DIRECTED UNFOLDING

Reachability Analysis of Concurrent Systems
& Applications to Automated Planning

by

Sarah Louise Hickmott

Thesis submitted for the degree of

Doctor of Philosophy

in

School of Electrical and Electronic Engineering
Faculty of Engineering
Computer and Mathematical Sciences

University of Adelaide, Australia

2008

This page left blank.

Abstract

The factored state representation and concurrency semantics of Petri nets are closely related to those of classical planning models, yet automated planning and Petri net analysis have developed independently, with minimal and mainly unconvincing attempts at cross-fertilisation. This thesis exploits the relationship between the formal reachability problem, and the automated planning problem, via Petri net unfolding, which is an attractive reachability analysis method for highly concurrent systems as it facilitates reasoning about independent sub-problems. The first contribution of this thesis is the theory of *directed unfolding*: controlling the unfolding process with informative strategies, for the purpose of optimality and increased efficiency. The second contribution is the application of directed unfolding to automated planning.

Inspired by well-known planning heuristics, this thesis shows how problem specific information can be employed to guide unfolding, in response to the formal problem of developing efficient, directed reachability analysis methods for concurrent systems. Complimenting this theoretical work, this thesis presents a new forward search method for partial order planning which can be exponentially more efficient than state space search.

Our suite of planners based on directed unfolding can perform optimal and suboptimal classical planning subject to arbitrary action costs, optimal temporal planning with respect to arbitrary action durations, and address probabilistic planning via replanning for the most likely path. Empirical results reveal directed unfolding is competitive with current state of the art automated planning systems, and can solve Petri net reachability problems beyond the reach of the original "blind" unfolding technique.

This page left blank.

For my parents.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

Acknowledgements

I would not have been able to complete this thesis without support from many people and in particular, supervision from Langford White, Ian Fuss and Sylvie Thiébaux. Lang provided me with an endless stream of ideas, but equally importantly, supported me to take my own direction with this project, and consistently conveyed his belief in me and this work. Ian challenged my ideas and worked to broaden my perspective and assist me in seeing a bigger picture. Sylvie provided invaluable guidance and encouragement; her energy and determination both assisted and inspired me. Without the knowledge, experience and generous support of each of these people I would not have been able to complete this work; my gratitude for each person's contribution is heartfelt.

Throughout this thesis there are significant contributions from Blai Bonet, Patrik Haslum and Jussi Rintanen, my co-authors in conjunction with Sylvie and Lang, on publications relating to this work. I am grateful for and appreciative of the insight, knowledge and enthusiasm Blai, Patrik and Jussi brought to this research. Patrik also read and provided useful feedback on this thesis for which I am thankful.

Over the last four years I have had the opportunity to discuss my work with many scientists, but am particularly grateful to Alban Grastien, Jinbo Huang, Subbarao Kambhampati, Sophie Pinchinat and David Smith for fruitful discussions; I also thank Stefan Schwoon for his assistance with MOLE.

This work has been financially supported by the University of Adelaide (UofA) Faculty of Engineering, Mathematical and Computer Sciences and the National ICT Australia (NICTA) via the Dynamic Planning, Optimisation and Learning Project (DPOLP). I would like to thank all members of DPOLP, for interesting discussions and useful responses to this work, in particular Jonathon Billington and Guy Gallasch for their technical advice, and Sanjeev Naguleswaran for valuable guidance. I thank Bob Williamson for helping to facilitate my extended visits to Canberra. I also wish to express my gratitude to Owen Thomas and Olivier Buffet, for programming support beyond the call of duty.

Finally, I take this opportunity to thank my friends, who have strongly supported and patiently endured me, throughout my increasing absorption in this project. And last but definitely not least I want to thank my family, who always stand by me, and have shared the difficulty as well as the excitement of this journey.

This page left blank.

Contents

This page left blank.

Chapter 1

Introduction

Consider a situation described by the values of a finite set of variables. There are various relationships between these variables, captured by a set of rules governing how they can deterministically transition from one discrete value to another. There are only finite possible values for each variable. Now change the focus: consider the same situation from the perspective that there is a set of possible actions. When one is executed, it may enable the occurrence of other actions not previously viable; it may also disable an action that was previously possible. The causal dependency between actions is governed by their interaction with particular variables. Overall, the repertoire of actions is finite.

Observe that the first perspective focuses on the sensed world (states); the second focuses on the world of process (actions). A useful question one may ask of this situation is "*What* is possible?" Or, more specifically, "Is *this* possible, and *how*?", where *this* is a particular assignment of values to variables, and *how* requests a partially ordered set of actions that will transform the situation to meet these values. Let us refer to the later question, as the *reachability problem*, and the dynamic situation described as a restricted *discrete event-driven system* (DES). A simple approach to the reachability problem is to take the state based perspective and systematically search through all possible value-variable assignments by applying the rules governing their interaction. Or, adopting the action based perspective, one could identify which actions interact with the variables of interest and attempt to coordinate them. This thesis builds on a unique method of analysis that utilises both state and action based perspectives to explicitly reveal the correlations between variables in the sensed world and actions in the world of process: *Petri net unfolding* [59, 119, 53, 125].

Whilst the detail captured in a restricted DES is limited, there is sufficient information to answer reachability questions, i.e. the problem is decidable. But even with such limited expression, the reachability problem quickly becomes intractable. This is because the num-

ber of states grows exponentially in the number of variables used to describe the system: the *state explosion problem*. This is obviously serious even for the case where variables are logical propositions, which can only be true or false. If analysing a *concurrent* system, consisting of largely independent processes, the problem worsens. Considering the global combination of each process's variables (i.e. the global state) makes the total number of states the product of the number of local states for each process. Interleaving independent state transitions (i.e. concurrent actions) further aggravates the problem, as the number of transitions between states will increase exponentially with the number of concurrent actions. Petri net unfolding addresses each of these "fictional" components of complexity via its factored state representation and concurrency semantics, which facilitate reasoning about the local state of a process and avoid the arbitrary interleaving of concurrent actions [119]. Note however that the cost incurred by such reasoning means it is only lucrative for highly concurrent systems.

This research is primarily interested by the reachability problem in the context of planning. Planning, generally speaking, involves identifying how one can actively influence a situation to achieve a particular result. In particular, this thesis addresses the *classical planning problem*; classical planning considers situations which can be represented as restricted DES [67]. Automated planning, a key component of Artificial Intelligence (AI), is the act of formalising the planning process computationally. Based on Petri net unfolding, this thesis presents a new approach to automated classical planning that is suitable for situations with many concurrent actions.

Petri nets are traditionally used for modelling and analysing distributed systems [126]. Thus it is not surprising that their factored state representation and concurrency semantics are similar to those of classical planning domains. However planning and Petri net analysis have developed independently, with minimal attempts at cross-fertilisation. Unfolding is a Petri net analysis technique which reveals all partially ordered runs of the net. In this way, contingent upon an appropriate mapping between a planning problem and Petri net reachability problem, unfolding the net is equivalent to a forward search through the space of partially ordered plans.

The reachability problem also arises in formal system verification, via model checking [40], and model based system diagnosis [149], e.g. [140]. In each of these application areas, but especially planning, there is benefit in identifying the cheapest or fastest way to achieve a particular value-variable assignment. But until now Petri net unfolding has not facilitated finding optimal solutions to the reachability problem, with respect to various cost functions. Furthermore, Petri net unfolding is not actually oriented to solving the reachability problem

as it was developed for deadlock detection, which requires exhaustive search. In the face of state explosion, and the context of reachability analysis, controlling the search for a solution and pruning the search space can be the critical difference in a problem being solvable within available time and memory constraints. *Heuristics*, which estimate the potential of different search decisions, are central to search control [132, 147]. In the last decade, AI planning research has revealed state based heuristics, extracted directly from a problem description, are particularly effective for guiding search and pruning the search space. Indeed, the use of state based heuristics has significantly increased the scalability of state based planning algorithms [18, 86, 116]. This has inspired the integration of heuristics in model checking algorithms, e.g [50, 47, 109, 143, 51]. Unfortunately however the state based perspective does not offer the flexibility of an action based perspective; in particular it generally encourages a total ordering of actions. The flexibility of action based approaches, which generally facilitate a partial ordering of actions based on causality, comes at a cost which is worsened by the relative inefficiency of non state based heuristics. This thesis shows how a compromise can be made via Petri net unfolding.

When applied to solve the reachability problem, Petri net unfolding traditionally performs a breadth-first search, which can only guarantee optimality with respect to the number of actions in a solution, and only manages to prune parts of the search space repeated elsewhere. This thesis presents the theory of *directed unfolding*: controlling the unfolding process with informative strategies, for the purpose of optimality and increased efficiency. In particular, it employs state based heuristic functions. So, directed unfolding offers the flexibility of an action based perspective, but at the same time utilises state based heuristic functions for guidance and pruning.

Inspired by well-known planning heuristics, this thesis shows how problem specific information can be employed to guide unfolding, in response to the formal problem of developing efficient, directed reachability analysis tools for concurrent DES. Complimenting this theoretical work, this thesis also presents a new forward search method for partial order planning which can be exponentially more efficient than state space search.

1.1 Initial Motivation

Before outlining the contributions of this thesis, let us summarise the reasons why we endeavoured to build a bridge between Petri nets and AI planning, via unfolding, and thus give due credit to the work that originally inspired the direction of this thesis.

This thesis contributes to the Dynamic Planning, Optimisation and Learning Project (DP-

OLP). DPOLP is an initiative of Ian Fuss from the Australian Defence Science and Technology Organisation (DSTO), Langford White from the University of Adelaide (UofA), and Bob Williamson from the National Information Communications Technology Australia (NICTA). Fuss, White, and Williamson sought to enhance the theory and practice of automated planning, with particular focus on military operations planning, by fostering collaboration between researchers from machine learning (ML), AI planning, and systems engineering. They were helped in setting up DPOLP's initial research agenda by other researchers in these disciplines, most notably Douglas Aberdeen (NICTA), Lars Kristensen (University of Aarhus, Denmark), Sylvie Thiébaux (NICTA), and Lin Zhang (DSTO).

This thesis began with a desire to develop and exploit connections between systems engineering and AI planning. The connection between Petri nets and planning was originally communicated to us by Lin Zhang, who was coordinating development of the Course of Action Scheduling Tool (COAST) [175]. This is a scheduling tool designed specifically to support operations planning in the Australian Defence Force. At the core of COAST is a *coloured* Petri net model, which captures and exposes the dynamics of a planning domain in an accurate and pragmatic manner. Unfortunately however, empirical results revealed that the state based approach employed by COAST does not scale well due to the state explosion problem. In response, some DPOLP members set out to explore the trade off with using slightly less expressive *predicate* Petri nets. Conversely, we decided to begin at the foundation of both AI planning and Petri nets: classical planning and *place transition* Petri nets.

In the investigation which followed Langford White was captured by Benveniste, Fabre and Haar's attempts to utilise the concurrency semantics of Petri net unfolding to the reduce the complexity of analysing Markov Decision Processes with concurrent actions [10]. This led us to McMillan [119] and Esparza, Römer and Vogler's [56] specific presentations of the unfolding algorithm. Further review revealed research in system verification, e.g. [57, 96, 79, 78], and diagnosis, e.g. [22, 21, 62, 11], which looks to unfolding for some relief of the state explosion problem in the context of concurrent systems. We could find no attempt to use unfolding for AI planning, despite the fact McMillan accredits inspiration for the unfolding method to partial order planning techniques [117]. It was also clear that the potential of unfolding is still under exploration. Thus there was apparent opportunity for us to translate, tailor and apply the method to planning problems, and inherently improve it as a method for reachability analysis. Furthermore, it appeared that unfolding navigates a unique search space, compared to popular automated planning methods. This suggested planning via unfolding could offer a different trade-off between flexibility and efficiency, compared to current planning approaches.

This, in short, is why we began exploring the connections between planning and Petri net unfolding. What followed is the topic and contribution of this thesis.

1.2 Contribution

The primary contributions of this thesis are the theory of directed unfolding, and its application to automated planning:

(1) *The theory of directed unfolding.*
This thesis shows how the unfolding process can be directed with informative strategies, for the purpose of optimality and increased efficiency, when solving the reachability problem. This contribution includes:

- (a) Identifying that the requirements on strategies for controlling the unfolding are stronger than necessary;
- (b) Recognising and developing the potential for Petri net unfolding to solve the reachability problem optimally by:
  - Identifying conditions which ensure optimality with respect to a particular cost function; and
  - Crafting strategies that minimise *additive* and *parallel* cost functions. These can be used to model financial cost or execution time respectively, for example.
- (c) Evolving the unfolding to be a principled method for solving the reachability problem, by enabling the process to be directed with problem specific information in the form of heuristics. This includes:
  - Developing a framework for associating an arbitrary cost function with a heuristic function, such that the cost function can be optimised if the heuristic function is admissible;
  - Providing the option to prioritise efficiency or solution optimality by using non-admissible or admissible heuristics respectively;
  - Instantiating this framework for the case of additive and parallel cost functions, using results from (b); and
  - Adapting heuristic functions developed for AI planning, to the Petri net structure, and thus demonstrating that a range of suitable heuristic functions can be automatically extracted from the original Petri net.

(d) Comparing the performance of directed unfolding with the original "blind" approach by:

- Extending MOLE[1], a freeware program which unfolds place transition Petri nets, to implement the theory of directed unfolding with additive and parallel cost functions and various heuristic functions adopted from AI planning;
- Providing empirical results comparing directed unfolding with the original "blind" approach on a series of Petri net benchmarks and planning problems;
- Explaining the superior performance of directed unfolding, for both positive and negative solutions to the reachability problem.

(2) *Planning via directed unfolding.*

Based on directed unfolding, this thesis presents a new forward search method for partial order classical planning that can optimise the additive or parallel cost of a plan, and be guided by admissible or inadmissible state based heuristics. This contribution includes:

(a) Translating a classical planning world to a place transition Petri net, then casting a classical planning problem as a Petri net reachability problem;

(b) Showing how directed unfolding can then be employed to synthesise a partially ordered solution plan which is optimal or suboptimal with respect to the additive cost of actions, or optimal with respect to the parallel cost of actions. Action costs can be arbitrary positive numbers;

(c) Comparing planning via directed unfolding with other approaches to automated planning by:

- Looking at planning via unfolding from the perspective of refinement planning and thus recognising that it offers a compromise between state based and action based (commonly referred to as plan space) approaches;
- Crafting artificial problems to demonstrate the ideal situation for planning via directed unfolding, and assist in explaining why it can be exponentially more efficient than state based approaches;
- Developing the PUP SUITE, a collection of planners that implement Petri net Unfolding Planning, and REPUP, a replanner built onto the PUP SUITE that addresses probabilistic planning;

[1] `http://www.fmi.uni-stuttgart.de/szs/tools/mole/`

- Presenting empirical results which display the performance of the PUP SUITE for suboptimal and optimal classical planning, optimal temporal planning, and replanning for probabilistic action effects, using appropriate benchmarks from the International Planning Competitions; and
- Comparing results from the PUP SUITE with those of current state of the art planing systems, giving consideration to the planner runtime, plan length, additive and parallel plan costs, and plan execution flexibility.

