisfaction of the LTL formula. This requires that tying constraints between a CC-SSP flow network and the projection heuristics must use an inequality, with accepting flow lower bounding the flow in the projection heuristics. As an extreme example, consider the case where it is impossible for the LTL formula to be satisfied from some non-goal state. In this case, there would be no accepting flow in the CC-SSP network, and the projection heuristics would have to take some actions to reach the goal. Under an equality constraint, this case would cause a contradiction. Similarly, because there is no guarantee that either of two CC-SSPs allow more accepting flow than the other, there cannot be any form of tying constraint between them.

Tying constraints between the accepting flow variable $\tilde{\mathrm{Y}}_{\mathcal{C}}$ and the occupation measures in another heuristic h_2 therefore are of the form

$$\sum_{y_{s,\phi,\alpha',s'} \in \tilde{\mathrm{Y}}_{\mathcal{C}} : \alpha' = \alpha} y_{s,\phi,\alpha',s'} \leq \sum_{s \in \mathrm{S}_{h_2}} x^{h_2}_{s,\alpha} \tag{6.1}$$

for all $\alpha \in \mathrm{A}$. Note that because the variables in $\hat{\mathrm{Y}}_{\mathcal{C}}$ are not associated with actions, they are not necessary for tying constraints.

Now that tying constraints have been proposed, as well as the notion of a weighted initial state, a heuristic to be integrated into PLTL-dual will be defined in the next section.

6.3 LP Formulation

Inclusion of accepting flow tracing in the definition $h^{\text{dec}}_{\mathcal{S}}$ results in a new heuristic called the traced decomposition heuristic $h^{\text{t-dec}}_{\mathcal{S}}$. For a pre-specified SSP $\mathcal{S}$, $h^{\text{t-dec}}_{\mathcal{S}}$ estimates the probability of satisfying given LTL CNF sets Ψ from associated states s while also reaching the goal of $\mathcal{S}$ by constructing and solving the linear program LP3 defined in section 6.3.1. The initial states and CNF sets are defined as a set $\hat{\mathrm{S}}_{\text{init}}$ of augmented states $\langle s, \Psi \rangle$, and a probability distribution $\mathsf{Pr}_{\hat{\mathrm{S}}_{\text{init}}}(\langle s, \Psi \rangle)$. The estimate derived by $h^{\text{t-dec}}_{\mathcal{S}}$ is the objective of LP3; $sink_{\text{acc}}$. This linear program is an extension of LP2 to include accepting flow tracing constraints, a weighted initial state, and when integrated with PLTL-dual, also includes tying constraints.

Just like $h^{\text{dec}}_{\mathcal{S}}$, the state space of $h^{\text{t-dec}}_{\mathcal{S}}$ is exponential, so SAS$^+$ and LTL projection is employed to generate relaxed SSPs $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that $h^{\text{t-dec}}_{\mathcal{S}_i}$ is not too computationally expensive to use as a heuristic for each $\mathcal{S}_i$. Let $h^{\text{sdt}}_{\mathcal{S},\Psi}$ be the split decomposition

trace heuristic, defined for some SSP $\mathcal{S}$ constrained by the PLTL constraint ψ. Similarly to $h^{\text{s-dec}}_{\mathcal{S},\psi}$, $h^{\text{sdt}}_{\mathcal{S},\psi}$ is defined over a set of batches $\{\mathcal{V}_{\psi,1}, \mathcal{V}_{\psi,2}, \ldots, \mathcal{V}_{\psi,n}\}$ as found by algorithm 4 or any other variable grouping algorithm, and computes $h^{\text{t-dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}$ for each $\mathcal{V}_{\psi,i}$, returning the minimal estimate found this way.

In the context of PLTL-dual however, $h^{\text{sdt}}_{\mathcal{S},\psi}$ constructs the linear program for $h^{\text{t-dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}$ for each $\mathcal{V}_{\psi,i}$ it maintains, and each is tied by tying constraints to the projection heuristics for PLTL-dual. These linear programs are effectively each part of the overall linear program for PLTL-dual.

6.3.1 Linear Program

The extension of LP2 to trace accepting flow is presented in this section, along with several functions introduced to make the definition of the linear program clearer. Similarly to LP2 in section 5.3.2, this linear program can be constructed once the first time $h^{\text{t-dec}}_{\mathcal{S}}(\hat{\mathrm{S}}_{\text{init}}, \mathbf{P})$ is computed, constructing only constraints for states reachable from each $\langle s, \Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}$, and is reused each time the heuristic is computed. In the case that $h^{\text{t-dec}}_{\mathcal{S}}(\hat{\mathrm{S}}'_{\text{init}}, \mathbf{P}')$ is to be computed, but some state in $\hat{\mathrm{S}}'_{\text{init}}$ is not yet generated, the linear program is extended to include constraints for all states reachable from each of the new initial states in $\hat{\mathrm{S}}'_{\text{init}}$.

Four functions are introduced for LP3 and are used to represent accepting flow leaving and entering both X-literal states and augmented states, with two to represent accepting flow in and out of each. They are defined as follows:

$$\text{accIn}(s, \Psi) = \sum_{\substack{\langle s',\phi\rangle \in \tilde{\mathrm{S}}, \alpha \in \mathrm{A}(s') \setminus \{\alpha_{die}\}: \\ (\text{CNF}(\text{un-}\mathbf{X}(\phi), s) = \Psi) \wedge (\mathrm{T}(s|s',\alpha) > 0)}} y_{s',\phi,\alpha,s} \qquad \forall \langle s, \Psi\rangle \in \hat{\mathrm{S}}$$

$$\text{accRedist}(s, \Psi) = \sum_{\phi : \langle s,\phi\rangle \in \text{decompose}(s,\Psi)} y_{s,\Psi,\phi} \qquad \forall \langle s, \Psi\rangle \in \hat{\mathrm{S}}$$

$$\text{accReceive}(s, \phi) = \sum_{\Psi : \langle s,\phi\rangle \in \text{decompose}(s,\Psi)} y_{s,\Psi,\phi} \qquad \forall \langle s, \phi\rangle \in \tilde{\mathrm{S}}$$

$$\text{accOut}(s, \phi) = \sum_{\substack{\alpha \in \mathrm{A}(s) \setminus \{\alpha_{die}\}, s' \in \mathrm{S}: \\ \mathrm{T}(s'|s,\alpha) > 0}} y_{s,\phi,\alpha,s'} \qquad \forall \langle s, \phi\rangle \in \tilde{\mathrm{S}}$$

These functions are very similar in definition and function to the ones for LP2. Notice however that because there are variables for augmented states, the flow through augmented states is represented with two functions separate from those for X-literal states, where LP2 uses only two functions for X-literal states. These functions are listed below:

- $\text{accIn}(s, \Psi)$ and $\text{accRedist}(s, \Psi)$ represent accepting flow through the augmented state $\langle s, \Psi\rangle$. Accepting flow coming in is defined in terms of variables $y_{s',\phi,\alpha,s} \in \hat{\Upsilon}_{\mathcal{C}}$, where s is the outcome of an action from the state $\langle s', \phi\rangle$. Flow out is represented as redistribution variables $y_{s,\Psi,\phi} \in \hat{\Upsilon}_{\mathcal{C}}$ redirecting flow to individual literal states in $\text{decompose}(s, \Psi)$.

- $\text{accReceive}(s, \phi)$ and $\text{accOut}(s, \phi)$ represent the accepting flow being distributed through the literal state $\langle s, \Psi \rangle$ from and to augmented states. Conversely to the flow through augmented states, the flow in is defined in terms of $\hat{\mathrm{Y}}_{\mathcal{C}}$, and the flow out is defined in terms of $\tilde{\mathrm{Y}}_{\mathcal{C}}$.

Combining accepting flow with a weighted initial state add a slight challenge for $h_{S}^{\text{t-dec}}$. Accepting flow is traced in such a way that the amount of accepting flow entering and leaving the network is identical. However, in the case that the amount of flow reaching goal states is less than that entering the network, only some of the flow entering the network will be labelled as accepting flow. The linear program must include constraints that specify that the accepting flow entering the network is equal to the accepting flow leaving, which for a single initial state is simple, the accepting flow entering at the source state would be exactly the amount leaving the network, e.g.,

$$\text{accRedist}(s_0, \Psi_0) - \text{accIn}(s_0, \Psi_0) - sink_{\text{acc}} = 0.$$

However, this is not applicable when there are multiple initial states, as it should be possible to trace accepting flow to each, but the total amount of flow leaving them should be exactly equal to $sink_{\text{acc}}$. To resolve this issue, LP3 contains an extra variable $i_{s,\Psi}$ for each state $\langle s, \Psi \rangle \in \hat{\mathrm{S}}_{\text{init}} \setminus \hat{\mathrm{F}}$, representing what portion of the flow entering the network at that particular state is accepting flow.