This work has already found interest in the planning community, e.g. [83], the Petri net community, e.g. [20], the model checking community, e.g. [82] and the broader AI community, e.g. [84]. Indeed, after being introduced to the notion of planning via unfolding, various researchers from the planning community became personally involved in the development, application and implementation of the theory presented here. The conclusions of Chapter 4 and Chapter 6 detail the invaluable contributions of Sylvie Thiébaux, Patrik Haslum, Blai Bonet and Jussi Rintanen. We are also aware that some members of the Petri net community are further investigating the implications of this research, having found it challenged their previous understanding of unfolding. On a minor note, developers of MOLE have expressed interest in extending their publicly available software to utilise directed unfolding.

1.3 Overview

In accordance with its primary contributions, this thesis is presented in two parts. Part I, titled *Directed Unfolding*, presents the background to, and theory of, directed unfolding. It comprises of Chapters 2, 3 and 4. Part II, titled *Planning via Directed Unfolding*, applies the theory of directed unfolding to automated planning. This part comprises of Chapters 5 and 6.

Chapter 2 provides a "big picture" perspective of the reachability problem. We detail the motivation behind modelling a situation as a DES, explain our particular interest in concurrent systems, formally define the reachability problem for a restricted DES, and describe its relevance to automated planning, system verification and diagnosis. We then discuss the state explosion problem in detail, focusing on the "fictional" complexity that can be mandated by how we model a problem, rather than the problem itself. This leads to explanation of why Petri nets are an appropriate model for concurrent systems. We then move from the issue of modelling to analysis: the main techniques for combating the state explosion

problem are symbolic algorithms, partial order methods, abstraction techniques, symmetry reduction and decomposition methods. We compare Petri net unfolding with a range of algorithms employing these ideas. In this chapter we also consider the value of optimal and/or partially ordered solutions to the reachability problem, and look at how state based heuristics have impacted the scalability of some AI planning algorithms. We present a simple state based search and show it can be controlled for the purpose of optimisation and efficiency, using state based heuristics, but suffers greatly from the aforementioned "fictional" complexity when used to analyse highly concurrent systems. In this way we motivate the development and application of *directed unfolding*, for answering reachability questions for concurrent DES.

Chapter 3 describes the syntax and semantics of Petri net unfolding. This includes a description of place transition Petri nets (PT-nets), and the technique and data structure of PT-net unfolding. We define the reachability problem for PT-nets, outline the main approaches to PT-net reachability analysis, and state our interest in the unfolding approach. We then formalise the connection between reachability analysis and PT-net unfolding, discuss the implications in terms of complexity theory, and finally present the fundamental algorithm this thesis builds on: *on-the-fly* reachability analysis via PT-net unfolding.

Chapter 4 presents the theory of directed unfolding. This chapter integrates intuitive explanation and useful examples, with thorough theoretical results. All contributions outlined in the previous section, collectively referred to as the theory of directed unfolding, are found in this chapter. The conclusion of this chapter includes a detailed synopsis of the collaborative work, and individual contributions, which led to the development of the theory of directed unfolding.

Moving into Part II, Chapter 5 serves as an introduction to automated planning: what, why and how? This includes an outline of the theoretical and practical motivations for automated planning in general and classical planning in particular. It presents a conceptual model for a planning problem which, subject to various assumptions, is equivalent to the reachability problem discussed in Chapter 2 and, importantly, represents the classical planning problem. We formulate the classical planning problem and summarise the main approaches to analysis: state space search, plan space search, planning as satisfiability and Graphplan. The discussion here includes comment on the impact of informative heuristic functions, and the benefits of partial order versus total order planning. We then present extensions to the classical planning problem, such that a solution plan is a partially ordered set of actions, and actions may be associated with an arbitrary cost or duration; we also briefly consider the possibility of actions with probabilistic effects.

Chapter 6 presents the application of directed unfolding to automated planning, which we refer to as Petri net Unfolding Planning (PUP). All contributions outlined in the previous section, collectively referred to as planning via directed unfolding, are found in this chapter. The chapter begins with a summary of other approaches to planning which employ a Petri net model. We then present our translation from a planning world to a PT-net, and subsequently cast the classical planning problem presented in Chapter 5 to the PT-net reachability problem presented in Chapter 3. Following this is a discussion of how PUP fits within the larger picture of automated planning algorithms. We then describe the PUP SUITE, and present empirical results for PUP versus a range of state of the art planning systems, on various International Planning Competition benchmarks. The conclusion of this chapter includes a detailed synopsis of the collaborative work, and individual contributions, which led to the development of planning via directed unfolding.

Finally, Chapter 7 concludes this thesis by highlighting the benefits and limitations of its contributions, and suggesting directions for future research.

This page left blank.

Part I

Directed Unfolding

This page left blank.

Chapter 2

Reachability Analysis

REACHABILITY IS A FUNDAMENTAL BASIS FOR STUDYING THE DYNAMIC PROPERTIES OF ANY SYSTEM
Tadao Murata [1]

This chapter introduces the reachability problem for a restricted model of discrete event systems, the impetus for solving this problem via Petri net unfolding, and the inspiration for the theory of *directed unfolding*.

The chapter begins by motivating the need to construct models in order to reason about the world, and narrows our scope from the widely applicable model of a system, to a discrete, event-driven system (DES), and finally to a restricted DES which in particular can represent a classical planning world. Here we also explain our specific focus on concurrent systems. We then define the reachability problem for such systems, describe its relevance to automated planning, formal verification and diagnosis, and consider the benefit of identifying a solution which is partially ordered and/ or optimal in some respect. We outline a straight forward approach to reachability analysis: forward state space search. Whilst this method has the quality of simplicity, and facilitates optimisation, it is impractical due to the *state explosion problem*, the fact the state space can be exponential in the number of variables used to describe it. This leads to an explanation of the "fictional" complexity which can be introduced when modelling a problem, and the particular ramifications for concurrent systems when the representation enumerates the state space and/ or does not have concurrency semantics; and here lies the key impetus for modelling a concurrent system as a Petri net and using the unfolding technique for reachability analysis. We then look at the main techniques for alleviating the state explosion problem: symbolic algorithms, partial order methods, abstraction techniques, symmetry reduction and decomposition methods,

[1] [126, p. 547]

and compare Petri net unfolding with a range of algorithms using these ideas. We then look at the inspiration for the theory of directed unfolding: the development of informative heuristic functions has significantly impacted the scalability of automated planning methods, to the extent state space search becomes viable when guided by heuristics. Whilst Petri net unfolding differs from state space search in that it avoids enumerating the state space and has concurrency semantics, there are critical similarities which make it equally amenable to optimisation and state-based heuristic guidance. In this way directed unfolding possesses some of the benefits of a state space search, whilst avoiding some of the complexity it necessitates for highly concurrent systems. In addition, it naturally finds partially ordered solutions to the reachability problem.

2.1 System Modelling

The size and complexity of the world raises problems for reasoning about it. To perform analysis we need to restrict the information considered by identifying what is relevant. In science we construct *models* to capture a particular perspective of the world appropriate to our interest in it. The volume of information considered is reduced by restricting it to pertinent components, properties and interactions. Faithfulness to reality is traded for a reduction in complexity, to the extent that the desired analysis becomes computationally feasible.

Two principle types of restriction are bounds on the situation of interest and limits on the detail of its description. The later, referred to as abstraction, is further motivated by the desire for domain-independent analysis. In order to avoid constructing an entirely new modeling concept and analysis method for every different situation, scientists conform a plethora of problems considered to have fundamental similarities, to a common formal *representation*. In this way, similar problems can be mapped to models and analysed with domain-independent methods. And, conversely, we can develop models and algorithms to solve problems for which we only know the formal representation.

2.1.1 Restricted State-Transition System

A model which finds a place in all the sciences is the *system*. Rechtin describes a system as *"a collection of things working together to produce something greater"[142, p. 1]*. This could be something tangible, such as an airplane or the human body, or intangible, for example a computer software program. Considering the range of possibilities there are clearly

numerous factors to consider when modelling and analysing a system. For example, does it interact with the outside environment? Does the system maintain a memory of past experience? Does it change in a deterministic manner? Should we describe it using continuous or discrete quantities?[2] Scientists were historically interested in systems involving natural phenomena and thus dealt with real-valued time-varying quantities such as velocity; hence the original infrastructure for system analysis was based on the assumption of a continuous state space with time-driven state transitions. However the emergence of man-made technology gave rise to systems concerned with discrete quantities, such as Boolean logic values, where changes in state depend on instantaneous events, like the clicking of a mouse button [33].

This thesis is concerned with systems of the later nature. That is, systems in which:

(a) There are clear distinctions between different states; and

(b) State transitions correspond to occurrences of discrete, asynchronous events.

A system satisfying these properties can be modelled as a *state-transition system*, also called a *discrete event system* (DES) [33]. We are interested in DES because many approaches to automated planning rely on modelling the world as a DES [67, p.5]. This is discussed further in Part II. Other, more common examples of DES include computer, communication, manufacturing, software and traffic systems. We take the following formal definition of a state-transition system from Ghallab *et al* [67, p. 5]:

Definition 1. *[State-transition system] A state-transition system is a 4-tuple* $\Omega \triangleq \langle S, A, E, \gamma \rangle$ *where:*

- $\diamond$ $S = \{s_1, s_2, \ldots\}$ *is a finite or recursively enumerable set of states;*
- $\diamond$ $A = \{a_1, a_2, \ldots\}$ *is a finite or recursively enumerable set of actions;*
- $\diamond$ $E = \{e_1, e_2, \ldots\}$ *is a finite or recursively enumerable set of events; and*
- $\diamond$ $\gamma : S \times A \times E \rightarrow 2^S$ *is a state-transition function.*

Let $s, s' \in S$, $a \in A$ and $e \in E$. If $s' \in \gamma(s, a, e)$ then the pair (a, e) can cause a *state transition* from s to s'. Let $\gamma(s, a)$ denote a state transition from s due only to action a, and similarly for events. *Actions* are transitions that are controlled by some external agent. Action $a \in A$ is *applicable* to state $s \in S$ if $\gamma(s, a)$ is not empty; applying a in s

[2] Cassandras and Lafortune [33] provide a detailed summary of system classification.

will take the system to a state $s' \in \gamma(s, a)$. Conversely *events* are contingent transitions which correspond to the internal dynamics of the system. If $\gamma(s, e)$ is not empty then e could occur when the system is in state s, transitioning it to a state in $\gamma(s, e)$ [67], but its occurrence can not be controlled. Let us impose these following assumptions on the system $\Omega = \langle S, A, E, \gamma \rangle$:

1. The set of states S is finite.
2. The result of every action and event is deterministic, i.e. for every $s \in S$, $a \in A$ and $e \in E$, $|\gamma(s, a, e)| \leq 1$.
3. The system remains static unless a controlled transition, i.e. an action, takes place. That is, the set of events E is empty.

Definition 2. *[Restricted state-transition system] A restricted state-transition system* $\Gamma \triangleq \langle S, A, \gamma \rangle$ *is a state-transition system satisfying Assumptions 1, 2 and 3.*

In Part II we show that a large proportion of research in automated planning makes assumptions about the world such that it can be modelled as a restricted state transition system [67]. This is referred to as classical planning, and is the focus of Part II.

Given that the system is deterministic, if action a is applicable to state s there will be only one element s' in $\gamma(s, a)$: for simplicity we will denote this as $\gamma(s, a) = s'$. Assuming that the event set is empty is a somewhat arbitrary decision with respect to the scope of Part I of this thesis, as we are concerned with the question of whether a state transition *could* take place, irrespective of whether it can be controlled or not. Generally speaking, the system analysis techniques considered later in this chapter are applicable independent of state transitions being actions and/ or events. Furthermore, the notion of directed unfolding presented in Chapter 4 requires no assumption regarding the underlying state transitions being controlled actions or contingent events. Given this, it is simpler to assume just one type of transition because combining actions and events requires further refinement of the model which assumes, for example, that no action takes place in states where events occur. This adds unnecessary complication for the scope of this thesis. We choose to include actions in the model, rather than events, as our application interest in automated planning requires at least the notion of controlled state transitions.

Cassandras and Lafortune [33] identify three main levels of abstraction in the study of DES: logical, timed and stochastic timed models. Logical models capture the set of all possible orderings of actions that can occur in the system. Timed models offer the set of all timed sequences of actions that a system could execute. Stochastic timed models combine

a timed model with a probability distribution function for the actions. As indicated by our model, the work of this thesis is based primarily in the first level of abstraction - the logical behaviour of the system. However in Chapter 6 we show that minor additions to this representation make it possible to answer some basic questions otherwise confined to timed models. Also, in this same chapter, we show how actions with probabilistic results can be considered.

2.1.2 Concurrent System

> *"Interrelationships are both the strength and the weakness of a system. They are responsible for its unique function on the one hand and its unavoidable complexity on the other" Rechtin [142, p 7].*

A system requires the interaction of different parts to perform its function, yet the more elements and interconnections there are the more complex a system becomes. The complexity of today's technology is in one way enabled by the ability to coordinate the interaction of largely independent components which are themselves complex systems. We are interested in systems consisting of components which operate independently for the most part, but require some interaction. Thus their internal operations are largely concurrent to each other, but their interrelationships deny their analysis as separate entities. We broadly refer to such systems as *concurrent systems*.

Many real-world planning problems require the co-ordination of agents which otherwise act largely independently. Achieving the overall objective of a military operation, for example, may depend on co-operation between various teams, acting otherwise independently to achieve individual missions. In such situations, from the perspective of military planning staff, the world is a highly concurrent system. Other examples of concurrent systems include computer networks, asynchronous circuits, operating systems, various forms of plant-controller systems, e.g. a telephone system, flight-control system, manufacturing-plant controllers, etc.

2.2 The Reachability Problem

A fundamental reason for analysing many forms of dynamical systems, is to to determine whether a particular situation can occur [126]. Given the system is in a particular state, could it possibly reach a state which satisfies certain given properties, some time in the future?

2.2.1 Definition

We formally define the reachability problem for restricted state-transition systems as:

> REACHABILITY(Γ, s_0, S_R) : Given a restricted state-transition system $\Gamma = \langle S, A, \gamma \rangle$, an initial state $s_0 \in S$ and a subset of goal states S_R, determine a sequence of actions in A,$\langle a_1, a_2, a_3, ..a_n \rangle$, corresponding to a sequence of states in S, $\langle s_0, s_1, \ldots, s_n \rangle$, such that: $s_1 = \gamma(s_0, a_1), s_2 = \gamma(s_1, a_2) \ldots, s_n = \gamma(s_{n-1}, a_n)$[3] and $s_n \in S_R$.

Note that when $|S_R| > 1$ this is sometimes referred to as the *coverability problem* [126].

2.2.2 Relevance: Automated Planning, Formal Verification and Diagnosis

This thesis looks at modelling a concurrent system as a Petri net, and employing the unfolding technique to solve REACHABILITY. More specifically, we present the theory of directed unfolding, a technique oriented to reachability analysis. Petri net unfolding is a technique developed for the verification of asynchronous circuits[119] and has since been formulated for model checking [57, 96, 79, 78], and used for model-based diagnosis [22, 21, 62, 11]. In fact it was the application of unfolding to the diagnosis of concurrent systems which first caught our attention [11]. Likewise, the theory of directed unfolding presented in this thesis is applicable to both these areas.

This thesis focuses primarily on the reachability problem in the context of planning as reachability analysis [67]. It will be shown in Part II that a classical planning problem can be cast as a REACHABILITY problem. The reachability problem also arises in formal verification and diagnosis.

Model Checking is an automatic technique for formally verifying finite state concurrent systems [40]. Whilst a planner searches for a sequence of actions to satisfy a goal, a model checker searches for a counterexample to falsify a given system specification. Reachability analysis is a main component of model checking [40]. Most safety properties can be checked directly using reachability analysis [9]; some other properties such as liveness can be translated into state reachability problems [15].

[3] Or equivalently, $(\gamma(\gamma(\ldots\gamma(s_0, e_1), \ldots, e_{k-1}), e_k) \in S_g$, given the assumption of deterministic state transitions (Assumption 2).

Automated diagnosis is concerned with identifying and isolating faulty behaviour in systems. Model Based Diagnosis [149] achieves this task by reasoning about system descriptions. Some work has been done identifying the connection between Model Based Diagnosis and reachability analysis [140]. Diagnosis problems relevant to reachability analysis relax some of the assumptions presented here, for example full observability and deterministic state transitions, and impose further constraints on a solution to the reachability problem (e.g. that a sequence of actions/events must coincide with certain observations of an only partially-observable system [149]). However we believe that applying the contributions of this thesis to diagnosis is a fruitful topic for future research. In particular, there may be benefit in combining the theory of directed unfolding with Benveniste, Fabre, Jard and Haar's ideas for equipping Petri net unfolding with information about the probability of an event [11, 1].

2.2.3 Optimal Solution

We have presented REACHABILITY as a function problem: what is a sequence of actions that can transform the system from its initial state s_0 to a state in S_R? In many cases we may be interested in the quality of this solution, seeking *optimality* with respect to some criteria. In system verification, for example, identifying the shortest execution sequence leading to a bad state can make for easier debugging. Alternatively, suppose there are no controllable actions, only actions with a probability of occurrence: the most likely sequence of actions leading to an error state is useful information for diagnosing a fault [11]. In automated planning applications, we may want to synthesise solutions which have a minimal monetary cost, and/ or minimise the total execution time (makespan) of a plan.

If there is a positive solution to REACHABILITY then, theoretically, an optimal solution can be found by identifying all possible solutions and comparing them. More practically, one attempts to find a solution without exhaustive search, e.g. on-the-fly, in a manner that guarantees it to be at least as good as any other solution.

A primary contribution of this thesis is showing how the Petri net unfolding algorithm can be used to solve REACHABILITY optimally, with respect to various cost functions. This forms one half of the theory of directed unfolding, and is presented in Chapter 4.

2.2.4 Partially Ordered Solution

It is probable that a solution to the reachability problem for a concurrent system will consist of actions which do not need to be executed sequentially to transform the system to a state in S_R. The solution could instead be a partially ordered set of actions, such that if the actions are executed in any order satisfying the partial order the system will transition from the initial state to a state in S_R.

A partially ordered solution is particularly useful when the solution represents a plan, as it provides the plan executor with more flexibility. Also, if we are interested in minimising the execution time of a plan then it is necessary to execute actions concurrently where possible.

In Chapter 3 we show how the Petri net unfolding algorithm can be used to obtain partially ordered solutions to REACHABILITY, such that actions are only ordered with respect to each other when they are causally dependent.

2.2.5 Forward State Space Search

Forward state space search is perhaps the simplest approach to solving REACHABILITY(Γ, s_0, S_R). A *reachability graph* represents the state space of Γ: nodes map to states, and there is an arc from node $s \in S$ to node $s' \in S$, labelled by $a \in A$ iff $\gamma(s, a) = s'$. A forward state space search begins at the node mapping to the initial state, and traverses the reachability graph to find a path to a node representing a state in S_R. A path from s_0 to $s \in S_R$ is a solution to REACHABILITY(Γ, s_0, S_R).

Algorithm 1 outlines the basic procedure for a forward search of the reachability graph of Γ. This algorithm terminates once a positive solution to REACHABILITY(Γ, s_0, S_R) is found, else exhaustively searches the entire reachability graph to conclude a negative solution.

Optimal Solution

A search strategy defines the way a search is executed; with respect to state space search, it defines how the reachability graph is explored. In Algorithm 1 the search strategy depends on the order in which elements are added to the queue. For example a breadth-first search strategy begins at the root node, explores all the neighbouring nodes, then explores each of their neighbouring nodes, and so on. In Algorithm 1 this is achieved by adding new elements to the end of the queue. Conversely, a depth-first search begins at the root node then explores one neighbour n, then explores one neighbour of n, and so on as far as

Algorithm 1 Forward State Space Search (Γ, s_0, S_R)

input: $\Gamma = \langle S, A, \gamma \rangle$, initial state s_0 and subset of goal states S_R.
Initialise *queue* with the element $(s_0, \emptyset, \emptyset)$. Let *nodes* $= \emptyset$ and *arcs* $= \emptyset$)
until element containing goal state is found, or the queue is empty:
    Remove the first element (s', s, a) from the queue.
    Add node s' to *nodes* and add arc (s', s), labelled by a, to *arcs*
    For every action $a \in A$ applicable to s', consider $s'' = \gamma(s', a')$:
        if $s'' \notin$ *nodes*
        then add (s'', s', a') to *queue* (i.e. reject all new paths with loops).
end
output: If a node $s \in S_R$ is found announce success and return the sequence of actions labelling the path from s_0 to s, else announce failure.

Note: if we only want to identify whether REACHABILITY(Γ, s_0, S_R) *is positive or negative (i.e. if a state in S_R is reachable from s_0 or not) then the arcs do not need to be stored.*

possible before backtracking. Algorithm 1 performs a depth-first search if new elements are added to the front of the queue.

Forward state-space search facilitates on-the-fly optimisation. Suppose we want to find a solution that minimises an objective function f. By making the queue a priority queue based on the f value of elements, this becomes a *best-first search*, i.e. it always extends the most promising path with respect to f. If f is an increasing function, which means that if there is a path from node a to node b in the reachability graph then $f(a) < f(b)$, then best-first search will return an optimal solution with respect to f [67].

Although technically simple, searching the state space is impractical even for relatively small systems, due to the problem of state explosion.

2.3 The State Explosion Problem

The *state explosion problem* refers to the exponential explosion of the size of a state space, in the number of variables used to describe a state of the system. For example, if each state $s \in S$ represents the truth or otherwise of a finite set of propositions X, then there could be 2^X states in S.

For a concurrent system, this has two ramifications with respect to the reachability graph:

1. The total number of nodes is the *product* of the size of the state space of each component. This may seem obvious and unavoidable, since it is equivalent to saying that the number of states is exponential in the number of variables used to describe the system. However if a system exhibits concurrency then it may be possible to address the state space of each component separately, to some extent, so that complexity is instead in the order of the *sum* of the of the size of the state space of each component.

2. The number of paths connecting nodes increases exponentially with the the level of concurrency. If multiple transitions are concurrent, they can be executed in any order, and the reachability graph will explicate all such orders. It has been recognised in literature that the arbitrary interleaving of concurrent actions/events contributes significantly to the state explosion problem, e.g. [10, 118, 56].

The first ramification suggests that a representation should make the variables defining a state accessible. A *factored* state representation avoids specifying the state space explicitly. It represents the state space implicitly through the state variables, i.e. variables = factors, and a representation of the transition function which exploits the fact an action usually only involves a small subset of variables. In classical planning, for example, it is common for the world to be represented by a set of propositions and a set of actions, where each action is specified by the propositions it depends upon and changes (see Chapter 5).

The *enumeration* of the set of states S is a one-to-one mapping from S onto an initial segment of the natural numbers $\{1, 2, 3 \ldots n\}$ where $n = |S|$. A *flat* state representation enumerates the set of states. Given the number of states can be exponential in the number of variables, a factored state representation can be exponentially smaller than a flat state space representation. In addition, a factored representation can can reveal structure in the system, such as regularity in the assignment of variable values, which is hidden when the state space is enumerated.

This fact has been recently observed in the study of Markov Decision Processes (MDPs), and there has been consequent movement toward factored representations of MDPs, e.g. [23, 24, 74]. A *factored MDP* represents the state space using state variables and the transition function using a dynamic Bayesian network[4] [24].

The second consequence of the state explosion problem particular to concurrent systems, suggests that a representation should support the partial order model of system execution.

[4] A Bayesian network is a directed acyclic graph where nodes map to variables and arcs specify the conditional dependencies between the variables they connect. A dynamic Bayesian network (DBN) [43] models sequences of variables by including a temporal dimension.

That is, it should facilitate avoiding the arbitrary interleaving concurrent actions during analysis.

A Petri net can be used to represent states in a factored manner, and reveal causal relations between the factors with respect to actions. An analysis technique can then exploit this information. The unfolding of a Petri net, for example, explicates all possible partially ordered runs of the net; actions are only ordered with respect to each other when they are causally dependent. In addition, the factored state representation is maintained.

The importance of a partial order model of system execution, and the benefit of Petri net unfolding for this purpose, is reflected in the work of Benveniste and colleagues. Benveniste *et al* [10, 11] have observed that the arbitrary interleaving of concurrent actions contributes significantly to the state explosion problem in system diagnosis, and have made significant attempts to alleviate this problem using the partial order semantics of Petri net unfolding. In particular they propose a structure called Markov Nets [10], which gives concurrency semantics to Markov Processes. A Markov net is generated by unfolding a Petri net, and propagating probability information such that concurrency implies probabilistic independence. Whilst intended for the analysis of non-controlled systems in which events have non-deterministic outcomes, their decision to use Petri net unfolding to address the state explosion problem in concurrent systems has been a strong motivation for our work.

It can thus be seen that in concurrent systems, the state explosion problem is particularly prevalent, but that some of problems it causes are possibly avoidable. Although modelling aims to reduce the complexity of a problem, it can actually introduce fictional complexity. For example, searching the reachability graph requires considering the state of the entire system, and imposes a total ordering on the transitions which is mandated by this representation, not by the problem itself.

2.3.1 Alleviating the Problem

If the information is not available in the initial model, it can not be exploited during analysis. And, whilst the information may be present, an analysis technique may not use it. For example, suppose a system Γ is modelled using a Petri net. Instead of unfolding the net, we could solve a reachability problem for Γ by searching its reachability graph; this process, described by Murata [126], essentially enumerates the state space. Furthermore, the causal relations are lost and a total order on actions is enforced.

We now look at some of the main techniques employed to reduce the impact of the state explosion problem when analysing a system: symbolic algorithms, partial order methods,

decomposition techniques, abstraction and symmetry reduction. These techniques are not mutually exclusive, and often several techniques are employed by the same algorithm in attempt to attack complexity from many angles. For instance [46] propose combining symmetry reduction with partial order reduction. We observe that depending on how these ideas are applied, the result is to *compress*, *reduce*, and/or *decrease* the search space. *Compression* of the search space means to represent it concisely without modifying the reachability graph. Conversely, *reduction* of the search space refers to generating a reduced reachability graph which none-the-less preserves necessary properties for the particular problem. If we limit the space in which a solution can possibly exist, this is called *decreasing* in the search space. Orthogonal to this, *on-the-fly* methods attempt to eliminate part of the computation process when state reachability is positive, by identifying a solution before an exhaustive search is made.

Petri net unfolding can be considered a symbolic, partial order method which compresses the state space. Using unfolding for on-the-fly reachability analysis, it also decreases the possible solution space by ensuring the solution lies in an identifiable subset of its complete (compressed) search space. Petri net unfolding can also be considered a decomposition method because it uses the causal relations to decompose the problem into independent subproblems, and re-combine solutions to these problems as appropriate. This is a debatable classification however since decomposition is generally considered a static technique and in Petri net unfolding the decomposition occurs in a dynamic manner, re-partitioning the search space as the causal dependencies between actions change.

Symbolic Algorithms

A *symbolic algorithm* is one which uses and exploits a factored state representation. Symbolic algorithms avoid building the state space explicitly, but rather explore some implicit representation of it. They generally maintain factored state composition, and use this to exploit structure such as regularity in the values of state variables. Symbolic Model Checking [30] is a classic example; we will discuss this shortly. As will be shown in Part II, the classical planning problem is commonly specified using a factored representation, hence most classical planning algorithms are symbolic.

McMillan presented the unfolding algorithm in his doctoral thesis [117]. The core of his thesis however was another symbolic approach, called *Symbolic Model Checking*, which had a revolutionary impact on the scalability of automatic verification [40]. Symbolic Model Checking operates on sets of states instead of individual states, and represents sets of states symbolically. It is based on the observation that systems with a large number

of components often have a regular structure which is reflected in the reachability graph. This regularity can be exposed by using propositional formulae to represent sets of states and their relations, and then exploited by methods which can manipulate these sets [30]. McMillan proposed the use of Binary Decision Diagrams (BDDs) [26], a canonical form for Boolean formulas, for this purpose. Each state is encoded as an assignment of Boolean values to state variables; the transition relation is then expressed as a Boolean formula in terms of two sets of variables - one for the old state and one for the new. BDD based search algorithms can then be used to determine if certain properties are satisfied, and produce an execution trace counter example where applicable [40].

A later approach to Symbolic Model Checking involves expressing a Boolean function in conjunctive normal form (CNF) rather than using BDDs [16]. In this way, questions such as state reachability can be solved using propositional satisfiability (SAT) algorithms. SAT-based approaches to the reachability problem are particularly prevalent in the area of automated planning, and so are discussed further in Chapter 5.

McMillan considered both Symbolic Model Checking and unfolding as a means to address the state explosion problem, by not explicitly representing states of the system. Symbolic Model Checking exploits regularity in the system, and thus works particularly well for synchronous systems such as hardware[117]; conversely, unfolding exploits concurrency and is thus more appropriate for asynchronous systems [119, 118].

Partial Order Methods

Partial Order Methods exploit concurrency by considering the partial order model of system execution. The motivation behind such approaches is that is that concurrent actions should be left unordered since their ordering is irrelevant (and imposing an order on them is in fact fictional). Important examples include Partial Order Reduction (POR) [134, 135], to be discussed shortly, and partial order planning algorithms, which are addressed in Chapter 5.