With the functions defined above, LP3 is defined over the variables $\mathrm{X}_{\mathcal{C}}$, $\tilde{\mathrm{Y}}_{\mathcal{C}}$, $\hat{\mathrm{Y}}_{\mathcal{C}}$ and $i_{s,\Psi}$ for $\in \hat{\mathrm{S}}_{\text{init}} \setminus \hat{\mathrm{F}}$ and is presented in figure 6.3. If LP3 is computed in conjunction with other heuristics (i.e., in PLTL-dual), it can include the tying constraints in equation 6.1 for each action $\alpha \in \mathrm{A}$, tied to any heuristic h_2 other than another instance of this one.

Before the constraints in LP3 are explained individually, consider first a holistic view of LP3. LP3 computes an internally bounded policy for a CC-SSP using a pair of flow networks. The primary flow network uses the variables in $\mathrm{X}_{\mathcal{C}}$, which are occupation measures for the CC-SSP, representing the expected movement of agents through the CC-SSP, and includes flow duplication and potentially spontaneous flow generation. The secondary flow network uses the variables in $\tilde{\mathrm{Y}}_{\mathcal{C}}$ and $\hat{\mathrm{Y}}_{\mathcal{C}}$, and is defined over exactly the same state space as the primary network. The secondary network network labels the flow in the primary network which is accepting flow, i.e., flow which will eventually reach states in $\hat{\mathrm{G}}$. In its entirety, the secondary network provides a trace of accepting flow from the source states to the goal states, with no flow duplication. As all the flow in the secondary network must exist in the primary network, the flow in the secondary network is upper-bounded by the flow in the primary network. Below, each constraint in LP3 is explained and fit into this explanation.

Accepting Flow Source (C14). The accepting flow entering the network at each source state can be no more than the flow reaching that state. This is clear from the definitions of both.

Objective Definition (C15). The flow leaving the network is defined to be $sink_{\text{acc}}$, which is equivalent also to the sum of all flow reaching goal states in each network. If a goal state is in the set of initial states $\hat{\mathrm{S}}_{\text{init}}$, which can occur primarily when

$$
\begin{aligned}
\max \quad & sink_{\text{acc}} && \text{(LP3)}\\
\text{s.t.} \quad & x_{s,\phi,\alpha} \geq 0 & \forall x_{s,\phi,\alpha} \in \mathrm{X}_{\mathcal{C}} \quad & \text{(C10)}\\
& y_{s,\phi,\alpha,s'} \geq 0 & \forall y_{s,\phi,\alpha,s'} \in \tilde{\mathrm{Y}}_{\mathcal{C}} \quad & \text{(C11)}\\
& y_{s,\Psi,\phi} \geq 0 & \forall y_{s,\Psi,\phi} \in \hat{\mathrm{Y}}_{\mathcal{C}} \quad & \text{(C12)}\\
& 0 \leq i_{s,\Psi} \leq 1 & \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}}_{\text{init}} \setminus \hat{\mathrm{F}} \quad & \text{(C13)}\\
& i_{s,\Psi} \leq \mathbf{P}_{s,\Psi} & \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}}_{\text{init}} \setminus \hat{\mathrm{F}} \quad & \text{(C14)}\\
& sink_{\text{acc}} - \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}} \cap \hat{\mathrm{G}}} \mathbf{P}_{s,\Psi} = \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{G}}} \mathrm{in}(s,\Psi) = \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{G}}} \mathrm{accIn}(s,\Psi) = \sum_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} i_{s,\Psi} & & \text{(C15)}\\
& \mathrm{out}(s,\phi) - \mathrm{receive}(s,\phi) \leq \sum_{\Psi : \langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} \mathbf{P}_{s,\Psi} \cdot \mathsf{Pr}\left(\phi \mid \Psi\right) & \forall \langle s, \phi \rangle \in \bigcup_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} \mathrm{decompose}(s,\Psi) \quad & \text{(C16)}\\
& \mathrm{out}(s,\phi) - \mathrm{receive}(s,\phi) \leq 0 & \forall \langle s, \phi \rangle \in \tilde{\mathrm{S}} \setminus \bigcup_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} \mathrm{decompose}(s,\Psi) \quad & \text{(C17)}\\
& \mathrm{accRedist}(s,\Psi) - \mathrm{accIn}(s,\Psi) - i_{s,\Psi} = 0 & \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}}_{\text{init}} \setminus \hat{\mathrm{F}} \quad & \text{(C18)}\\
& \mathrm{accRedist}(s,\Psi) - \mathrm{accIn}(s,\Psi) = 0 & \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}} \setminus (\hat{\mathrm{S}}_{\text{init}} \cap \hat{\mathrm{F}}) \quad & \text{(C19)}\\
& \mathrm{accOut}(s,\phi) - \mathrm{accReceive}(s,\phi) = 0 & \forall \langle s, \phi \rangle \in \tilde{\mathrm{S}} \quad & \text{(C20)}\\
& y_{s,\phi,\alpha,s'} - x_{s,\phi,\alpha} \cdot \mathrm{T}\left(s' \mid s, \alpha\right) \leq 0 & \forall y_{s,\phi,\alpha,s'} \in \tilde{\mathrm{Y}}_{\mathcal{C}} \quad & \text{(C21)}\\
& y_{s,\Psi,\phi} - \mathrm{in}(s,\Psi) \cdot \mathsf{Pr}\left(\phi \mid \Psi\right) \leq \sum_{\Psi : \langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} \mathbf{P}_{s,\Psi} \cdot \mathsf{Pr}\left(\phi \mid \Psi\right) & \forall \langle s, \phi \rangle \in \bigcup_{\langle s,\Psi\rangle \in \hat{\mathrm{S}}_{\text{init}}} \mathrm{decompose}(s,\Psi) \quad & \text{(C22)}\\
& y_{s,\Psi,\phi} - \mathrm{in}(s,\Psi) \cdot \mathsf{Pr}\left(\phi \mid \Psi\right) \leq 0 & \forall \langle s, \Psi \rangle \in \hat{\mathrm{S}} \setminus \{\langle s_0, \Psi_0 \rangle\}, \phi : \exists \Phi \in \Psi, \phi \in \Phi \quad & \text{(C23)}
\end{aligned}
$$

Figure 6.3: An LP to compute the traced decomposition heuristic, simulating flow through a CC-SSP and tracing the flow which reaches accepting states.

the formula for an initial state is $\top$, then agents starting at that initial state will reach a goal without moving. This cannot be represented using occupation measures, so $sink_{\text{acc}}$ is artificially increased by the weight of all initial goal states. The final equality requires that the total accepting flow entering the network, represented with the variables $i_{s,\Psi}$ for each $\langle s, \Psi \rangle \in \hat{\mathrm{S}}_{\text{init}}$, is equal to the accepting flow leaving the network. Note that the previous network required that the objective was upper bounded at 1, but the inclusion of the secondary network automatically creates this upper bound. Specifically the combination of constraint C14 and C15 create this upper bound.

Primary Flow Network (C16-C17). The representation of flow through the network for the variables in $\mathrm{X}_{\mathcal{C}}$. (C16) represents the flow into the initial states, allowing an extra $\mathbf{P}_{s,\Psi} \times \mathsf{Pr}\left(\phi \mid \Psi\right)$ flow to enter the network. This is the amount of flow reaching the state $\langle s, \Psi \rangle$ from external sources redistributed to this X-literal

state. In the same way as constraints C8 and C9, these constraints use inequalities to represent the α_{die} action.

Secondary Flow Network (C18-C20). The flow network for the variables in $\tilde{\mathrm{Y}}_{\mathcal{C}}$ and $\hat{\mathrm{Y}}_{\mathcal{C}}$. This flow network uses an all-outcomes determinisation to label the flow to each outcome individually. In contrast to the primary flow network, the constraints use equality, as the α_{die} action guarantees that an agent will not reach a goal, and there is no duplication of flow from disjunctions. C18-C19 encode the flow through augmented states, allowing the LP solver to choose which decomposed states flow should go to. In C18 flow into the secondary flow network at any one state in $\hat{\mathrm{S}}_{\mathrm{init}}$ is set equal to $i_{s,\Psi}$. C20 encodes the flow through literal states, allowing the solver to choose exactly which augmented states flow should go to.

Accepting Flow Upper Bound (C21-C23). As explained in section 6.2.2, these constraints require that the flow through the secondary network is upper bounded by the flow in the primary network. Specifically, they require that for each outcome of an action or for each X-literal state that flow is redistributed to, the flow for that outcome or state in the secondary network is less than the associated fraction of the flow in the primary network. The most important case of this is C22, where the flow being redistributed from a *source state* is accounted for, increasing the upper-bound on $y_{s,\Psi,\phi}$ by $\mathbf{P}_{s,\Psi} \cdot \mathsf{Pr}\,(\phi \mid \Psi)$.

6.4 Limitations

The primary limitation of this approach is that tracing accepting flow introduces a very large number of extra constraints and variables. This naïve form of the all-outcomes determinisation can result in an exponential increase in the number of actions, as was observed in [Rintanen, 2003]. In the context of this heuristic, this equates to potentially an exponential number of variables in the linear program. As well as this, one of the advantages of $h^{\mathrm{dec}}_{\mathcal{S}}$ was that $\hat{\mathrm{S}}$ was not directly represented, and the much smaller space $\tilde{\mathrm{S}}$ was represented using constraints. Recall that $\hat{\mathrm{S}}$ is equal to $\mathrm{S} \times 2^{2^{\Sigma(\psi)}}$ in the worst case, where $\tilde{\mathrm{S}}$ is only $\mathrm{S} \times \Sigma(\psi)$ in the worst case. LP3 includes several variables and constraints for each state in $\hat{\mathrm{S}}$, making it much more sensitive to the size of the LTL constraint.