It appears William Overman [131] was the first to suggest that using the commutativity between concurrent transitions can potentially change a state space from exponential to polynomial in the number of processes. Overman considered a very restricted case and his approach was soon superseded by the work of Valmari, Godefroid, Wolper and Peled, who each made significant contribution to the family of algorithms which achieve POR. These include Valmari's *stubborn sets* [161], Godefroid and Wolper's *persistent sets* [70, 172] and Peled's *ample sets* [133]. Although differing on the details, these approaches all contain similar ideas, each seeking to identify a subset of the actions enabled at each state, which

can suffice to capture all relevant system behaviour. Like unfolding, POR methods are effective only for asynchronous systems, and can be performed on-the-fly [133, 162]. This is where the similarity ends however. As implied by the name, POR methods actually reduce the reachability graph: they do not deal with direct representations of partial orders, but rather limit the expansion of a partial order computation to just one of its interleaves [39]. This is contrary to unfolding which represents all partial orders explored so far plus the branching from one to another. Consequently, whilst it is a partial order method, POR leads to totally ordered solutions to the reachability problem. Furthermore, POR methods use global-state based analysis whereas unfolding avoids generating global-states altogether. There has been recent work combining POR with symmetry reduction [46] and symbolic model checking [101, 2], thus leading to further reduction or compression respectively, when dealing with systems of identical components or systems exhibiting regularity in the reachability graph.

Another approach to combating the exponential explosion of concurrent action sequences is non-serial dynamic programming (NSDP) [129, 160]. NSDP includes constructs that model concurrency. NSDP can optimise two concurrent processes X and Y separately, requiring joint optimisation only over the states where the processes interact. In order to use NSDP techniques for a given problem, we must first delineate the concurrent processes and their synchronisation points. This in itself is a complicated operation, and the standard approach is to generate a dependency graph (thus requiring enumeration of the state space) [160].

Decomposition Techniques

Decomposition involves breaking a problem into several sub-problems which can be solved independently; local solutions are combined to form a global solution. Decentralised algorithms, compositional reasoning, and some types of abstraction (for example hierarchical abstraction) can be considered forms of decomposition. We now look at two popular techniques employed to decompose the analysis of a system: hierarchical abstraction and compositional reasoning.

Hierarchical abstraction is a form of vertical problem decomposition, which has been developed both in the context of AI planning, e.g [5, 99, 28], and more general systems theory, e.g. [29]. When applied to the reachability problem, hierarchical abstraction involves solving REACHABILITY for a relaxed version of the system; the solution to this is a sequence of actions that transition the system though a series of states. These states are then used to define a series of intermediate reachability problems. These new prob-

lems are solved for a more detailed version of the system and combined in accordance with the preceding abstract solution. The process continues in this way, until a solution for the original system is obtained. Broadly speaking, detail is removed from the original system using *precondition-elimination*: this entails the systematic removal of action preconditions [67]. Depending on the problem instance, and particular choice of abstractions, hierarchical abstraction can either be significantly more efficient than analysis without abstraction, or be dominated by back-tracking when a particular abstract solution can not be refined and consequently have a negative effect [68]. Hierarchical abstraction can generally be implemented as a modification of other algorithms for solving REACHABILITY. Whilst beyond the scope of this thesis, there appears no reason why hierarchical abstraction could not be incorporated with on-the-fly unfolding. Furthermore, in Chapter 4 we use a notion similar to precondition-elimination for relaxing the original system in order to find approximate solutions to REACHABILITY which we use to better inform the unfolding process.

Compositional reasoning is a term commonly found in literature on formal verification, meaning to infer properties of a global system from the local properties of its components [40]. Early work on compositional reasoning exploited the parallel behaviour of a system, requiring components to form a clear hierarchy of dependencies, e.g. [41]. Subsequent approaches, e.g. [120], accommodate circular dependencies among components. Many methods are based on the *assume-guarantee* paradigm, e.g. [138, 73, 87, 124], which entails analysing each component separately, subject to assumptions regarding its environment. For instance if component M depends on component M one may need to define a set of assumptions that must be satisfied by component M', in order to guarantee some particular behaviour of component M. The range of compositional reasoning techniques capture a trade-off between efficiency and automation: more powerful methods usually require an expert user and significant manual effort; automatic techniques struggle with complex systems (and much intellectual work must still be done by the user) [12]. Whilst compositional reasoning itself is not a partial order method, like unfolding it exploits the independence of separate components. A major difference between the decomposition involved in compositional reasoning techniques and unfolding, is that unfolding identifies sub-problems dynamically, thus accommodating any dependency structure, and automatically; this obviously has a computational cost however.

Abstraction and Symmetry Reduction

The final two common approaches to dealing with the state explosion problem are abstraction and symmetry reduction. Whilst some level of abstraction has already been obtained

by representing the real world situation with a model, it may be possible to remove further detail to increase computability. For example abstraction can be used to hide internal state information or simplify the behavior of the system [110]. This may be done in a manner that maintains correctness with respect to the properties of interest, or that simply allows an approximate solution to be obtained.

Symmetry reduction techniques exploit, for example, the replication of components, or symmetric use of data values, to obtain a reduced model of the system. Such techniques are based on the observation that symmetry implies the existence of permutation groups that can be used to define an equivalence relation on the state space [40]. Emerson and Sistla [52] show how to exploit symmetry in model checking systems containing many identical or isomorphic components.

Abstraction and symmetry reduction are not techniques employed by Petri net unfolding, and future research may look at incorporating past successes in these areas to attack the state explosion problem from other angles. For example abstraction is the foundation of Hierarchical Task Networking, a popular technique in the planning community, which involves synthesising a plan using high-level actions, and recursively expanding them until a solution defined by the original low-level actions is obtained. This is similar to Suzuki and Murata's step-by-step refinement of the nodes in a Petri net [158]. This suggests that future research could consider how HTN techniques can be incorporated with Petri net unfolding. Future research may also consider the benefit of applying symmetry reduction techniques for Petri nets, such as those described in [157] and [150], prior to unfolding the net.

Finally, heuristic functions (to be discussed later) are often defined for an abstraction of the original problem which simplifies system behavior and allows computation of an approximate solution in polynomial time [147]. A contribution of this thesis is the use of heuristic functions to increase the efficiency of the Petri net unfolding technique. We show such functions can be calculated using an abstraction of the original Petri net.

2.3.2 Heuristic Search

Whilst an algorithm may be domain independent, in some cases it is possible to make it more efficient by incorporating problem-specific information in the form of *heuristics*. Pearl [132, p. vii] describes heuristics as "strategies using readily accessible though loosely applicable information to control problem-solving processes in human beings and machines". In Engineering this generally equates to rules-of-thumb for guiding learning, design and analysis [142]. In Computer Science, heuristic functions estimating the distance

between nodes in a search space are central to search control techniques [132, 147]. That is, a heuristic function may indicate which path is "cheaper", "faster", or has greater potential for leading to a state in S_R. The better this heuristic function, the more accurate and discriminating the information it provides. A heuristically informed search is sometimes referred to as *heuristic search*.

We are interested in heuristic search because solving REACHABILITY does not necessarily require exhaustive exploration of the search space; if an on-the-fly approach is taken then the search can cease as soon as a positive solution is found. The extent of the exploration thus depends critically on the search strategy employed - depth-first, breadth-first, etc. A heuristically informed strategy may identify a solution more quickly. Furthermore, heuristics can be used not only to guide, but also to prune (i.e. reduce) the search space - even when the solution is negative. In some instances a heuristic can identify that no state in S_R can be reached via a particular path. In such cases, the search can stop looking further in that direction. Using heuristics to effectively guide the search and prune the search space can make the crucial difference to solving REACHABILITY within the available time and memory limits.

Techniques for automatically extracting effective heuristics from the representation of a transition system have significantly impacted the scalability of automated planning [18, 86, 116]. This has inspired efforts to combine heuristic search with model checking algorithms, resulting in what is now referred to as *directed model checking* [50]. For example BDD-based symbolic model checking originally employed a breadth-first search. Edelkamp and Reffel later proposed a BDD-version of the A* algorithm, replacing breadth-first search with heuristic search, e.g. [143, 51]. Similarly, on-the-fly POR techniques have been guided with heuristic information, e.g. [47, 109].

Heuristic State Space Search

A forward state space search is easily guided by heuristic information. Recall that Algorithm 1 becomes a best-first search when new elements are added to a priority queue based on their f value. In standard heuristic state space search the function f comprises of a cost function g and a heuristic function h, such that $f = g + h$. Ideally the heuristic function is an oracle that specifies the distance, or cost, of the path from the current state to a state in S_R. As the purpose of the heuristic function is to avoid the complexity of calculating this exactly, in reality it is an estimate. Heuristic functions can be constructed which guarantee never to overestimate the actual distance or cost, i.e. they provide a lower bound. Heuristic functions that guarantee to provide a lower bound are called *admissible*. If Algorithm 1

is implemented as a best-first search with respect to $f = g + h$, and h is an admissible heuristic, then it will find an optimal solution with respect to g [171].

This leads to another major contribution of this thesis: significantly improving the efficiency of the Petri net unfolding algorithm, for the purpose of solving REACHABILITY, by making it a heuristic search. In Chapter 4 we show how the unfolding process can be directed using heuristic information, in a similar manner to heuristic state space search. Furthermore, heuristic functions which have proved valuable in AI planning can be extracted from the Petri net. This forms the second half of the theory of directed unfolding, complimentary to the results on optimality.

2.4 Directed Unfolding: Facilitating Optimality and Improving Efficiency

We have discussed the benefit of finding optimal solutions to REACHABILITY with respect to different criteria, and observed the success of heuristic search.

We have shown that forward state space search facilitates finding optimal solutions to REACHABILITY, and heuristic search. Whilst Petri net unfolding has some critical differences with state space search, including the fact it uses a factored state representation and considers the partial order model of system execution, there are similarities that make it equally suitable for optimisation and state based heuristic guidance. In particular:

(a) A node has a unique history, which can be used to define its cost; and,

(b) A node can be associated with a state, which can be used to estimate its distance from a state in S_R.

The typical implementation of the Petri net unfolding algorithm enables it to find a minimum length solution to REACHABILITY on-the-fly. It is not clear whether, prior to the research presented in this thesis, this guarantee of optimality was recognised nor whether other criteria besides solution length could be considered for optimisation. In fact, the only apparent attempt to use another strategy to guide the unfolding process proved unsuccessful [58]. In addition, no-one had previously endeavoured, or even suggested, utilising heuristic information to improve the efficiency of solving REACHABILITY on-the-fly via unfolding.

This thesis presents the theory of directed unfolding: controlling the unfolding process with strategies that utilise heuristic information and facilitate finding optimal solutions to

REACHABILITY. This opens the door to a family of strategies for guiding the unfolding process, where optimality can be traded for efficiency.

2.5 Conclusion

This chapter introduced the REACHABILITY problem for restricted state transition systems, and motivated our interest in reachability analysis of concurrent systems. Namely, that a classical planning world can be modelled as a restricted state transition system, and a planning problem can be cast as a reachability problem. The contribution of this thesis is not restricted to automated planning however, as the reachability problem is also prevalent in the areas of system verification and diagnosis.

This thesis advocates modelling a concurrent system as a Petri net, and using unfolding as a means to solve the reachability problem. In order to compare this with other approaches to reachability analysis, we identified the main techniques employed to combat the state explosion problem: symbolic algorithms, abstraction techniques, partial order methods, symmetry reduction and decomposition methods. Petri net unfolding is symbolic, considers the partial order model of system execution, and decomposes the problem into independent sub-problems; we subsequently summarised other approaches employing similar techniques and contrasted them with the unfolding approach.

In this chapter we also described the benefit of finding optimal solutions to REACHABILITY with respect to various criteria, and considered how heuristics can be used to guide algorithms more quickly towards a solution to REACHABILITY. In the next chapter we present the syntax and semantics of Petri net unfolding. Then, in Chapter 4 we present the theory of *directed unfolding*, which enables the unfolding process to find optimal solutions to REACHABILITY with respect to various cost functions, and makes it a heuristic search. The application of directed unfolding, to automated planning, forms Part II of this thesis.

This page left blank.

Chapter 3

Unfolding a Petri Net

CARL ADAM PETRI, WHO PIONEERED THE SCIENTIFIC MODELLING OF DISCRETE CONCURRENT SYSTEMS.
Robin Milner [1]

This chapter describes the syntax and semantics of Petri net unfolding. Following a brief summary of the ideas underlying the development of Petri nets, we introduce a class of Petri nets called place transition nets (PT-nets), and list the assumptions made in this thesis with respect to these nets. We then describe the data structure and technique of PT-net unfolding, which is a partial order semantics of PT-nets employed for their analysis. This leads to the *ERV* unfolding algorithm, which generates a finite yet complete prefix of the unfolding of a PT-net. Following this we define the reachability problem for PT-nets, outline the main approaches to solving it, and state our interest in the unfolding approach. We then formalise the connection between reachability analysis and PT-net unfolding, discuss the implications in terms of complexity theory, and finally present the fundamental technique this thesis builds on: on-the-fly reachability analysis via unfolding. The chapter concludes with a summary of its content and contributions.

3.1 Petri net

Carl Adam Petri was the first person to develop models of interacting sequential processes, in his thesis *Kommunikation mit Automaten* [137]. Petri deduced that to be practically relevant and avoid enforcing unnecessary structure on a system, a general theory of information processing should be founded on asynchronous, local operations. Petri worked to

[1] On the occasion of his acceptance speech for the Turing Award 1991 [25, p. 4]

develop a modeling technique for asynchronous distributed systems, that would avoid the construction of "fictional" global states. Since, generally, a discrete action only affects a subset of system components, Petri aimed to depict and exploit the locality of actions. He also sought to make local causes and effects both evident and reversible, and support the pragmatic manner in which people build systems. Brauer and Reisig [25] claim that these three requirements were achieved by Petri in his development of the *Petri nets* language. Petri nets is a formal and graphical language, which may be considered a generalisation of automata theory with concurrency semantics. Both can be used to represent the behaviour of a DES by representing its transition structure. In an automaton this is achieved by explicitly enumerating all states then connecting the possible transitions between them. Conversely, in a Petri net, states are not enumerated, rather state information is distributed among a set of nodes that capture key conditions governing the operation of the system [33]. Consequently, whilst automata represent a system run as a totally ordered sequence of action occurrences, Petri nets encourage a partial order on actions in accordance with the cause and effect relation. Un-order, which we refer to as *concurrency*, is a result of causal independence. Petri nets thus form a natural framework for decomposing or modularising a potentially complex concurrent system.

Over the years different types of Petri nets have evolved in attempt to accommodate the need for more expressive power. In this thesis we focus on the original low-level place transition net (PT-net). This is motivated by our desire to construct a foundation-level connection between the work on Petri nets and automated planning, and our particular interest in the Petri net unfolding technique which was originally formulated as an analysis technique for PT-nets. For those interested in exploring Petri nets further than achieved here, *Petri Nets World*[2] provides an expanse of Petri net resources including extensive bibliographies and tool databases.

3.1.1 Place Transition Net

Murata [126] and Reisig [144] each provide a detailed overview of PT-nets and their properties. Here we present only those concepts relevant to this thesis.

A *PT-net* $N \triangleq \langle P, T, F \rangle$ is a directed, bipartite graph with disjoint sets of *place* nodes P and *transition* nodes T, where $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. We use the standard graphical representation, with places shown as circles and transitions as boxes. The *preset* ${}^\bullet x$ of node x is the set $\{y \in P \cup T : (y, x) \in F\}$; its *postset* $x^\bullet$ is the set

[2] http://www.informatik.uni-hamburg.de/TGI/PetriNets/

$\{y \in P \cup T : (x, y) \in F\}$. The *marking*, i.e. state, of N maps each place to a non-negative integer: $M : P \to \{0, 1, 2...\}$. $M(p) = k$ is shown pictorially by k black dots (*tokens*) in p. We will often identify a marking M with a multi-set containing $M(p)$ copies of p for every $p \in P$. If a place p is in the multi-set of a marking M, then we say p is *marked* in M. To simulate the dynamic behaviour of the modelled system, the marking of a Petri net is changed by the occurrence of transitions. A transition can only occur when it is *enabled*. Marking M *enables* a transition t if $M(p) \geq 1$ for every $p \in {}^\bullet t$. The occurrence of an enabled transition absorbs a token from each of its preset places and puts one token in each postset place: this changes the marking from M to $M' \triangleq M \setminus {}^\bullet t \cup t^\bullet$. This is denoted as $M \xrightarrow{t} M'$.

The transitive closure of the flow relation F is the relation $<$, such that:

- $<$ is transitive;
- $F \subseteq <$; and
- For any other transitive relation R, if $F \subseteq R$ then $< \subseteq R$.

We will refer to $<$ as the causal relation. The reflexive transitive closure of F, which we denote by $\leq$, is the union of $<$ with the equality relation on $P \cup T$.

A *PT-net system* $\Sigma \triangleq \langle N, M_0 \rangle$ consists of a PT-net N and a marking M_0 which is called the *initial marking*. Figure 3.1 is an example of a small PT-net system with

$$P = \{p_a, p_b, p_c, p_d, p_e, p_f, p_g\},$$
$$T = \{t_A, t_B, t_C, t_D, t_E, t_F, t_G, t_H\} \text{ and}$$
$$M_0 = \{p_a, p_b\}.$$

In this PT-net F contains the elements $(p_a, t_B), (p_a, t_A)$ and (t_B, p_d) for example, but not (t_B, p_c).

A sequence of transitions $\sigma = (t_1, t_2 \dots t_n)$ is an *occurrence sequence* in Σ if there exist markings $M_1, M_2 \dots M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots M_{n-1} \xrightarrow{t_n} M_n$. This is denoted by $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* in Σ if and only if there exists an occurrence sequence σ such that $M_0 \xrightarrow{\sigma} M_n$. In Figure 3.1 the sequence of transitions (t_A, t_C, t_D) is an occurrence sequence corresponding to the sequence of markings

$$\{p_a, p_b\} \xrightarrow{t_1} \{p_b, p_c\} \xrightarrow{t_3} \{p_e, p_c\} \xrightarrow{t_4} \{p_f, p_g\}.$$

A PT-net system is described as *1-safe* if for every reachable marking M, $M(p) \leq 1$ for all $p \in P$.

As inferred previously, a defining property of Petri nets is their representation of causality. Specifically, in a PT-net $N = \langle P, T, F \rangle$ two nodes x and x' are:

- ◇ *Causally related*, if $x < x'$;
- ◇ In *forward conflict*, denoted by $x \# x'$, if there are distinct transitions $t, t' \in T$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ and $t \leq x$ and $t' \leq x'$; or
- ◇ *Concurrent*, denoted by $x \; co \; x'$, if neither $x \# x'$ nor $x < x'$ nor $x' < x$;

We wish to highlight another situation that may occur in a PT-net, termed backward conflict. *Backward conflict* is the case where a place has more than one upstream transition. In Figure 3.1, for example, p_g is in backward conflict as it is fed by t_E, t_D and t_H. Due to this backward conflict, if p_g contains a token we can not necessarily identify which of these transitions occurred to produce it.

If transitions t_1 and t_2 are concurrent, and both enabled by marking M, then $M \overset{(t_1,t_2)}{\rightarrow} M'$ and $M \overset{(t_2,t_1)}{\rightarrow} M'$. That is, the ordering of t_1 and t_2 in an occurrence sequence is arbitrary; in fact, it is not even necessary that they be interleaved as the sets of places each interacts with are disjoint. The concurrency semantics of Petri nets motivates us to define the concept of an *occurrence poset*. Occurrence posets are essentially occurrence sequences without any fictional ordering imposed on them; ordering constraints are limited to the causal relation. An occurrence poset $\sigma_\triangleright$ consists of a multi-set of transitions $T' = \{t_1, t_2, \ldots t_n\}$ and a set of ordering constraints $\triangleright$ over the transitions, of the form $t_i \triangleright t_j \Rightarrow t_i$ must precede t_j[3], such that $t_i \triangleright t_j \Rightarrow t_i < t_j$, and any sequence containing all elements in T' and satisfying the ordering constraints is an occurrence sequence. Note that every occurrence sequence captured by a particular occurrence poset $\sigma_\triangleright$ will transform the net to the same marking M_n; we denote this by $M_0 \overset{\sigma_\triangleright}{\rightarrow} M_n$. In Figure 3.1 the set of transitions $\{t_A, t_C, t_D\}$ together with the constraints $\{t_A, t_C\} \triangleright t_D$ is an occurrence poset. It captures the previously mentioned occurrence sequence (t_A, t_C, t_D), and the occurrence sequence (t_C, t_A, t_D). The execution of either of these occurrence sequences will transform the PT-net to marking $\{p_f, p_g\}$.

3.1.2 General Assumptions

The scope of this thesis is restricted to PT-net systems with the following properties:

[3]Multiple instances of the same transition may be subject to different constraints.

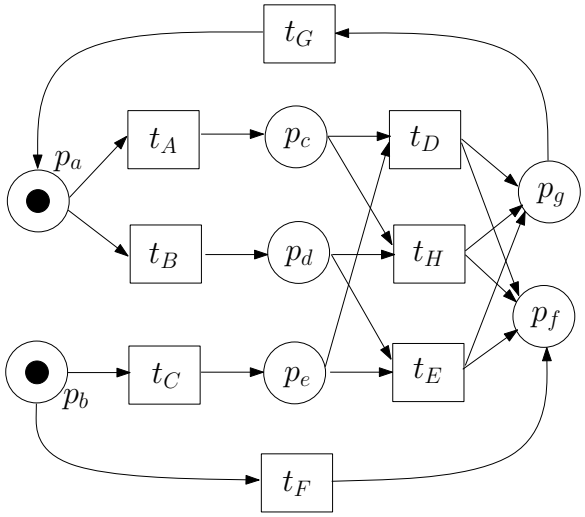


Figure 3.1: Example PT-net system. Places=circles, transitions=boxes and tokens=dots.

(1) The number of places and transitions is finite;

(2) The PT-net system is 1-safe; and

(3) Every transition has a non-empty preset and a non-empty post-set.

A consequence of (2) is that the initial marking is restricted to: $M_0 \rightarrow \{0, 1\}$.

3.2 Unfolding a Place Transition Net

The partial order semantics of a Petri net was traditionally the set of its "processes", e.g. [71, 144], where a process models a run of the net from its initial marking. The work of Nielsen, Plotkin and Winskel [125], continued over a decade later by Engelfriet [53], took these partial order semantics to a higher level by considering the relationship between different processes. Rather than contemplating each run of the net separately, one looks at a single branching run termed a *branching process*. This represents several runs of the net, together with an explicit indication of choice between alternatives. Intuitively, in a branching process, whenever there is a forward or backward conflict the model branches into each possible independent resolution of the conflict. Note that processes share a common representation, where applicable, up to the conflict. The procedure for obtaining a branching process from a PT-net system is called *unfolding*. McMillan consolidated the unfolding theory for practical application, proposing an algorithm to obtain a part of the branching process that could be used for the verification of asynchronous circuits [119]. Esparza,

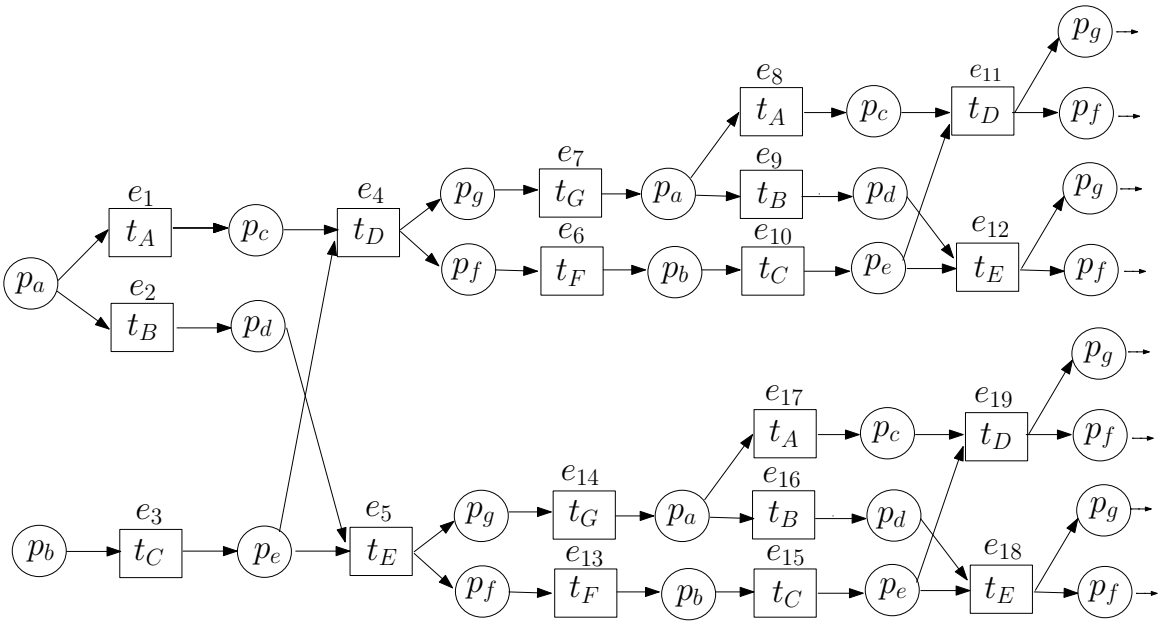


Figure 3.2: Unfolding of the PT-net system in Figure 3.1. Conditions=circles, events=boxes. Inside the condition and event nodes are references to the places and transitions they map to in the original PT-net. In addition, each event is given a unique label, shown above its box.

Römer and Vogler developed a more general and transparent structure around McMillan's unfolding algorithm, to produce what we shall refer to as the *ERV* unfolding algorithm [56].

3.2.1 Unfolding: Representation and Method

Branching Process

Consider how a rooted graph can be unfolded into a labelled tree, with each branch corresponding to a different walk through the graph (from the root node). There is a homomorphism mapping nodes and arcs of the tree to nodes and arcs of the graph. Similarly a PT-net can be unfolded into a labelled occurrence net, called a branching process. A branching process of a net comprises of an occurrence net, and a mapping to the original net. We now define the net, then the mapping, and finally combine them to form a branching process.

An *occurrence net* is a PT-net $ON \triangleq \langle B, E, G \rangle$ where:

- ◇ There is no backward conflict;

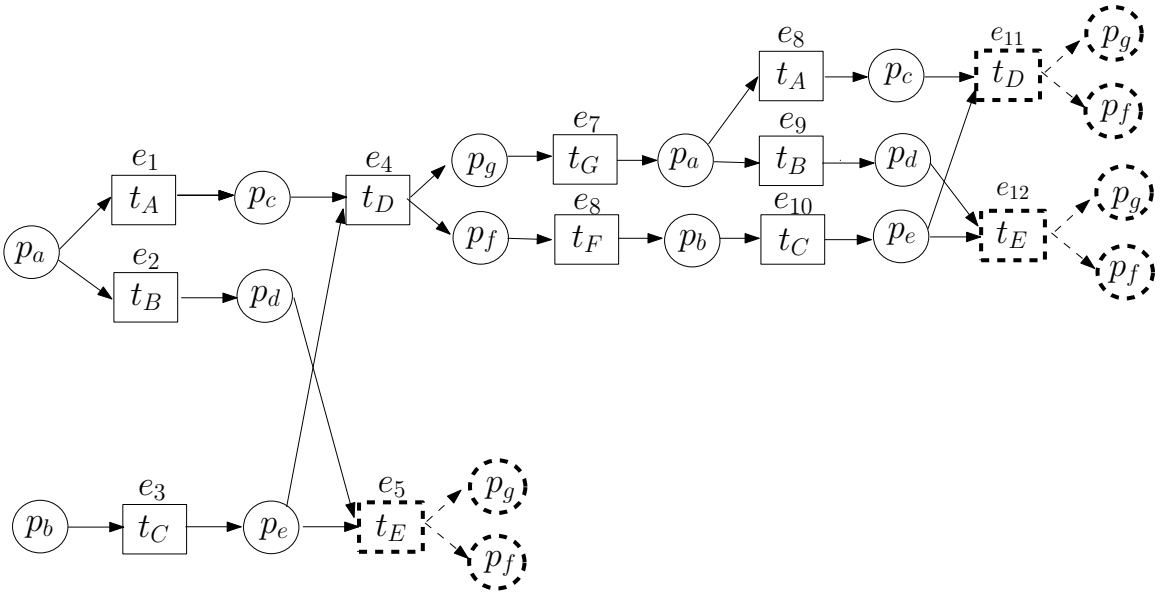


Figure 3.3: Finite prefix of the unfolding of the PT-net system in Figure 3.1. The events drawn with broken lines are those identified as cut-offs.

- ◇ ON is finitely preceded, i.e. for every $x \in B \cup E$ the set of elements $y \in B \cup E$ such that $y < x$ is finite, which means there are no cycles; and

- ◇ No element is in forward conflict with itself (this is also called self-conflict).

As is standard in the literature, to differentiate this particular type of PT-net in discussion, we refer to the set of places B as *conditions*, and the set of transitions E as *events*. G is the causal relation. A causal chain of events is a finite or infinite sequence of events such that if event e precedes event f in the chain then $e < f$.