$h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ averts this issue somewhat by projecting both the state space and the formulae onto subsets of $\mathcal{V}_{\psi}$, but the number of constraints produced to compute $h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ can still be very large, and when the structure of the formula forces there to be many overlapping minimal combinations of $\mathcal{V}_{\psi}$, $h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ will make many copies of the same state space. As well as new issues, some of the limitations of $h^{\mathrm{dec}}_{\mathcal{S}}$ persist in $h^{\mathrm{t\text{-}dec}}_{\mathcal{S}}$.

Like $h^{\mathrm{dec}}_{\mathcal{S}}$, $h^{\mathrm{t\text{-}dec}}_{\mathcal{S}}$ can abuse loops in the underlying state space, though only so much as an internally bounded policy will allow. The cases shown in figures 5.6 and 5.7 still result in the same significant overestimates for $h^{\mathrm{t\text{-}dec}}_{\mathcal{S}}$, primarily because these loops have 1 unit of flow entering them. With an accepting flow trace, if such a loop could only be reached by f units of flow, then at most f units of accepting flow can pass through that loop.

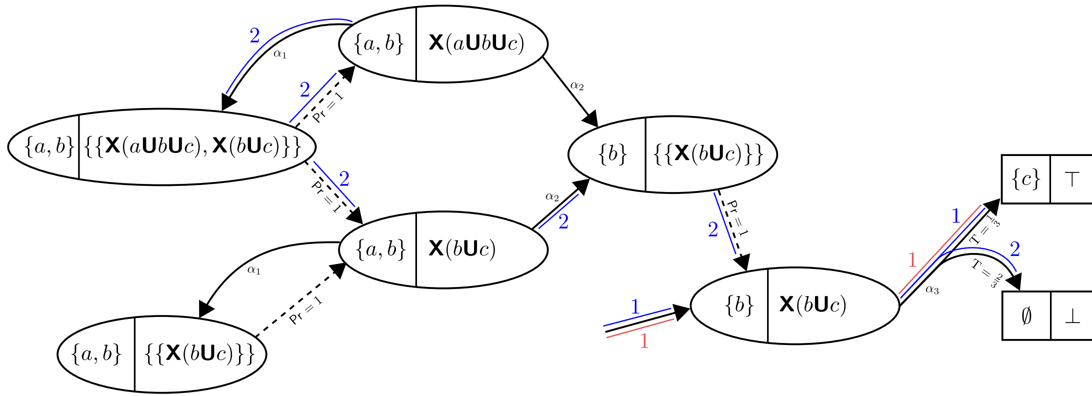


Figure 6.4: A case where spontaneous flow generation from an unreachable loop contributes to the accepting flow trace shown in red.

Section 6.2.3 discussed how adding the accepting flow tracing constraints prevented most cases of spontaneous flow generation from increasing the heuristic estimate, however there are still some cases where spontaneous flow generation creates flow that contributes to accepting flow. Figure 6.4 shows a CC-SSP generated with the LTL constraint $a\mathbf{U}(b\mathbf{U}c)$ from the state $\{a, b\}$, but where the heuristic is computed from the augmented state $\langle b, \{\{\mathbf{X}(b\mathbf{U}c)\}\}\rangle$. There is a loop unreachable from this initial state which allows spontaneous flow generation, which feeds into $\langle b, \{\{\mathbf{X}(b\mathbf{U}c)\}\}\rangle$. Only 1/3 of the flow leaving $\langle b, \{\{\mathbf{X}(b\mathbf{U}c)\}\}\rangle$ from the action α_3 reaches the goal state $\langle c, \top\rangle$, so without the spontaneous flow generation leading into that state, there would only be 1/3 units of flow reaching $\langle c, \top\rangle$. Instead, because of the spontaneous flow generation, the estimate found by $h_{\mathcal{S}}^{\text{t-dec}}$ is 1.

The issue of spontaneous flow generation could possibly be averted by forcing all the occupation measures leading into a state to be equally responsible for accepting flow, meaning that all ways the flow enters a state which has accepting flow leaving it ($\langle b, \{\mathbf{X}(b\mathbf{U}c)\}\rangle$ in this case) would have to be traced back to some initial state. Adding this constraint would prevent spontaneously generated flow from reaching any state which had accepting flow leaving it, i.e., any state from which flow reaches goals.

6.5 Summary

Constructing a heuristic based on decomposition which can be integrated into PLTL-dual requires that the heuristic is constructed as a linear program accepting a weighed set of initial states from which the heuristic is computed, and inclusion of tying constraints to extract extra information out of the heuristic. A weighted initial state is straightforward to include, but $h_{\mathcal{S}}^{\text{dec}}$ does not lend itself to tying constraints. To resolve this and other issues the concept of an internally bounded policy and an accepting flow trace were introduced.

To enforce that the policy found by a linear program is internally bounded, accept-

ing flow is defined such that if f units of flow reach goal states of a CC-SSP from some specific state, then f units of flow must have reached that state from the source state. Equivalently, if f units of accepting flow leave a specific state, then f units of accepting flow must enter it, and this accepting flow can be traced forwards and backwards to the goal states and source respectively.

Including a concept of accepting flow resolves some issues that existed in $h^{\text{dec}}_{\mathcal{S}}$ related to spontaneous flow generation, and tying constraints can be defined on the accepting flow trace, meaning that $h^{\text{t-dec}}_{\mathcal{S}}$, which is $h^{\text{dec}}_{\mathcal{S}}$ with the addition of accepting flow tracing and a weighted initial state, can be integrated into PLTL-dual.

Like $h^{\text{dec}}_{\mathcal{S}}$, the state space for $h^{\text{t-dec}}_{\mathcal{S}}$ is often intractable, so when integrated into PLTL-dual, instead $h^{\text{sdt}}_{\mathcal{S},\psi}$ is used, which computes $h^{\text{t-dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}$ over several projections $\mathcal{V}_{\psi,i}$ of $\mathcal{V}_{\psi}$ simultaneously. Each of these instances of $h^{\text{t-dec}}_{\mathcal{S}_{\mathcal{V}_{\psi,i}}}$ is tied to the projection heuristics for PLTL-dual.

In the next chapter, the results of an empirical evaluation of $h^{\text{sdt}}_{\mathcal{S},\psi}$ are presented. In these experiments, $h^{\text{sdt}}_{\mathcal{S},\psi}$ is compared with the state-of-the-art planner PLTL-dual and other competitive solvers for MO-PLTL SSP problems on several planning domains, and the results are analysed.

Chapter 7

Experiments

Empirical experiments were performed to analyse the effectiveness of $h^{\text{sdt}}_{\mathcal{S},\psi}$ in comparison to other approaches, including h^{BA} in PLTL-dual, i^2-dual with the trivial heuristic, and the PRISM model checker [Kwiatkowska et al., 2011]. Baumgartner et al. [2018] found success when PLTL-dual used a mixture of h^{BA} and the trivial heuristic, not using h^{BA} for constraints where the NBA was larger than 100 states. The experiments compare against this variant also.

The benchmark domains and experimental setup are detailed in section 7.1, and the results are presented in section 7.2. The results are then analysed in section 7.3.

7.1 Setup

The experiments were performed on three domains, two from [Baumgartner et al., 2018] and one adapted from [Trevizan et al., 2016]. The experiments were run on Ubuntu 16.04 with an Intel i7-7700 CPU clocked at 3.60 GHz, and used one thread. Gurobi version 8.1.1 [Gurobi Optimization, 2019] was used for optimizing LPs. Problems are defined in the probabilistic STRIPS fragment of PPDDL, which was converted to SAS^+ using the Fast Downward translator [Helmert, 2006]. LTL formulae are converted to NBAs using the LTL3BA converter [Babiak et al., 2012].

7.1.1 Algorithms

Because of the large number of problems, each algorithm was run only once on each problem instance in each domain, time limited to 20 minutes per instance. The algorithms compared are

1. i^2-dual using the trivial heuristic and the progression mode. Baumgartner et al. [2018] found this algorithm to be quite competitive, as it doesn't suffer as much as h^{BA} as the size of the formulae increases. They also found that it expands fewer states than i^2-dual using the NBA mode with the trivial heuristic, so the progression mode is used over the NBA mode. This algorithm is listed as "Trivial heuristic" in the results.

2. PLTL-dual using h^{BA} and the NBA mode. This is the only other heuristic algorithm for MO-PLTL SSP problems with a heuristic that depends on the LTL

mode state. This algorithm is referred to as "NBA heuristic".

3. PLTL-dual using the NBA mode and using h^{BA} for constraints where the size of the NBA is 100 states or less. This was the most effective algorithm in the experiments by Baumgartner et al. [2018], showing that the NBA heuristic does improve the effectiveness of i^2-dual. This variant is referred to as "NBA heuristic (100)".