When an occurrence net is obtained by unfolding a net, the conditions and events represent particular occurrences of the places and transitions in the original net, respectively. The represented places and transitions can necessarily be reached and enabled in the original net, from the initial marking, via the distinct paths leading to them in the occurrence net. That is, the occurrence net represents occurrence sequences in the original net. The unfolding method achieves this by eliminating backward conflict[4], cycles and self-conflict. Whilst the reachability graph of a PT-net also represents occurrence sequences, a defining characteristic of unfolding is that it does not represent these individually, nor impose order on their elements unnecessarily: the unfolding of a PT-net system consists of occurrence

[4]In planning terms, the elimination of backward conflicts achieves the property of *post-uniqueness* of the action set [6], which implies that we know the exact set of actions that causes a state variable to have a certain value at some point in the plan.

posets.

We denote by *Min(0N)* the set of minimal elements of ON (with respect to $<$). In the scope of this thesis, these elements are necessarily conditions since we only consider nets in which every transition has an non-empty preset (Assumption 3).

A *homomorphism* φ, from an occurrence net ON to a PT-net system Σ, is a mapping such that

- ◇ Conditions map to places and events to transitions: $\varphi(B) \subseteq P$ and $\varphi(E) \subseteq T$;
- ◇ Transition environments are preserved: for every $e \in E$, φ restricted to ${}^\bullet e$ is a bijection onto ${}^\bullet\varphi(e)$ and similarly for $e^\bullet$;
- ◇ Minimal conditions correspond to the initial marking: φ restricted to *Min(ON)* is a bijection onto the multi-set M_0; and
- ◇ There is no redundancy: for all $e, e' \in E$ if ${}^\bullet e = {}^\bullet e'$ and $\varphi(e) = \varphi(e')$ then $e = e'$.

A *branching process* of Σ is a pair $\beta_\Sigma \triangleq \langle ON, \varphi \rangle$ where φ is a homomorphism from ON to Σ. The nodes of the branching process are uniquely defined by the nodes they map to in the original net *and* the particular partially ordered history of events that led to them (from the initial marking).

As a PT-net can be unfolded to varying extents, there is a natural partial order on its branching processes. A branching process $\beta'_\Sigma = \langle ON', \varphi' \rangle$ is a prefix of a branching process $\beta_\Sigma = \langle ON, \varphi \rangle$, denoted by $\beta'_\Sigma \sqsubseteq \beta_\Sigma$, if ON' is a sub-net of ON satisfying:

- ◇ *Min(ON') = Min(ON)*;
- ◇ If a condition is in ON' then its preset event in ON is also in ON'; and
- ◇ If an event is in ON' then its preset and postset conditions in ON are also in ON'.

A PT-net system Σ has a unique maximal branching process with respect to the prefix relation [53]; this is called the *unfolding* of Σ and is denoted here by Unf_Σ. Specifically, the process is unique up to isomorphism (renaming of the conditions and events). The unfolding of the PT-net system in Figure 3.1 is infinite; we show part of it in Figure 3.2.

Configuration

To understand the unfolding of a net (both the technique and representation), the most important notions are those of configurations and their final markings. Intuitively, a configuration is one process in the branching process: it represents, in the occurrence net, an occurrence poset in the original PT-net system. The final marking of a configuration is the state the original net would be in, if the process it represents were to occur. We now formalise these concepts.

A *configuration* of an occurrence net is a set of events C which are:

- ◇ Conflict free, i.e. for all $e, e' \in C$, e is not in forward conflict with e'; and
- ◇ Causally closed, i.e. if $e' \in C$ and $e < e'$ then $e \in C$.

A configuration in a branching process of Σ maps to an occurrence poset in Σ. The events map to transitions, and the constraints are based on the causal relation over events. In the unfolding shown in Figure 3.2, $\{e_1, e_3, e_4, e_6, e_7\}$ is a configuration, subject to the constraints $e_1 \triangleright \{e_1, e_3\} \triangleright \{e_6, e_7\}$. This configuration captures four occurrence sequences in the original net, each of which transform it to the marking $\{p_a, p_b, p_e\}$.

$C \oplus \mathcal{E}$ denotes that $C \cup \mathcal{E}$ is a configuration, obtained by extending configuration C with the finite set of events $\mathcal{E}$ disjoint from C. Two sets of events $\mathcal{E}_1$ and $\mathcal{E}_2$ are *structurally isomorphic*, denoted by $\mathcal{E}_1 \backsim \mathcal{E}_2$ if the directed graphs induced by each set of events and their adjacent conditions are isomorphic up to a relabelling [35].

The *local configuration* of an event e, denoted $[e]$, is the minimal configuration containing e. For example, in Figure 3.2, $[e_{14}] = \{e_2, e_3, e_5, e_{14}\}$.

A *coset* is a set of conditions which are all pairwise concurrent. A *cut* is a maximal coset, with respect to set inclusion. Every configuration is associated with a cut. The cut of configuration C is the set of conditions:

$$Cut(C) \triangleq \left(Min(ON) \cup \bigcup_{e \in C} [e]^\bullet \right) \setminus \left(\bigcup_{e \in C} {}^\bullet[e] \right)$$

The cut of configuration C in branching process $\beta_\Sigma = \langle ON, \varphi \rangle$ corresponds to a set of places in Σ given by $\text{Mark}(C) \triangleq \varphi(Cut(C))$. This is called the *final marking* of C and represents a reachable marking of Σ, i.e. if $\sigma_\triangleright$ is the occurrence poset captured by C, then $M_0 \xrightarrow{\sigma_\triangleright} \text{Mark}(C)$. For instance, in Figure 3.1, the final marking of configuration $\{e_2, e_3, e_5, e_{13}\}$ is $\{p_g, p_b\}$. Every marking represented in a branching process of a PT-net system Σ is reachable in Σ. Furthermore, every reachable marking of Σ is represented in Unf_Σ [56].

Algorithm 2 *Unfolding*(Σ)

input: PT-net system $\Sigma = \langle P, T, F, M_0 \rangle$.
Let $Unf_\Sigma = \langle ON, \varphi \rangle$, where $ON = \langle B, E, G \rangle$.
Initialise $B = \{b | \varphi(b) = p \text{ and } p \in M_0\}$.
Initialise a queue with the possible extensions of Unf_Σ.
while queue is not empty:
    Remove a possible extension (t, X) from the queue;
    Add new event e to Unf_Σ, such that $\varphi(e) = t$, ${}^\bullet e = X$;
    For every $p \in t^\bullet$ add a new condition c to Unf_Σ, such that $\varphi(c) = p$ and ${}^\bullet c = e$;
    Add all new possible extensions of Unf_Σ to the queue.
endwhile
output: $Unf_\Sigma = \langle ON, \varphi \rangle$ where $ON = \langle B, E, G \rangle$.

Possible Extensions

Finally, before presenting the algorithm for unfolding, we identify how to arbitrarily extend a branching process. Let β_Σ be a branching process of the PT-net system Σ. The *possible extensions* to β_Σ are the pairs (t, X) where t is a transition of Σ and X is a coset of conditions in β_Σ satisfying:

- $\varphi(X) = {}^\bullet t$; and
- (t, X) is not already in β_Σ, i.e. there does not exist $e \in \beta$ such that $\varphi(e) = t$ and ${}^\bullet e = X$.

The Unfolding Algorithm

The *Unfolding* algorithm (see Algorithm 2) initialises the branching process Unf_Σ with conditions corresponding to the initial marking of Σ, and no events. The possible extensions are identified, and a single event is added to Unf_Σ together with its postset conditions. New possible extensions are identified, and another event and its postset conditions added. The algorithm continues in this way until no possible extensions remain. The *Unfolding* algorithm is not guaranteed to terminate, because the unfolding of a net-system can be infinite.

3.2.2 A Complete Finite Prefix of the Unfolding

The maximal branching process of a PT-net system Σ, i.e. Unf_Σ, is finite if and only if Σ has no infinite occurrence sequences [56]. We seek a finite yet complete prefix of Unf_Σ which contains as much information as Unf_Σ. Formally:

Definition 3. *The prefix $\beta_\Sigma \sqsubseteq Unf_\Sigma$ is complete if and only if for every marking M reachable in Σ:*

(a) There exists a configuration $C \in \beta_\Sigma$ such that Mark$(C) = M$; and

(b) For every transition t enabled by M there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and $\varphi(e) = t$.

In his thesis, McMillan proposes an algorithm for generating a complete finite prefix of the unfolding of a PT-net system [117]. His approach is founded on identifying those events at which we can cease unfolding without loss of information. Such events are referred to as *cut-off* events, and are defined in the context of constructing β_Σ:

Definition 4. *Let $\prec$ be a partial order on configurations. During the unfolding of a PT-net system, an event e is identified as a cut-off with respect to $\prec$ if the branching process constructed so far already contains some event e' such that:*

(a) Mark$([e])$ = Mark$([e'])$; and

(b) $[e'] \prec [e]$.

In his thesis, McMillan uses a partial order on configurations $\prec_m$ based on cardinality:

$$C \prec_m C' \Leftrightarrow |C| < |C'|.$$

He presents an algorithm similar in essence to Algorithm 2 with the addition that the queue is ordered with respect to $\prec_m$ and every event taken from the queue is tested against the definition of a cut-off event. The postset conditions of events identified as cut-offs are not used in the generation of possible extensions. McMillan proves that unfolding in this manner guarantees a finite and complete prefix is obtained [119]. The reasonning behind this is that two events with the same marking will continue unfolding isomorphically to each other. The continuation from either event may involve conflicting parts of the net however so the cutting needs to be consistent. McMillan's unfolding algorithm maintains

consistency by ensuring that for every reachable marking M, the smallest configuration C such that Mark$(C) = M$ will never contain a cut-off event.

Esparza, Römer and Vogler [56] later identified that any partial order on configurations satisfying certain conditions, may be used to identify cut-offs. They refer to suitable orders as *adequate orders*:

Definition 5. *A partial order $\prec$ on finite configurations is adequate if*

(a) $\prec$ is well founded;

(b) $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$; and

(c) $\prec$ is preserved by finite extensions: if $C_1 \prec C_2$ and Mark(C_1) = Mark(C_2), then for all finite extensions $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$ such that $\mathcal{E}_1$ and $\mathcal{E}_2$ are structurally isomorphic, we have $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.

When used to identify cut-offs, condition (b) ensures the prefix will be finite. Conditions (a) and (c) prevent the inconsistent cutting of events just mentioned.

The ERV Unfolding Algorithm

A finite complete prefix can be built using the *ERV* algorithm (see Algorithm 3). This algorithm maintains a priority queue in which the events to be added to the unfolding are sorted in increasing order of $\prec$ (with respect to their local configuration). At each iteration, a minimal event is extracted from the queue and added to the unfolding together with all the conditions in its postset. Additionally, all the events enabled by the new conditions are inserted into the priority queue. The algorithm finishes when the queue becomes empty.

It is shown in [56] that if $\prec$ is an adequate order then the output of *ERV*$(\Sigma, \prec)$, i.e. β_Σ, is a complete and finite prefix of Unf_Σ.

MOLE: An Implementation of The ERV Algorithm

MOLE[5] is a freeware program which unfolds 1-safe PT-nets. It implements the *ERV* algorithm using an adequate order $\prec_{erv}$ on configurations which refines McMillan's order $\prec_m$ by comparisons based on Parikh-vectors and the Foata normal form. This makes $\prec_{erv}$

[5] `http://www.fmi.uni-stuttgart.de/szs/tools/mole/`

Algorithm 3 *ERV* $(\Sigma, \prec)$

input: Net system $\Sigma = \langle P, T, F, M_0 \rangle$; partial order on configurations $\prec$.

Let $\beta_\Sigma = \langle ON, \varphi \rangle$, where $ON = \langle B, E, G \rangle$.

Initialise $B = \{b | \varphi(b) = p \text{ and } p \in M_0\}$.

Initialise priority queue with the possible extensions of β_Σ.

Note: The queue of possible extensions is sorted by the local configurations of the events they define, in increasing order w.r.t. $\prec$.

while queue is not empty:

  Remove the first possible extension (t, X) from the queue;

  Consider the event e with $\varphi(e) = t$ and ${}^\bullet e = X$:

  Add e to β_Σ, such that $\varphi(e) = t$ and ${}^\bullet e = X$;

  if e is not a cut-off w.r.t. $\prec$

    For every $p \in t^\bullet$ add a new condition c to β_Σ, such that $\varphi(c) = p$ and ${}^\bullet c = e$;

    Insert new possible extensions into the queue.

  endif

endwhile

For every cut-off event e, for every $p \in \varphi(e)^\bullet$ add a new condition c to β_Σ, such that $\varphi(c) = p$ and ${}^\bullet c = e$.

output: β_Σ, a prefix of Unf_Σ.

a total order, thus maximising the number of cut-off events identified, and consequently minimising the size of the generated prefix [59].

The prefix in Figure 3.1 is the complete finite prefix that MOLE generates for our example. The events e_5, e_{11}, and e_{12} are all cut-off events. This is because each of their local configurations, firstly, has the same marking as the local configuration of event e_4, i.e. $\{p_f, p_g\}$, and, secondly, is greater than the local configuration of event e_4 with respect to the adequate partial order implemented by MOLE. Notice that the finite prefix of the unfolding ceases at cut-off events.

3.3 The Reachability Problem

We define the *reachability problem* for 1-safe PT-nets as follows:

REACHABILITY$_\Sigma$: Given a 1-safe PT-net system $\Sigma = \langle P, T, F, M_0 \rangle$ and a

subset $P' \subseteq P$, determine whether there is an occurrence sequence σ such that $M_0 \xrightarrow{\sigma} M$ where $M(p) = 1$ for all $p \in P'$.

Note that this is often referred to as the *coverability problem* in Petri net literature, e.g. [126].

There are two main approaches to solving REACHABILITY$_\Sigma$. The first is to search the reachability tree of the Petri net [126], which is equivalent to performing a state space search of the modelled system. This method can be used to identify a *totally ordered* solution to REACHABILITY$_\Sigma$. Searching the entire reachability tree will obviously suffer greatly from the state explosion problem. Consequently there are numerous techniques designed to reduce the reachability tree, including abstraction techniques for simplifying a net whilst preserving some properties (this includes what are referred to as reduction techniques [13, 104, 139] in Petri net literature) and methods to identify symmetry relations [157, 150]. There are also decomposition methods to divide and conquer the building of the reachability tree, including top-down synthesis [158, 111] which involves the substitution of nodes with sub-nets, and bottom-up synthesis [36, 100] where sub-nets are combined in an iterative manner.

The second common approach to Petri net analysis is based on methods of linear algebra, using a matrix representation of a Petri net [126]. A major drawback of this method, for our purposes, is that whilst it can be used to identify the transitions which solve REACHABILITY$_\Sigma$ it does not provide any information about their ordering.

Unfolding[6] is a less common method of Petri net analysis. The fact that unfolding is the partial order semantics of a Petri net means we can identify solutions to REACHABILITY$_\Sigma$ which are constrained only by causal relations; in addition, as this thesis shows, unfolding can be employed to solve REACHABILITY$_\Sigma$ optimally with respect to criteria which must necessarily consider concurrency semantics, such as makespan.