4. PLTL-dual using the progression mode and $h^{\text{sdt}}_{\mathcal{S},\psi}$ for all constraints $\psi_i \in \phi$. By a minor abuse of terminology, this is listed as "Decomposition heuristic" in the results and is referred to as such throughout this chapter.

5. The PRISM model checker [Kwiatkowska et al., 2011], version 4.5. This was the state-of-the-art for MO-PLTL SSP problems before PLTL-dual was introduced, and no comparison exists for the priority search domain, so it included for completeness.

7.1.2 Domains

Experiments were done on problems from each of the following 3 domains.

Factory. The factory domain models a production line of n reliable or unreliable machines, where machine m_i makes part p_i, and with the exception of m_1, part p_{i-1} is consumed when using machine m_i. For an unreliable machine m_i, using it fails to make part p_i with probability 0.2, despite still consuming p_{i-1}. Each problem has k unreliable machines, being m_2 to m_{k+1}, and problems are indexed as 'n-k'. Using a machine m_i requires it to be on, and for machines other than m_1, p_{i-1} must be in stock. The actions and costs are: turning a machine on (cost: 1); turning a machine off (cost: 1); using m_1 (cost: 4); using m_i, (cost: 3 if m_i is unreliable, 5 otherwise). The initial state has no parts produced and all machines off, and the goal state is to have p_n produced and all machines off.

There are two PLTL constraints for the factory domain. The first is that m_1 must eventually be on ($\mathsf{Pr}\,(\mathbf{F}(m_1 = \mathsf{on})) = 1$). The second is that, starting at machine m_1 when m_1 is turned on, an imaginary baton is passed down the line, where a machine m_i with the baton must not be turned off until m_{i+1} is on, and upon turning off m_i, the baton is passed to m_{i+1} and m_i must never be turned on again. When m_n has the

baton, no requirement is placed on m_n. This is written

$$\begin{aligned}\psi_{f_2} \equiv \mathbf{G}[(m_1 = \mathsf{on}) \rightarrow & ((m_1 = \mathsf{on})\mathbf{U} \\ & ((m_2 = \mathsf{on}) \wedge \mathbf{G}\neg(m_1 = \mathsf{on})\wedge \\ & ((m_2 = \mathsf{on})\mathbf{U} \\ & ((m_3 = \mathsf{on}) \wedge \mathbf{G}\neg(m_2 = \mathsf{on})\wedge \\ & \quad \dots \\ & ((m_{n-1} = \mathsf{on})\mathbf{U} \\ & ((m_n = \mathsf{on}) \wedge \mathbf{G}\neg(m_{n-1} = \mathsf{on}))) \dots)))))]\end{aligned}$$

The probability bound for the second constraint is $\mathsf{Pr}\,(\psi_{f_2}) = 1$.

Wall-e. The Wall-e domain models the movement of two robots, Wall-e and Eve, moving through a chain of n connected locations $l_1, \dots, l_n$, where each location l_i is connected to a room r_i. At each step, either robot can move to an adjacent room or location, or if they are together, they can move simultaneously to the same location or room. All actions cost 1, except for leaving a room together, which costs 5. All actions are deterministic, except for leaving a room together, where they may bump into each other and both fail to leave the room with probability 0.1. The initial state has Wall-e in l_1 and Eve in r_2, and the goal has Wall-e in l_n

There are 5 PLTL constraints for the Wall-e domain. The first is that they must eventually be together:

$$\mathsf{Pr}\,(\mathbf{F}(\mathtt{together} = \top)) \in [0.5, 1].$$

The second is that Wall-e cannot enter any rooms except r_n twice:

$$\mathsf{Pr}\left(\bigwedge_{i<n} \mathbf{G}((\mathtt{wpos} = r_i) \rightarrow ((\mathtt{wpos} = r_i)\mathbf{U}\mathbf{G}\neg(\mathtt{wpos} = r_i)))\right) \in [0.8, 1],$$

where `wpos` is the position of Wall-e. The remaining three are that Eve must stay in her starting room until Wall-e comes to fetch her:

$$\mathsf{Pr}\,((\mathtt{eve\text{-}in\text{-}room} = \top)\mathbf{U}(\mathtt{together} = \top)) \in [0.8, 1],$$

once they meet, they want to (eventually) be together forever:

$$\mathsf{Pr}\,(\mathbf{G}((\mathtt{together} = \top) \rightarrow \mathbf{FG}(\mathtt{together} = \top))) = 1,$$

and that Eve must visit the first three rooms:

$$\mathsf{Pr}\,(\mathbf{F}(\mathtt{epos} = r_1) \wedge \mathbf{F}(\mathtt{epos} = r_2) \wedge \mathbf{F}(\mathtt{epos} = r_3)) = 1.$$

Priority Search. The priority search domain is adapted from the Search and Rescue domain in [Trevizan et al., 2016]. This domain features a robot in an $n \times n$ grid, locating

missing victims of a disaster. A location $l_{i,j}$ can either be unknown, i.e., it is not known if there is a victim there, or it is known that there is no victim at $l_{i,j}$. When at an unknown location, the robot can search that location, though unlike the Search and Rescue domain, the robot is incapable of rescuing people alone, so no further interaction is required beyond searching unknown locations. Whenever the robot decides, it can end its mission by returning to its starting location and using the end mission action. The robot can travel to the 4 adjacent locations at one of three speeds: slow, normal and fast. The normal and fast movement actions fail with probability 0.05 and 0.1 respectively, in which case the robot does not move its current position. The costs represent time, and are: search an unknown location (cost: 1); move (cost: 1 for fast, 2 for normal speed, and 4 for slow); end the mission (cost: 1). Problems are randomly initialised, with the robot in a random location, and each location is labelled as unknown with probability $p \in \{0.25, 0.5, 0.75\}$. Let the set of initially unknown locations for a search priority problem be U. The following PLTL constraint requires that all locations in U are searched, thus preventing the agent from performing the goal action before searching for victims.

$$\mathsf{Pr}\left(\bigwedge_{l_{i,j}\in U}\mathbf{F}\neg(l_{i,j}\text{-}\mathsf{unknown}=\top)\right)=1,$$

The PLTL constraints for this problem are defined in terms of a randomly chosen “danger zone”. The danger zone is randomly chosen given a problem initialisation, and is a vertical or horizontal line of $n-1$ contiguous locations $DZ = \{l_{i,j}, l_{i,j+1} \ldots l_{i,j+n-2}\}$ or $DZ = \{l_{i,j}, l_{i+1,j} \ldots l_{i+n-2,j}\}$ that does not contain the initial location but does contain at least one unknown location. The danger zone is also never along the edge of the grid. There are two PLTL constraints, first is that the locations in the danger zone have priority over others. The robot must search these first before searching anywhere else:

$$\mathsf{Pr}\left(\left(\bigwedge_{l_{i,j}\in U\setminus DZ}(l_{i,j}\text{-}\mathsf{unknown}=\top)\right)\mathbf{U}\left(\bigwedge_{l_{i,j}\in DZ}(l_{i,j}\text{-}\mathsf{unknown}=\top)\right)\right)=1,$$

and the second is that the agent must not stay within the danger zone for more than two steps, for its own operational safety:

$$\psi_{ps_3}\equiv\mathbf{G}\left[\left(\bigvee_{l_{i,j}\in DZ}(\mathtt{loc}=l_{i,j})\right)\rightarrow\left(\neg\bigvee_{l_{i,j}\in DZ}\mathbf{X}(\mathtt{loc}=l_{i,j})\vee\neg\bigvee_{l_{i,j}\in DZ}\mathbf{XX}(\mathtt{loc}=l_{i,j})\right)\right].$$

The probability bound on the second constraint is more forgiving than the first; $\mathsf{Pr}\,(\psi_{ps_3}) \in [0.8, 1]$. Moving hastily out of the danger zone has the possibility of leaving the robot stuck in the zone for an extra step, violating this constraint.

Because priority search problems are randomly generated, 10 problems of each parameterisation $n \in \{4, 5\}$, $p \in \{0.25, 0.5, 0.75\}$ were generated, and each was solved once with each algorithm. To demonstrate the importance of batch choice when allo-

cating from $\mathcal{V}_\psi$, a variant of $h^{\text{sdt}}_{\mathcal{S},\psi}$ was also included in experiments on the priority search domain. This variant used the batches returned by algorithm 4, except for the first constraint, where the set of batches generated by algorithm 4 is $\{\{l_{i,j}\text{-}\texttt{unknown}\} \mid l_{i,j} \in U\}$, this variant instead uses the batches $\{\{l_{i,j}\text{-}\texttt{unknown}, \texttt{pos}\} \mid l_{i,j} \in U\}$. This is denoted "Decomposition heuristic + position" in the results.

7.2 Results

The results of the experiments are presented in the form of the amount of time in seconds it took to each algorithm to solve each problem once. The time taken for the factory and Wall-e domains are plotted in figure 7.1, where the x axis represents the problem size, and the y axis represents the number of seconds on a log scale. The measured values in seconds can be found in appendix A. Missing data points are those for which that algorithm was cut off by the 20 minute time limit, notably for both NBA heuristic and NBA heuristic (100) on factory problem 7-6 and 8-4. The PRISM model checker also suffered memory cut-offs for Wall-e problems with $n \geq 6$, and factory problems with $n \geq 5$.

The time taken for the Priority Search domain is tabulated in table 7.1, which shows for each parameterisation, the coverage of that algorithm, i.e., how many of the 10 random problems were completed within the 20 minute time limit. The mean time of those is listed, along with a 0.95 confidence interval. The mean and confidence interval are N/A when no problems were completed by that algorithm for those parameters. The case $n = 5, p = 0.75$ is omitted from the table as no algorithms completed any problems generated with those parameters within the time limit.

The number of states expanded is a relevant metric for comparing the informativeness of heuristics, but CPU time is the primary metric on which heuristics are usually compared in the literature, as it captures the trade-off between informativeness and speed, and the practical purpose of a heuristic is ultimately to make search faster. To prevent clutter, the number of states expanded by each algorithm (except PRISM) are presented in appendix A.

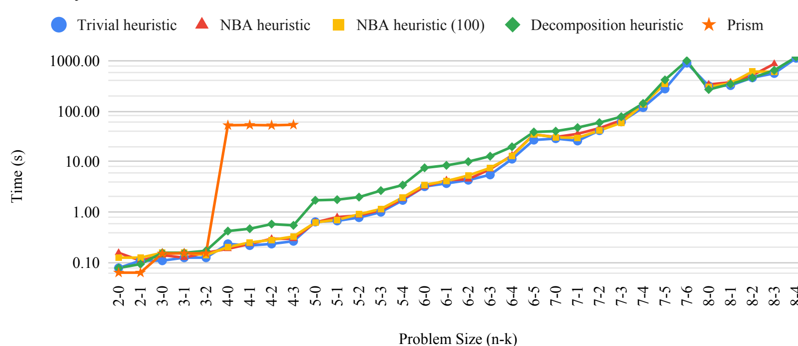
7.3 Discussion

The reader may note that there is no experimentation here for $h^{\text{s-dec}}_{\mathcal{S},\psi}$. This is because, while experiments were performed to determine how informative $h^{\text{s-dec}}_{\mathcal{S},\psi}$ is, the probability estimate $h^{\text{s-dec}}_{\mathcal{S},\psi}(\langle s, \Psi\rangle)$ is indistinguishable from the trivial heuristic on all three domains. In fact, the probability estimate found by $h^{\text{sdt}}_{\mathcal{S},\psi}(\langle s, \Psi\rangle)$ and the NBA heuristics are also equivalent to the trivial heuristic in practice. This is evidence that most of the information PLTL-dual extracts from heuristics is through the tying constraints, rather than from the probability estimate.

The results for the factory and Wall-e domains match those found by Baumgartner et al. [2018] for the most part, except that in the Wall-e domain, the NBA heuristic capped at 100 states does not dominate the i^2-dual with the trivial heuristic, only

	Trivial heuristic	NBA heuristic	NBA heuristic (100)	Decomposition heuristic	Decomposition heuristic + position	PRISM
$n = 4, p = 0.25$						
coverage	10	8	9	10	10	10
average time (s)	1.78	632.51	768.66	5.47	6.37	1.76
0.95 CI	0.24	213.42	219.43	0.49	1.96	0.07
$n = 4, p = 0.50$						
coverage	10	0	10	10	10	10
average time (s)	92.27	N/A	323.79	197.63	59.95	59.49
0.95 CI	41.38	N/A	142.20	84.45	36.50	4.75
$n = 4, p = 0.75$						
coverage	0	0	0	0	6	0
average time (s)	N/A	N/A	N/A	N/A	555.50	N/A
0.95 CI	N/A	N/A	N/A	N/A	367.85	N/A
$n = 5, p = 0.25$						
coverage	10	0	10	10	10	10
average time (s)	36.57	N/A	105.52	105.36	69.76	32.56
0.95 CI	18.24	N/A	54.09	40.72	18.56	1.91
$n = 5, p = 0.50$						
coverage	0	0	0	0	3	0
average time (s)	N/A	N/A	N/A	N/A	288.52	N/A
0.95 CI	N/A	N/A	N/A	N/A	192.53	N/A

Table 7.1: Time in seconds to complete each parametrisation of the priority search domain, aggregated over 10 random problems.



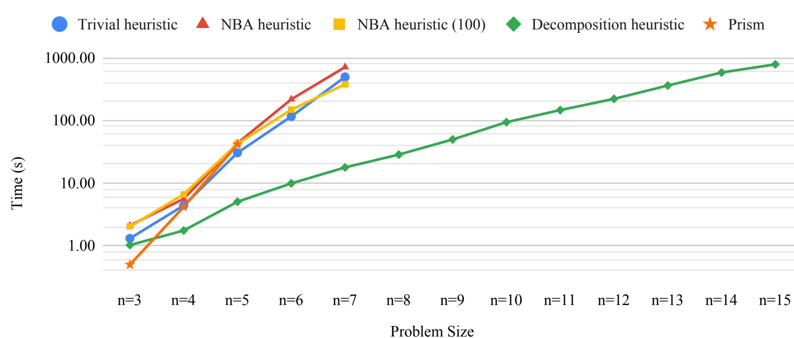


Figure 7.1: The solution time in seconds of the factory and Wall-e domains plotted against problem size.

computing a solution faster when $n = 7$. This can be attributed to the fact that the experiments in this thesis were performed only once, leading to noisy results. Furthermore, there were two bugs found in their implementation, one of which made the progression representation not canonical, potentially reducing the efficiency of the progression mode. The results for the factory domain differ only in that the shorter time limit of 20 minutes caused more time-outs than they observed with their 30 minute time limit.

As is expected and was observed in [Baumgartner et al., 2018], the PRISM model checker performs very well on small problems, but cannot scale up, primarily because it runs out of memory while generating large DRAs. Interestingly, in the priority search domain, PRISM was competitive with the trivial heuristic in all cases. This may be because the DRAs for the priority search domain are not too large, e.g., the DRAs for $n = 4, p = 0.50$ have 256, 3 and 5 states respectively.

7.3.1 Wall-e Domain

The most striking result for the decomposition heuristic is the result in Wall-e, where it completes problems of over twice the size in the same time as any other algorithm. Even for the largest problems it solved, it expanded fewer states than the NBA heuristic (100) did. For $n = 7$, the NBA heuristic (100) expanded 9144 states, where the decomposition heuristic expanded 1543, an 83% improvement, and for the largest problem the decomposition heuristic solved ($n = 15$), it only expanded 5889 states.

To analyse how the decomposition heuristic helps solve Wall-e so much compared to the trivial heuristic, first the behaviour of the projection heuristics on the Wall-e domain should be considered. As the goal is simply to have Wall-e reach location l_n, the projection heuristic for Wall-e's location will find an optimal policy for this problem without constraints, and the projection onto Eve's location and the other variables will not need to do anything. Hence, the trivial heuristic solver will first rapidly search states in which Wall-e moves through locations l_1 to l_n and Eve does not move, however Wall-e needs to enter room r_2 to fetch Eve with probability 0.8, and Eve must move around. The projection heuristics, however, provide no information about how close any one state is to this goal, so Eve blindly searches around.

From this, the success of $h^{\text{sdt}}_{\mathcal{S},\psi}$ in the Wall-e domain can likely be attributed primarily to the fifth constraint: that Eve must enter the first three rooms. The decomposition heuristic will project this constraint onto one batch, containing Eve's position, in which 50% of agents will have Eve go to room r_1 and 50% of agents will have Eve go to room r_3. This 50% is enough that the planner will greatly prefer policies which have Eve travel towards one of her objectives. Constraint one and three, which can both only be satisfied if Eve and Wall-e are together, will require that an action that moves them together (at any location) is performed by the projection heuristics.

7.3.2 Factory Domain

In a notable contrast to Wall-e, in the factory domain the decomposition heuristic performs worse than any other algorithm except PRISM. The only exception is the

hardest two problems completed, 7-6 and 8-4, where the NBA heuristics timed out, but the decomposition heuristic was still outperformed by the trivial heuristic. On the smaller problems, the decomposition heuristic took approximately twice as long as the trivial heuristic for $n < 6$. Notably, the decomposition heuristic prunes more states that the other algorithms only for $n \geq 7$. The proportion of the search space pruned by the decomposition heuristic seems to be positively associated with n and *negatively* associated with k.

Similarly to the Wall-e domain, it is informative to consider the behaviour of the trivial heuristic. The projection heuristics would aim to have part p_n produced, and each machine off. The projection for p_n can always produce p_n immediately, as the preconditions for producing it are disjoint from the projection. The projection for the status of each machine would immediately turn off the machine. Because of this, the heuristic value of each state is simply how many machines are on plus 1 if p_n is yet to be produced. If a machine is turned off in such a way that it passes the imaginary baton when it is still needed to produce a part, the result is a dead end that the trivial heuristic will not detect immediately.