3.3.1 Connection with Unfolding

We have discussed the fact that a configuration in a branching process of Σ represents an occurrence poset in Σ, which in turn captures a set of occurrences sequences, each of which transform Σ to the same marking. So, considering the problem of REACHABILITY$_\Sigma$ for Σ and P', a configuration $C \in \beta_\Sigma$ with final marking M such that $P' \subseteq M$ represents at

[6] Whilst not within the scope of this thesis, it appears that various techniques used to reduce the reachability tree could in fact be applied prior to unfolding, as they statically reduce the Petri net model.

least one occurrence sequence which is a solution to the reachability problem. It may in fact represent multiple solutions. Consequently, in this thesis we will commonly refer to configurations as solutions to REACHABILITY$_\Sigma$, without referring to the homomorphism that maps them onto occurrence sequences in the original net. We hope that this does not confuse, but rather is sufficiently intuitive to simplify discussion.

We now formalise the connection between the solution to REACHABILITY$_\Sigma$ for a PT-net system Σ, and a complete prefix of the unfolding of Σ. As mentioned previously, every marking represented in a branching process of Σ is reachable in Σ [56]. Furthermore, for every reachable marking M of Σ, a complete prefix of the unfolding of Σ contains at least one configuration C such that Mark$(C) = M$ (by part (a) of Definition 3). We can thus conclude the following:

Proposition 3.3.1 (Reachable marking)**.** *Let β_Σ be a complete prefix of Unf_Σ. A marking M is reachable in Σ iff β_Σ contains a configuration C such that Mark$(C) = M$.*

From this we can make the connection between a reachability problem for a PT-net system, and a complete prefix of its unfolding:

Corollary 3.3.2 (REACHABILITY$_\Sigma$ via β_Σ)**.** *Consider the problem of* REACHABILITY$_\Sigma$ *defined by PT-net system $\Sigma = \langle P, T, F, M_0 \rangle$, and subset $P' \subseteq P$. Let β_Σ be a complete prefix of Unf_Σ. Then:*

(a) There is a positive solution to REACHABILITY$_\Sigma$ *iff β_Σ contains a configuration C such that Mark$(C) = M$ where $P' \subseteq M$; and*

(b) C represents a solution to REACHABILITY$_\Sigma$.

Proof. Part (a) follows directly from Proposition 3.3.1 and the definition of REACHABILITY$_\Sigma$. Part (b) follows from part (a), and the fact a configuration represents an occurrence sequence in the original net (by construction of a branching process of Σ, and the definition of configuration). □

3.3.2 Complexity

In terms of complexity theory the REACHABILITY$_\Sigma$ problem for a 1-safe net Σ is PSPACE-complete [38]. This means it can be solved by a deterministic Turing machine using a length of work tape which is polynomial in the size of the input.

From another perspective, REACHABILITY$_\Sigma$ is NP-complete in the size of the finite prefix of Σ [60]. That is, a Turing machine can check in polynomial time whether a given solution of polynomial length is correct. And thus, a non-deterministic Turing machine which always guesses the correct answer, can solve the problem in polynomial time. Note that NP $\subseteq$ PSPACE.

But, whilst the reachability problem is PSPACE-complete, a finite complete prefix of the unfolding of Σ can require space which is exponential in the size of the original Petri net. This suggests the unfolding is not optimal for solving reachability. The picture is not so grim however, as this is the worst case scenario for unfolding and we can generally identify which problems will be much easier in practice. In some cases a complete prefix of the unfolding of Σ is exponentially smaller than the state space of Σ. Everything in between depends on a range of factors that have yet to be precisely analysed. What is clear intuitively, and supported by empirical results (see the Artificial Problems presented in Chapter 6), is that as the level of concurrency in the system increases the size of the unfolding prefix can decrease exponentially with respect to the number of reachable states. This is due to the fact unfolding exploits concurrency by essentially breaking the problem into independent sub-problems. So given our interest in concurrent systems, the unfolding of a PT-net is an enticing approach to reachability analysis despite its theoretical complexity.

There are several algorithms that solve REACHABILITY$_\Sigma$ for a PT-net system Σ and subset of places P', via a complete prefix of the unfolding of Σ. These are based on:

- ◇ *Linear programming.* Melzer's [121] linear programming approach is based on an algebraic equation representation of the set of reachable markings of an acyclic net. The complete finite prefix is an acyclic net that can be employed to represent Σ for this purpose. This approach has exponential complexity in the size of the complete prefix (which is itself exponential in the size of the Petri net) [60].

- ◇ *SAT.* Heljanko [79] translates a complete finite prefix of the unfolding of Σ into a rule based logic program, and then checks if a model exists. This reduces a reachability problem for Σ to SAT. The algorithm has exponential complexity in the size of the complete prefix [60].

- ◇ *Graph theory.* Shröter and Esparza [60] propose a graph-theoretic approach based on whether conditions mapping to P', in the finite prefix of the unfolding of Σ, are concurrent. This extends the concurrency relation defined earlier over pairs of nodes to sets of conditions. This approach is polynomial in the size of the prefix and exponential in the cardinality of P'.

◇ *Local reachability / On-the-fly.* A *local reachability problem* for a Petri net Σ involves determining whether a particular transition can be enabled; this can be achieved by identifying whether it is represented by an event in the finite prefix of the unfolding of Σ. This local reachability problem is polynomial in the size of the complete prefix [60]. McMillan [119] proposes an algorithm for solving REACHABILITY$_\Sigma$ on-the-fly by mapping the reachability problem for Σ to a *local* reachability problem for an extended net Σ_R. The apparent reduction in complexity (from NP-complete in the size of Σ to polynomial in the size of Σ_R) suggests that it may take exponentially more time to generate a complete prefix for Σ_R than for Σ.

Esparza and Schröter [60] empirically compare the above approaches and conclude that algorithms which map the problem to SAT and local reachability are the most efficient. The SAT approach is often more efficient than that of local reachability when the solution to the problem is negative. For the case of a positive solution, suppose n reachability problems are to be solved for the same PT-net system. If $n = 1$ then solving the local reachability problem on-the-fly appears consistently and significantly more efficient than solving the SAT problem [60]. If $n > 1$, then there is a break point for n, dependent on the particular problem, at which the SAT approach is more efficient.

The on-the-fly approach is attractive for our purposes, as we are interested in solving single reachability problems (i.e. $n = 1$). This thesis improves the efficiency of on-the-fly reachability analysis, using the concept of directed unfolding, by reducing the size of the prefix which must be generated to solve both positive and negative reachability problems. Furthermore, the concept of directed unfolding enables us to find optimal solutions to reachability, with respect to various criteria.

3.3.3 On-the-fly Reachability Analysis via Unfolding

Unfolding a Petri net to solve REACHABILITY$_\Sigma$ on-the-fly was first suggested by McMillan [119]. The original net is extended by a single transition, t_R, such that ${}^\bullet t_R = P'$ and $t_R{}^\bullet = P'$, i.e. this transition can only be enabled by a marking that satisfies REACHABILITY$_\Sigma$[7]. The extended net is then unfolded, as described previously, but stops when an event e_R is retrieved from the queue, where $\varphi(e_R) = t_R$. Throughout this thesis we will use e_R to denote an event which satisfies this homomorphism. If such an event is found we conclude the set of places P' is reachable, and a solution to REACHABILITY$_\Sigma$ is the local configuration of e_R, without e_R itself. If no such event is identified, the algorithm will

[7] Setting the post-places of t_R is arbitrary; we do it here as it simplifies the reasoning for correctness.

continue to generate the prefix of the unfolding until there are no more possible extensions. If we know the prefix generated is complete, with respect to Definition 3, then at this point we can conclude P' is not reachable.

In the literature, the process of reachability analysis via on-the-fly unfolding is generally presented in the manner of the previous paragraph; to our knowledge it has not been formalised to the extent which follows. In particular, we formally define the PT-net system extended by transition t_R, present an algorithm called *ERV-Fly* which takes this as input, and subsequently identify and prove the conditions under which the later solves REACHABILITY$_\Sigma$.

Let us first define a PT-net system Σ_R, which captures a reachability problem defined by Σ and P' by including t_R appropriately:

Definition 6. *Given a problem of* REACHABILITY$_\Sigma$ *defined by PT-net system* $\Sigma = \langle P, T, F, M_0\rangle$ *and subset* $P' \subseteq P$*, define a new PT-net system* $\Sigma_R \triangleq \langle P, T \cup t_R, F \bigcup_{p\in P'}(p, t_R) \cup (t_R, p), M_0\rangle$*. We say that* Σ_R *defines the* REACHABILITY$_\Sigma$ *problem for* Σ *and* P'*.*

Extending Σ by transition t_R is arbitrary in that it does not alter the set of reachable markings. The only effect of t_R is that any occurrence sequence leading to some marking M such that $M(p) = 1 \forall p \in P'$ can be extended by t_R; as t_R does not change the marking, the possible continuation of such sequences does not change. Thus we can propose a new version of Proposition 3.3.1:

Proposition 3.3.3 (Reachable marking II)**.** *Let* Σ_R *be an extension of PT-net system* Σ*, as given in Definition 6. Let* β_{Σ_R} *be a complete prefix of* Unf_{Σ_R}*. A marking* M *is reachable in* Σ *iff* β_{Σ_R} *contains a configuration* C *such that Mark*$(C) = M$*.*

From this follows a new version of Corollary 3.3.4:

Corollary 3.3.4 (REACHABILITY$_\Sigma$ via β_{Σ_R})**.** *Consider the problem of* REACHABILITY$_\Sigma$ *for PT-net system* $\Sigma = \langle P, T, F, M_0\rangle$*, and subset* $P' \subseteq P$*. Let* Σ_R *define* REACHABILITY$_\Sigma$*, and let* β_{Σ_R} *be a complete prefix of* Unf_{Σ_R}*. Then:*

(a) There is a positive solution to REACHABILITY$_\Sigma$ *iff* β_{Σ_R} *contains a configuration* C *such that Mark*$(C) = M$ *where* $M(p) = 1$ *for all* $p \in P'$*; and*

(b) C *represents a solution to* REACHABILITY$_\Sigma$*.*

Proof. Follows from Corollary 3.3.2, Proposition 3.3.1 and Proposition 3.3.3. □

The ERV-Fly Algorithm

We can now present an algorithm *ERV-Fly*, see Algorithm 4, which uses Σ_R to solve the associated reachability problem. The difference between the original *ERV* algorithm and *ERV-Fly* is highlighted in italics. If implemented with an appropriate order on configurations, $\prec$, then *ERV-Fly*$(\Sigma_R, \prec)$ will solve the problem of REACHABILITY$_\Sigma$ defined by the PT-net system Σ_R:

Theorem 3.3.5 (*ERV-Fly* solves REACHABILITY$_\Sigma$)**.** *If ERV*$(\Sigma_R, \prec)$ *generates a finite and complete prefix of the unfolding of* Σ_R*, then ERV-Fly*$(\Sigma_R, \prec)$ *solves* REACHABILITY$_\Sigma$ *(as defined by* Σ_R*).*

Proof. Let β_{Σ_R} be the prefix of Unf_{Σ_R} generated by *ERV-Fly*$(\Sigma_R, \prec)$. Let β'_{Σ_R} be the prefix of Unf_{Σ_R} generated by *ERV*$(\Sigma_R, \prec)$. By inspection of Algorithm 3 and Algorithm 4, we see $\beta_{\Sigma_R} \sqsubseteq \beta'_{\Sigma_R}$, with equality occurring when there is no event e_R in β_{Σ_R}, where $\varphi(e_R) = t_R$. In the following we say *ERV-Fly* is *generating* a finite and complete prefix if β'_{Σ_R} is finite and complete.

If *ERV-Fly* is generating a finite prefix then the existence or not of e_R will be determined in finite time. If *ERV-Fly* is generating a complete prefix, then by part (b) of Definition 3 and construction of t_R, it will find an event e_R if and only if there exists $C, C' \in \beta'_{\Sigma_R}$ such that $C' = C \cup \{e_R\}$ and Mark$(C) = M$ where $M(p) = 1$ for all $p \in P'$. By Corollary 3.3.4 this will be true if and only if REACHABILITY$_\Sigma$ is positive, in which case C is a solution. The minimal such C', with respect to $<$, is $C' = [e_R]$, in which case $C = [e_R] \setminus \{e_R\}$.

Thus if *ERV*$(\Sigma_R, \prec)$ generates a finite and complete prefix of the unfolding of Σ_R, then *ERV-Fly*$(\Sigma_R, \prec)$ finds an event e_R if and only if REACHABILITY$_\Sigma$ is positive, in which case $[e_R] \setminus \{e_R\}$ is a solution. □

MOLE: An Implementation of the ERV-Fly Algorithm

MOLE provides the option to cease generating a branching process when an event mapping to a user-specified transition is removed from the queue of possible extensions. That is, we can request MOLE to stop executing the *ERV* algorithm when an event e_R is found such that $\varphi(e_R) = t_R$. In addition to this, we made minor amendments to enable MOLE to identify the local configuration of such an event, and return the associated occurrence poset. In this way MOLE can be used to implement *ERV-Fly*.

Algorithm 4 *ERV-Fly*$(\Sigma_R, \prec)$

input: PT-net system $\Sigma_R = \langle P, T, F, M_0 \rangle$; partial order on configurations $\prec$

Let $\beta_\Sigma = \langle ON, \varphi \rangle$, where $ON = \langle B, E, G \rangle$.

Initialise $B = \{b | \varphi(b) = p \text{ and } p \in M_0\}$.

Initialise priority queue with the possible extensions of β_Σ.

Note: The queue of possible extensions is sorted by the local configurations of the events they define, in increasing order w.r.t. $\prec$.

while queue is not empty:

Remove the first possible extension (t, X) from the queue;

Consider the event e with $\varphi(e) = t$ and ${}^\bullet e = X$:

if $t \in t_R$ *let* $\sigma = [e] \setminus e$*, and exit loop.*

Add e to β_Σ, such that $\varphi(e) = t$ and ${}^\bullet e = X$;

if e is not a cut-off w.r.t. $\prec$

For every $p \in t^\bullet$ add a new condition c to β_Σ, such that $\varphi(c) = p$ and ${}^\bullet c = e$;

Insert new possible extensions into the queue.

endif

endwhile

For every cut-off event e, for every $p \in \varphi(e)^\bullet$ add a new condition c to β_Σ, such that $\varphi(c) = p$ and ${}^\bullet c = e$.

output: *If* t_R *is found announce success and return* σ *else announce failure*.

Note: Depending on the data structure used, when defining the preset of a node one may also want to define the associated postset relation, e.g. $\forall c \in X \; c^\bullet = e$.

3.4 Conclusion

This chapter presented and motivated the application of PT-net unfolding to solving the reachability problem for concurrent systems on-the-fly. In particular, we formalised the *ERV-Fly* Algorithm and identified and proved the conditions under which it would solve the reachability problem for 1-safe nets.