With the decomposition heuristic, practically no further guidance is provided. The batches that the second constraint are projected onto are those for the status of m_1 and one other machine, with one batch for each machine. Each one simply requires that they are turned off and then on in a specific order, but tying constraints relax the order of actions between heuristics. The heuristics do still require that the machines are turned off, but the projection heuristics already do this. In the case of dead ends, it is not clear whether the decomposition heuristic recognises them, and as the parts are not included in the projections, it seems likely that the dead ends are not recognised by the decomposition heuristic either.

From how little information there is to be gained from LTL heuristics for the factory domain, it seems likely that the NBA heuristic is just as uninformative on this problem as the decomposition heuristic. This is reinforced by the evidence that using the NBA heuristic takes consistently slightly longer than the trivial heuristic. From the way that using the decomposition heuristic made the solution time twice as slow compared to any other algorithm, it seems that it is computationally much more expensive than the NBA heuristic for reasonably sized NBAs.

7.3.3 Priority Search Domain

The priority search domain results show several interesting trends, the first reflecting the observation of Baumgartner et al. [2018] that sometimes using the NBA mode results in more product states than the progression mode. The NBA heuristics both took much longer and expanded many more states than the trivial heuristic with the progression mode for these problems. An analysis of one priority search problem with $n = 4, p = 0.25$ found that with the NBA mode, there were 2397 unique product states reachable from the initial state, where with the progression mode, there were 1739 unique states, only about 75% of the quantity.

The priority search domain is, in essence, just the travelling salesman problem

(TSP) with probabilistic actions and the danger zone constraints (some locations have priority and the agent can't stay in it for more than 2 steps). Existing algorithms for the TSP have solved instances with a number of goals in the tens of thousands. What makes this problem hard is the inclusion of probabilistic actions, that movement between the goals is not abstracted out of the problem (the TSP is defined on a graph) and that this is being solved by a general planning algorithm, rather than a specialised algorithm. To make this even more difficult, the branching factor for this formulation is much higher, as there are three forms of movement.

Along with the NBA heuristics, the decomposition heuristic with default batches also performed very poorly, consistently taking more than twice as long as the trivial heuristic and expanding only 2% fewer states on average. Clearly the decomposition heuristic does not provide any significant information, and this is primarily because of the way projections and tying constraints work. Consider only the constraint that all unknown locations must be searched. If this constraint is projected onto a batch $\{l_{i,j}\texttt{-unknown}\}$ for some $l_{i,j}$ it will be $\mathsf{F}\neg\langle l_{i,j}\texttt{-unknown}, \top\rangle$, and SAS$^+$ projection will have two states, one where it is unknown and the other where it is not unknown, reachable by a single action: search in $l_{i,j}$. Because of the tying constraints, the batches in combination will force the projection heuristics to take these actions. Conceptually, this should in turn force the projection onto the position of the robot to include actions which move to these locations, however in practice, in this projection searching a location does not change the state, so the LP finds a solution where the search actions are performed as isolated subtours. If however, the position and the unknown status of a given location are in the same projection, searching that location could not be an isolated subtour.

The variant decomposition heuristic + position had batches chosen so that the position was included in each batch for the second constraint of the priority search domain. The results for this variant show a significant improvement in both completion time and states expanded. The increase in informativeness was expected, but adding the position to each batch means that the constraints representing the state space for the position variable are duplicated for each unknown location. Theoretically, the increase in the number of constraints is linear in the number of unknown locations, where the total number of states is exponential in the number of unknown locations. Because of this, it is reasonable that an improvement in informativeness will pay off the linear increase in the number of variables, and more so for larger problems or problems with more unknown locations.

This is the trend that can be observed in the data, the variant of the decomposition heuristic performs worse than no heuristic when $p = 0.25$, and for $n = 4, p = 25$ it even does worse than the decomposition heuristic with default batches. However, for problems where $p \geq 0.50$, this variant provides an improvement over the trivial heuristic, and even completed some problems that no other algorithm could complete.

There is an anomaly in the data where the time the NBA heuristic (100) variant took to complete problems of size $n = 4, p = 0.50$ was twice as fast as the easier problems $n = 4, p = 0.25$, while every other algorithm performed significantly worse than on the easier problems. This is because the increase in probability increased the

size of the NBAs to 512, 258 and 4, respectively, meaning that the variant only used a heuristic for the last constraint. For $p = 0.25$, the NBAs had size 64, 34 and 4, respectively. From this massive decrease in computation time from removing instances of h^{BA}, it is clear that the constraints h^{BA} add make the LP much harder to solve and don't provide sufficient information in this domain to make up for it.

7.3.4 Conclusions

From the experimental results, it seems that $h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ is computationally expensive, but also quite informative. There is significant evidence that the effectiveness of it is sensitive to the choice of batches for projection, and issues with projections making the relaxed problem too trivial was a common theme, both in the factory and the priority search domains.

As is common for heuristics, the effectiveness of the heuristic was only observed for larger problem instances, and on smaller problem instances the heuristic was a significant hindrance. This can be overlooked, as a 6 times slower solution time on a problem that only takes one second is not a particular worry.

7.4 Summary

In this chapter, experimental evaluation of the decomposition heuristic was presented. The novel heuristic $h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ was compared with the previous state of the art and other competitive algorithms on three different domains. Two domains had been used for benchmarking MO-PLTL SSP problem solvers previously (Wall-e and factory), and the results for existing algorithms were replicated. One new domain (priority search) was introduced which reinforced the earlier findings.

The novel heuristic performed exceptionally well the Wall-e domain, while the others showed mixed results. Creating a variant with a forced choice of projection variables caused a significant improvement on the priority search domain, demonstrating that $h^{\mathrm{sdt}}_{\mathcal{S},\psi}$ is very sensitive to the choice of projection variables.

In the final chapter, a summary of the thesis is presented, along with directions of research that might deal with the limitations presented in this chapter and others, as well as other directions that show promise for more research in the field.

Chapter 8

Conclusion

Finding safe optimal policies under uncertainty is a difficult task with many applications. The problem is represented as a Stochastic Shortest Path problem subject to multiple constraints specified in probabilistic linear temporal logic, the combination of which is referred to as a MO-PLTL SSP problem. Solving an MO-PLTL SSP problem requires the synchronised product of the SSP with a state-based representation for the PLTL constraints, referred to as a mode.

Modes for LTL feature a number of states which is double exponential in the length of the LTL formula, and the state space of SSPs is also exponential in the size of its factored representation. As such, the number of product states is too large to feasibly compute in most cases.

Existing techniques for MO-PLTL SSP problems primarily construct the full product C-SSP, and then perform a reachability analysis on it. A more recent algorithm called PLTL-dual applies heuristic search, which is a technique that uses heuristic estimates to construct a policy without enumerating the complete state space.

There exists only one non-trivial heuristic for PLTL introduced with PLTL-dual. This heuristic relies on the non-deterministic Büchi automata mode, and empirical evaluation shows that can be less efficient than the progression mode with the trivial heuristic. The existing heuristic involving the NBA mode also proved quite inefficient when the NBA for PLTL constraints became too large.

This thesis addresses these issues by introducing a new heuristic for MO-PLTL SSP problems, aimed for integration with PLTL-dual and relying on the progression mode. This heuristic combines two relaxations, projection and the novel concept of formula decomposition.

Projection is commonly used for heuristics, but a projection cannot be subject to a PLTL constraint which references variables removed by projection. Projection was extended to PLTL constraints in such a way that only projection variables remain in the formula, but the resulting formula describes some of the same paths as the original formula.

Decomposition is a relaxation in which the satisfaction of an LTL formula is reduced to the satisfaction of the subformulae in appropriate combinations. In order to represent this decomposition being repeated at each step, a new planning problem called a Concurrent Constrained SSP was introduced. It features multiple agents being cloned into states associated with subformulae of the formula associated with their current

state.

The solution to this problem is computed using a linear program, which represents the movement of these agents as a flow network. This approach suffered from significant limitations, including spontaneous generation of flow, which allowed most of a CC-SSP to be skipped. Furthermore, this approach could not be integrated directly into PLTL-dual.

To remedy this, further constraints were added to the LP for CC-SSPs to encode the requirement that flow should find its way from the source to the goals. These constraints were based on the concept of an accepting flow trace, i.e., that flow which reached goals should be traced back to the source in full. This removes most issues with spontaneous flow generation, and also allows this approach to be integrated into PLTL-dual.

The split decomposition trace heuristic $h^{\text{sdt}}_{\mathcal{S},\psi}$ was introduced, which constructs several projections, and performs decomposition with an accepting flow trace on each. This heuristic was empirically evaluated against the other algorithms and variants for MO-PLTL SSP problems, showing that it made significant improvement over the other approaches. The empirical evaluation also showed several areas for improvement.

The split decomposition trace heuristic was shown to be very sensitive to the choice of variables on which to project, and appears to have a larger overhead on smaller problems than the NBA heuristic.

8.1 Future Work

The future work here is separated into two groups: improvements or analysis of the heuristics discussed in this thesis, and future work in related topics.

8.1.1 Work in this Thesis

There are several areas of potential improvement for the work in this thesis, but mostly the suggestions here are for further analysis of the work presented.

Batch Choice Algorithms As was made very evident in the experiments, particularly on the priority search domain, the efficacy of $h^{\text{sdt}}_{\mathcal{S},\psi}$ is very sensitive to the choice of batches onto which projections are made. It would be improved very significantly if a more effective batch choice algorithm were studied. This algorithm could also be used for the projection heuristics in PLTL-dual, replacing the use of single variables with intelligently chosen batches of variables.

Disjunctive Normal Form The choice of conjunctive normal form for formulae in this thesis was inspired partially by the discussion of CNF with a 'Tseitin style' transformation in Baumgartner et al. [2018], and primarily because separating the clauses in CNF lends itself to a probability upper bound. The separation of X-literals within clauses was a later addition once CNF was already practically locked in. Roşu and Havelund [2005] state that (at least for runtime verification) disjunctive normal

form (DNF) is typically used as a canonical representation of LTL formula, as opposed to CNF. The methods in chapters 5 and 6 could relatively easily be adapted for a DNF representation.

Further analysis of $h^{\mathbf{sdt}}_{\mathcal{S},\psi}$ Decomposition has been tested without use of projection, with the expected result that the state space is still so complex that computing the heuristic is prohibitively slow for large problems. However, projection of PLTL and SSPs has not been tested independently of decomposition. It is not clear without performing tests on projection alone how much of the improvements seen from $h^{\text{sdt}}_{\mathcal{S},\psi}$ can be attributed to decomposition.

Accepting Flow Trace Improvements The number of variables introduced by including an accepting flow trace is several times more than the number of occupation measures in the flow problem without the trace. This greatly increases the overhead for $h^{\text{sdt}}_{\mathcal{S},\psi}$, and this overhead was observed in some experiments in the factory domain. It may be possible that some variables can be eliminated from the representation, which would greatly improve the solution time for the LP with the accepting flow trace. It may also be possible to make the accepting flow trace more informative by making all flow into a state equally responsible for the accepting flow leaving it.

8.1.2 Future Related Work

The construction and analysis of $h^{\text{sdt}}_{\mathcal{S},\psi}$ has revealed some areas for further research, especially relating to PLTL-dual.

NBA Mode vs Progression Mode The experiments on the new priority search domain revealed an extreme case where the NBA mode generated significantly more states than the progression mode. Investigating exactly what cases this occurs in (and why) may lead to improvements in the NBA mode and other representations for LTL.

Other Heuristics The success of $h^{\text{sdt}}_{\mathcal{S},\psi}$ reveals that there are definitely opportunities for effective heuristics for PLTL constraints. Discovering better heuristics for MO-PLTL SSP problems may lead to improvements not only in planning but also in strategy synthesis for LTL.

Split Automata Camacho et al. [2018] make use of technique which splits large automata up into smaller ones, resulting in much more manageable state spaces. It may be possible to apply this optimisation in PLTL-dual, and it may also be possible to extend the same idea to the progression mode. Doing so would likely make the NBA mode much more competitive, given that it currently performs much slower than the progression mode.

Appendix A

Further Results

The following is the data from the experiments listed in chapter 7. In that chapter, only aggregate results were provided in the form of graphs and a table of averages and coverage, and all showed only the solution time. The number of states is also relevant, so it is provided here, along with the times for the Wall-e and factory domains. Instances where the algorithm was cut off by the 20 minute time limit are marked 't/o' for time-out. The number of states expanded by PRISM is omitted, as PRISM does not output the complete number of states in the problem, only the states in each DRA and the MDP.

Problem Size	Trivial heuristic (s)	NBA heuristic (s)	NBA heuristic (100) (s)	Decomposition heuristic (s)	PRISM (s)
$n = 3$	1.30	2.09	2.00	1.02	0.49
$n = 4$	4.36	5.58	6.59	1.73	4.14
$n = 5$	30.31	43.22	42.25	5.00	41.32
$n = 6$	115.42	217.13	148.38	9.83	t/o
$n = 7$	494.52	711.36	377.36	17.67	t/o
$n = 8$	t/o	t/o	t/o	28.39	t/o
$n = 9$	t/o	t/o	t/o	49.50	t/o
$n = 10$	t/o	t/o	t/o	93.50	t/o
$n = 11$	t/o	t/o	t/o	145.98	t/o
$n = 12$	t/o	t/o	t/o	220.59	t/o
$n = 13$	t/o	t/o	t/o	360.45	t/o
$n = 14$	t/o	t/o	t/o	582.22	t/o
$n = 15$	t/o	t/o	t/o	785.97	t/o

Table A.1: Time in seconds to complete each Wall-e instance.

Problem Size	Trivial heuristic	NBA heuristic	NBA heuristic (100)	Decomposition heuristic
$n = 3$	651	682	669	383
$n = 4$	1382	1362	1420	555
$n = 5$	3169	3117	3102	911
$n = 6$	5620	5846	5829	1172
$n = 7$	9700	9183	9144	1543
$n = 8$	t/o	t/o	t/o	1886
$n = 9$	t/o	t/o	t/o	2284
$n = 10$	t/o	t/o	t/o	2956
$n = 11$	t/o	t/o	t/o	3389
$n = 12$	t/o	t/o	t/o	3897
$n = 13$	t/o	t/o	t/o	4564
$n = 14$	t/o	t/o	t/o	5330
$n = 15$	t/o	t/o	t/o	5889

Table A.2: States expanded for each problem in the Wall-e domain.

	Trivial heuristic	NBA heuristic	NBA heuristic (100)	Decomposition heuristic	Decomposition heuristic + position
$n = 4, p = 0.25$					
coverage	10	8	9	10	10
average	560.40	771.63	741.67	546.60	377.90
0.95 CI	69.41	108.83	104.33	64.88	78.42
$n = 4, p = 0.50$					
coverage	10	0	10	10	10
average	3542.80	N/A	4836.70	3484.60	1887.40
0.95 CI	951.32	N/A	1300.37	914.81	848.06
$n = 4, p = 0.75$					
coverage	0	0	0	0	6
average	N/A	N/A	N/A	N/A	7427.33
0.95 CI	N/A	N/A	N/A	N/A	2390.53
$n = 5, p = 0.25$					
coverage	10	0	10	10	10
average	2327.00	N/A	3038.30	2278.60	1375.90
0.95 CI	651.13	N/A	856.55	613.71	396.24
$n = 5, p = 0.50$					
coverage	0	0	0	0	3
average	N/A	N/A	N/A	N/A	3794.67
0.95 CI	N/A	N/A	N/A	N/A	1529.22

Table A.3: Aggregated number of states to complete each parametrisation of the priority search domain.

Problem Size	Trivial heuristic (s)	NBA heuristic (s)	NBA heuristic (100) (s)	Decomposition heuristic (s)	PRISM (s)
2-0	0.08	0.16	0.13	0.08	0.06
2-1	0.11	0.11	0.13	0.09	0.06
3-0	0.11	0.14	0.16	0.16	0.15
3-1	0.13	0.13	0.16	0.16	0.15
3-2	0.13	0.16	0.14	0.17	0.15
4-0	0.23	0.19	0.20	0.42	52.83
4-1	0.22	0.23	0.25	0.47	53.44
4-2	0.23	0.30	0.28	0.58	52.71
4-3	0.27	0.30	0.33	0.55	53.99
5-0	0.64	0.63	0.63	1.72	t/o
5-1	0.67	0.80	0.70	1.77	t/o
5-2	0.78	0.86	0.91	1.98	t/o
5-3	1.00	1.09	1.16	2.67	t/o
5-4	1.72	1.83	1.95	3.44	t/o
6-0	3.22	3.20	3.48	7.56	t/o
6-1	3.70	4.30	4.13	8.45	t/o
6-2	4.31	4.48	5.33	10.03	t/o
6-3	5.50	7.09	7.53	12.84	t/o
6-4	11.30	13.36	12.86	19.75	t/o
6-5	26.92	34.69	34.81	38.84	t/o
7-0	28.75	30.84	30.30	40.34	t/o
7-1	25.89	35.59	29.81	47.53	t/o
7-2	41.48	46.23	41.48	59.55	t/o
7-3	61.77	66.34	59.00	78.03	t/o
7-4	118.77	132.86	137.11	143.11	t/o
7-5	278.42	368.64	351.44	419.48	t/o
7-6	904.98	t/o	t/o	1006.23	t/o
8-0	318.25	340.30	304.50	269.78	t/o
8-1	324.61	374.66	359.11	347.05	t/o
8-2	463.47	516.33	613.11	461.38	t/o
8-3	564.56	872.11	608.28	649.63	t/o
8-4	1131.81	t/o	t/o	1190.31	t/o

Table A.4: Time in seconds to complete each factory instance.

Problem Size	Trivial heuristic	NBA heuristic	NBA heuristic (100)	Decomposition heuristic
2-0	26	22	22	19
2-1	24	22	22	24
3-0	68	59	59	58
3-1	78	67	67	70
3-2	78	78	82	80
4-0	191	161	161	167
4-1	180	178	201	205
4-2	220	242	242	240
4-3	255	251	251	264
5-0	456	442	454	436
5-1	521	503	481	488
5-2	568	551	558	571
5-3	623	600	599	686
5-4	764	765	796	794
6-0	1058	1068	1074	995
6-1	1069	1201	1186	1122
6-2	1220	1227	1342	1253
6-3	1389	1438	1456	1412
6-4	1813	1835	1843	1826
6-5	2733	2701	2636	2610
7-0	2901	2596	2538	2252
7-1	2649	2782	2659	2550
7-2	3224	3174	2963	2760
7-3	3794	3512	3388	3052
7-4	4376	4336	4271	3897
7-5	6328	5983	5859	5864
7-6	9568	t/o	t/o	8414
8-0	6726	6271	6758	4889
8-1	6604	6409	7092	5712
8-2	8727	7153	8291	6207
8-3	9013	8163	8755	6925
8-4	11023	t/o	t/o	8502

Table A.5: States expanded for each problem in the factory domain.

Bibliography

Babiak, T.; Křetínský, M.; Řehák, V.; and Strejček, J., 2012. LTL to büchi automata translation: Fast and more deterministic. In *Tools and Algorithms for the Construction and Analysis of Systems*, 95–109. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-28756-5_8. (cited on page 85)

Bacchus, F. and Kabanza, F., 1996. New directions in ai planning. chap. Using Temporal Logic to Control Search in a Forward Chaining Planner, 141–153. IOS Press, Amsterdam, The Netherlands, The Netherlands. ISBN 90-5199-237-8. (cited on page 31)

Bacchus, F. and Kabanza, F., 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22, 1 (Feb 1998), 5–27. doi:10.1023/A:1018985923441. (cited on pages 10, 14, 15, 32, and 41)

Baier, C. and Katoen, J.-P., 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press. ISBN 026202649X, 9780262026499. (cited on pages 8 and 35)

Baier, J. A.; Bacchus, F.; and McIlraith, S. A., 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173, 5 (2009), 593 – 618. doi:10.1016/j.artint.2008.11.011. Advances in Automated Plan Generation. (cited on page 33)

Baier, J. A. and Mcilraith, S. A., 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06 (Boston, Massachusetts, 2006), 788–795. AAAI Press. (cited on pages 11, 32, and 33)

Bauer, A. and Haslum, P., 2010. LTL goal specifications revisited. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, 881–886. IOS Press, Amsterdam, The Netherlands, The Netherlands. (cited on page 11)

Bauer, A.; Leucker, M.; and Schallhart, C., 2010. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 20, 3 (2010), 651–674. (cited on page 11)

Baumgartner, P.; Thiébaux, S.; and Trevizan, F., 2018. Heuristic search planning with multi-objective probabilistic LTL constraints. In *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning.* (cited on pages 3, 6, 17, 27, 36, 38, 85, 86, 89, 91, 92, and 96)

Bäckström, C. and Nebel, B., 1995. Complexity results for SAS+ planning. *Computational Intelligence*, 11 (1995), 625–655. (cited on page 22)

Bertsekas, D. P. and Tsitsiklis, J. N., 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16, 3 (1991), 580–595. doi:10.1287/moor.16.3.580. (cited on page 2)

Bonet, B. and Geffner, H., 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proceedings of the Thirteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'03 (Trento, Italy, 2003), 12–21. AAAI Press. (cited on page 23)

Camacho, A.; Baier, J. A.; Muise, C.; and McIlraith, S. A., 2018. Finite ltl synthesis as planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*. (cited on pages 34 and 97)

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A., 2017. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17 (San Francisco, California, USA, 2017), 3716–3724. AAAI Press. (cited on page 33)

Culberson, J. C. and Schaeffer, J., 1998. Pattern databases. *Computational Intelligence*, 14, 3 (1998), 318–334. doi:10.1111/0824-7935.00065. (cited on page 26)

De Giacomo, G. and Vardi, M. Y., 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13 (Beijing, China, 2013), 854–860. AAAI Press. (cited on page 11)

De Giacomo, G. and Vardi, M. Y., 2015. Synthesis for LTL and LDL on finite traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15 (Buenos Aires, Argentina, 2015), 1558–1564. AAAI Press. (cited on pages 34 and 35)

Desrochers, M. and Laporte, G., 1991. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Oper. Res. Lett.*, 10, 1 (Feb. 1991), 27–36. doi:10.1016/0167-6377(91)90083-2. (cited on page 69)

Doherty, P. and Kvarnstram, J., 2001. Talplanner: A temporal logic-based planner. *AI Magazine*, 22, 3 (Sep. 2001), 95. doi:10.1609/aimag.v22i3.1581. (cited on page 31)

Duret-Lutz, A., 2016. Spot's temporal logic formulas. Technical report, Tech. rep. Available online: https://spot. lrde. epita. fr/tl. pdf. (cited on page 8)

Edelkamp, S., 2006. On the compilation of plan constraints and preferences. In *Proceedings of the Sixteenth International Conference on International Conference*

on Automated Planning and Scheduling, ICAPS'06 (Cumbria, UK, 2006), 374–377. AAAI Press. (cited on pages 32 and 33)

Edelkamp, S., 2007. Automated creation of pattern database search heuristics. In *Model Checking and Artificial Intelligence*, 35–50. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-540-74128-2_3. (cited on page 44)

Gurobi Optimization, L., 2019. Gurobi optimizer reference manual. http://www.gurobi.com. (cited on pages 20 and 85)

Hansen, E. A. and Zilberstein, S., 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129, 1 (2001), 35 – 62. doi:10.1016/S0004-3702(01)00106-0. (cited on page 23)

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S., 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, 1007–1012. http://www.aaai.org/Library/AAAI/2007/aaai07-160.php. (cited on pages 4 and 44)

Helmert, M., 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26 (2006), 191–246. doi:10.1613/jair.1705. (cited on page 85)

Helmert, M.; Haslum, P.; and Hoffmann, J., 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07 (Providence, Rhode Island, USA, 2007), 176–183. AAAI Press. (cited on pages 4 and 44)

Kerjean, S.; Kabanza, F.; St-Denis, R.; and Thiébaux, S., 2006. Analyzing LTL model checking techniques for plan synthesis and controller synthesis (work in progress). *Electronic Notes in Theoretical Computer Science*, 149, 2 (2006), 91–104. (cited on page 3)

Klauck, M.; Steinmetz, M.; Hoffmann, J.; and Hermanns, H., 2018. Compiling probabilistic model checking into probabilistic planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling.*

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H., 2011. Heuristic search for Generalized Stochastic Shortest path MDPs. In *Proceedings of the Twenty-First International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'11 (Freiburg, Germany, 2011), 130–137. AAAI Press. (cited on page 19)

Kwiatkowska, M.; Norman, G.; and Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, vol. 6806 of *LNCS*, 585–591. Springer. (cited on pages 35, 85, and 86)

Kwiatkowska, M. and Parker, D., 2013. Automated verification and strategy synthesis for probabilistic systems. In *Automated Technology for Verification and Analysis*, 5–22. Springer International Publishing, Cham. doi:10.1007/978-3-319-02444-8_2. (cited on pages 3, 27, and 35)

Mausam and Kolobov, A., 2012. Planning with Markov Decision Processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6, 1 (2012), 1–210. doi:10.2200/S00426ED1V01Y201206AIM017. (cited on page 17)

Pnueli, A., 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. doi:10.1109/SFCS.1977.32. (cited on pages 2 and 7)

Rabin, M. O., 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 (1969), 1–35. doi:10.2307/1995086. http://www.jstor.org/stable/1995086. (cited on page 35)

Rintanen, J., 2003. Expressive equivalence of formalisms for planning with sensing. In *ICAPS*, 185–194. (cited on page 81)

Roşu, G. and Havelund, K., 2005. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12, 2 (Apr 2005), 151–197. doi:10.1007/s10515-005-6205-y. (cited on pages 17 and 96)

Sanner, S., 2010. Relational Dynamic Influence Diagram Language (RDDL): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf. (cited on page 22)

Steinmetz, M.; Hoffmann, J.; and Buffet, O., 2016. Revisiting goal probability analysis in probabilistic planning. In *ICAPS*. (cited on page 19)

Trevizan, F.; Thiébaux, S.; and Haslum, P., 2017a. Occupation measure heuristics for probabilistic planning. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*. (cited on pages 4, 22, 25, and 37)

Trevizan, F. W.; Teichteil-Königsbuch, F.; and Thiébaux, S., 2017b. Efficient solutions for Stochastic Shortest Path problems with dead ends. In *UAI*. (cited on pages 19 and 61)

Trevizan, F. W.; Thiébaux, S.; de Rodrigues Quemel e Assis Santana, P. H.; and Williams, B. C., 2016. Heuristic search in dual space for constrained stochastic shortest path problems. In *ICAPS*. (cited on pages 6, 85, and 87)

Younes, H. L. and Littman, M. L., 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2 (2004), 99. (cited on page 22)