PT-nets are a model for concurrent systems which, in particular,

(a) Avoid the construction of "fictional" global states by providing a factored representation of the state space; and,

(b) Depict the causal, conflict and concurrency relations between actions by making their causes and effects explicit. This provides a framework for:

- ◇ Considering the partial order model of system execution, and
- ◇ Decomposing a potentially complex concurrent system.

Unfolding is a PT-net analysis technique which exploits the above qualities. Noting that unfolding can generate an infinite structure, referred to as the unfolding of the PT-net, we presented the *ERV* Algorithm, which generates a finite and complete prefix of the unfolding of a PT-net when implemented with an appropriate strategy.

Whilst there are various ways to use the prefix of the unfolding of a net to answer reachability problems, the on-the-fly approach is most appropriate for our interest in planning as reachability analysis. We formulated the *ERV-Fly* Algorithm which, to our knowledge, has not been presented to this level of detail previously. In addition we proved the conditions which ensure it is sound and complete with respect to solving REACHABILITY$_\Sigma$; again, this formalisation has not appeared previously in the literature.

In the next section, we present the theory of *directed unfolding*, which makes *ERV-Fly* a principled method for reachability analysis. It considers problem-specific information when deciding which configuration to next extend, in attempt to direct the building of a branching process toward an event e_R, such that $\varphi(e_R) = t_R$, and to provide the possibility of finding optimal solutions to REACHABILITY$_\Sigma$ with respect to various criteria.

3.4.1 Personal Contribution

The particular formalisation of Σ_R and the *ERV-Fly* Algorithm shown here is my own work. This provided the structure for me to identify and prove the conditions under which the later is sound and complete with respect to solving REACHABILITY$_\Sigma$. The notion of an occurrence poset is also my own contribution.

This page left blank.

Chapter 4

Directed Unfolding

The Petri net unfolding technique has gained the interest of researchers in verification [57], diagnosis [11] and, motivated by the research presented in this thesis, automated planning [84]. All have reason to analyse state reachability in concurrent transition systems, looking to unfolding for some relief of the state explosion problem. Unfolding a Petri net reveals all possible partially ordered runs of the net, without necessitating the arbitrary combinatorial interleaving of independent events. Whilst the full unfolding of a Petri net can be infinite, McMillan identified the possibility of a finite prefix containing all reachable states [119]. Esparza, Römer and Vogler generalised his approach, to produce the now commonly used *ERV* algorithm [59]. This algorithm provides the framework to control the building of the prefix using different strategies. Typically, it is implemented to generate a prefix in a breadth-first manner, using the number of events in a configuration to select the next node to add (and determine terminating nodes). The MOLE unfolding tool follows this strategy.

Of the various unfolding based reachability techniques, experimental results indicate on-the-fly analysis (see Algorithm 4: *ERV-Fly*) is most efficient for solving a single REACHABILITY$_\Sigma$ problem [60]. Nevertheless, generating a prefix in a breadth-first manner quickly becomes impractical when it is wide or the shortest path to a marking satisfying REACHABILITY$_\Sigma$ is deep. Furthermore, the only guarantee regarding the quality of the solution is that it has a minimal number of events. It has not been obvious what other strategies can be used to control building the prefix; recent results have shown a depth-first strategy is incorrect [58].

Perhaps the reason why more informed strategies have not yet eventuated is that unfolding was traditionally used to prove the absence of deadlocks: this set the focus on making the entire complete prefix smaller rather than on reducing the search space explored to find a particular marking. In the context of the REACHABILITY$_\Sigma$ problem, there are two unique

factors to consider:

(1) Particular applications are concerned with the quality of a positive solution, seeking optimality with respect to some criteria.

(2) For on-the-fly analysis via unfolding, REACHABILITY$_\Sigma$ is translated to the problem of checking whether transition t_R can be enabled. Using the *ERV-Fly* algorithm, a positive solution to REACHABILITY$_\Sigma$ is identified as soon as an event e_R, such that $\varphi(e_R) = t_R$, is selected for addition to the branching process being generated. It is then unnecessary to build the remainder of the prefix of the unfolding.

An unfolding algorithm for reachability analysis that does not consider (1) and exploit (2) is probably not achieving its full potential. The basic idea behind our work is: when solving a REACHABILITY$_\Sigma$ problem defined by Σ_R, *ERV-Fly*$(\Sigma_R, \prec)$ can be considered a "search process" in quest of a configuration that includes e_R, where the order $\prec$ defines the "search strategy" by selecting which configuration to extend next. Thus, for the purpose of optimality $\prec$ should favor those configurations which "cost less", and for the purpose of efficiency $\prec$ should favor configurations with final markings that appear "closer" to a marking that enables t_R. We use this reasonning to turn unfolding into an informed algorithm oriented at solving the reachability task. We define sound search strategies that incorporate a cost function and a heuristic function, to achieve the desires of point (1), and to exploit point (2). The heuristic function is used to estimate the minimum cost, or shortest distance, from a given state to one which satisfies REACHABILITY$_\Sigma$. The resulting approach is called *directed unfolding* as opposed to the standard "blind unfolding" approach. The term "directed" has been used elsewhere to emphasise the informed nature of other model-checking algorithms [50].

Techniques for automatically extracting suitable heuristics from the representation of a transition system and using them to guide search have significantly impacted the scalability of automated planning [18, 86, 116]. We show that heuristic values can be similarly calculated from a Petri net. If the chosen heuristic is *admissible* (it never overestimates the actual minimum cost), then directed unfolding identifies an optimal solution with respect to the cost function. Provided REACHABILITY$_\Sigma$ is positive, directed unfolding can solve much larger problems than the original breadth-first *ERV-Fly* algorithm. Moreover, its implementation requires only minor additions to the latter.

In this chapter, we first consider the role of an order on configurations $\prec$ in *ERV*$(\Sigma, \prec)$ and *ERV-Fly*$(\Sigma, \prec)$; this provides grounding for the subsequent discussion in Section 4.2 and Section 4.3 regarding directing the unfolding process via more informative implementations

of this order. We also inspect the fact that if $\prec$ is adequate then the *ERV* algorithm will generate a finite and complete prefix of the unfolding of Σ; this leads to the observation that condition (b) of an adequate order (Definition 5) is a sufficient but not necessary condition to guarantee the prefix is finite. We consequently define a weaker order, which we call *semi-adequate*, that also ensures finiteness and completeness. The result of semi-adequate orders opens the door to a new family of strategies for directing the unfolding.

Section 4.2 looks at directing the unfolding method to solve optimisation problems. We formally define the concept of an optimal solution to $\text{REACHABILITY}_\Sigma$ and identify conditions which ensure *ERV-Fly*$(\Sigma_R, \prec)$ will find an optimal solution, if one exists, with respect to $\prec$. These conditions are satisfied by transitive adequate orders on configurations, but not necessarily by semi-adequate orders. We then extend the net system model to include a cost function for transitions, and subsequently define additive and parallel cost functions for configurations. Finally we propose an adequate order that can be used to optimise the additive cost function, and a semi-adequate order for optimising the parallel cost function.

Section 4.3 formulates the idea of using a heuristic function to direct *ERV-Fly* to extend configurations which appear most likely to be solutions to the reachability problem. We present the general framework for a semi-adequate order incorporating a cost function and a heuristic function. It is based on three inputs: (1) a cost function for configurations, (2) a semi-adequate order that prefers minimal configurations with respect to this cost function, and (3) a heuristic function. The conditions for optimality are satisfied by this strategy when the heuristic function is admissible. In this case, the resulting semi-adequate order is guaranteed to direct *ERV-Fly* to an optimal solution relative to the cost function (if one exists).

Section 4.4 discusses the size of the prefix that may be generated using directed unfolding.

Section 4.5 describes some of the reasoning used to develop heuristic functions in the area of automated planning. We show how heuristic values can be extracted directly from a Petri net, presenting functions that consider the distance between configurations in terms of additive and parallel cost.

Section 4.6 presents experimental results comparing "blind" and directed unfolding. More experimental results are presented in Part II.

We finally conclude this chapter by summarising the main contributions made here, in the context of directed unfolding.

4.1 Reconsidering Adequate Orders

Here we contemplate the role of an order on configurations $\prec$ in generating a prefix of the unfolding of a net system Σ via *ERV*$(\Sigma, \prec)$. One use of this order is to select the next event, from the list of possibilities, to add to the current branching process. An event is selected when its local configuration is minimal, with respect to $\prec$, among the local configurations of other events that could be added. The order in which events are added to the branching process determines the manner in which the branching process grows. For example, a breadth-first strategy involves building all local configurations of just one event, then building all local configurations of two events, then three, etc. A breadth-first strategy is achieved by employing an order based on the cardinality of a configuration, i.e. $C \prec C' \Leftrightarrow |C| < |C'|$, as advocated by [119, 59].

The second use of $\prec$ in the the *ERV* algorithm is to determine whether a particular event selected for addition to the branching process is a cut-off (see Definition 4). An event e is identified as a cut-off if the prefix already contains an event e' such that Mark$(e) =$ Mark(e') and $e' \prec e$. If e is identified as a cut-off then no subsequent event e'', is added to the branching process, such that $e < e''$. For our purposes, the decision to cut-off at particular events should ensure the prefix is finite, whilst maintaining completeness as given by Definition 3. Esparza, Römer and Vogler [56] identified that if $\prec$ is an adequate order, then *ERV*$(\Sigma, \prec)$ will generate a finite and complete prefix of Unf_Σ. We have recognised that a weaker condition on this order can still guarantee finiteness and completeness.

4.1.1 A Semi-Adequate Order on Configurations

Upon revising the role of an adequate order $\prec$ in *ERV*$(\Sigma, \prec)$, we found that condition (b) of adequacy, i.e. $C \subset C' \Rightarrow C \prec C'$ (see Definition 5), is only required to guarantee the generated prefix is finite. Indeed, let n be the number of reachable markings of the PT-net system Σ and consider an infinite sequence of events $e_1 < e_2 < e_3 < \cdots$ in the unfolding of Σ. Then, there exists $i < j \leq n + 1$ such that Mark$([e_i])$ = Mark$([e_j])$, and since $[e_i] \subset [e_j]$, condition (b) implies $[e_i] \prec [e_j]$. Thus when *ERV*$(\Sigma, \prec)$ takes $[e_j]$ from the queue it will identify it as a cut-off event due to the existence of e_i (Definition 4). Hence the generated prefix is finite [59]. A similar result can be achieved if condition (b) is replaced by the weaker condition that in every infinite chain $e_1 < e_2 < e_3 < \cdots$ of events there are $i < j$ such that $[e_i] \prec [e_j]$. We thus define:

Definition 7. *A partial order $\prec$ on finite configurations is semi-adequate if*

(a) *$\prec$ is well founded, i.e. it has no infinite descending chains;*

(b) *In every infinite chain $C_1 \subset C_2 \subset C_3 \subset \cdots$, there are $i < j$ such that $C_i \prec C_j$; and*

(c) *$\prec$ is preserved by finite extensions: if $C_1 \prec C_2$ and Mark$(C_1) =$ Mark(C_2), then for all finite extensions $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$ such that $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic, we have $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.*

Condition (b) of an adequate order (Definition 5) implies condition (b) of a semi-adequate order. Thus an adequate order is semi-adequate, but the reverse is not necessarily true. Many orders presented in this chapter are semi-adequate, but not adequate.

Theorem 4.1.1 (Finiteness and Completeness)**.** *If $\prec$ is a semi-adequate order on configurations, then ERV($\Sigma, \prec$) generates a finite and complete prefix of the unfolding of Σ.*

Proof.

- ◇ *Finiteness.* Khomenko, Koutny and Vogler extend König's Lemma to branching processes, proposing that a branching process is infinite if and only if it contains an infinite causal chain of events [97, p. 9]. Let $\prec$ be a semi-adequate order on configurations and suppose that *ERV*$(\Sigma, \prec)$ generates a branching process with an an infinite causal chain of events $e_1 < e_2 < e_3 < \cdots$. Each event e_i defines a configuration $[e_i]$ with marking Mark$([e_i])$, and since the number of markings is finite, there is at least one marking that appears infinitely often in the chain. Let $e'_1 < e'_2 < e'_3 < \cdots$ be an infinite subchain such that Mark$([e_1])$ = Mark$([e_j])$ for all $j > 0$. It follows from the definition of a configuration that if $e'_1 < e'_2 < e'_3 < \cdots$ then $[e'_1] \subset [e'_2] \subset [e'_3] \subset \cdots$. Thus by condition (b) of a semi-adequate order, there are $i < j$ such that $[e_i] \prec [e_j]$. This, in conjunction with the fact Mark$([e_i])$ = Mark$([e_j])$, implies that when e_j was taken from the queue it would have been identified as a cut-off event and thus the chain cannot be infinite. Thus the prefix generated by *ERV*$(\Sigma, \prec)$ is finite.

- ◇ *Completeness.* Let Unf_Σ be the maximal branching process of Σ. Let $\beta_\Sigma \sqsubseteq Unf_\Sigma$ be the branching process generated by *ERV*$(\Sigma, \prec)$ where $\prec$ is a semi-adequate order on configurations. Recall that every reachable marking of Σ is represented in Unf_Σ. Furthermore, because a semi-adequate order $\prec$ is well-founded (condition (a)) for every reachable marking M in Σ there must be a minimal configuration $C \in Unf_\Sigma$ such that Mark$(C) = M$. We want to show that there can be no event $e \in C$ which is a cut-off in β_Σ. For a proof by contradiction let $C = [e] \oplus E$ for some set of events E and suppose e is a cut-off event in β_Σ. This means there must be some other event

e' such that Mark$(e') =$ Mark(e) and $[e'] \prec [e]$. Because $\prec$ is preserved by finite extensions (condition (c) of a semi-adequate order) this implies $C' = [e'] \oplus E' \prec [e] \oplus E = C$ for some set of events E' such that $E' \backsim E$. This contradicts the minimality of C. Hence C must contain no cut-off events. Thus for every reachable marking M in Σ there is a configuration $C \in \beta_\Sigma$ such that Mark$(C) = M$. This satisfies condition (a) of the definition of a complete prefix of Unf_Σ (see Definition 3). Furthermore, for every transition t enabled by M, it must be that $C \oplus e \in \beta_\Sigma$, where $\varphi(e) = t$, because C contains no cut-off event and thus all possible extensions are included in the prefix. This satisfies condition (b) of the definition of a complete prefix. Thus *ERV*$(\Sigma, \prec)$ generates a complete prefix of the unfolding of Σ.

Comment: Proposition 4.9 in [59, p. 14] states that the prefix computed by ERV$(\Sigma, \prec)$ *is complete when $\prec$ is an adequate order on configurations. The proof provided for this suffices for the case when $\prec$ is a semi-adequate order, since it is based only on conditions (a) and (c) of an adequate order (see Definition 5), which are common to the definition of a semi-adequate order. That is, the proof of completeness does not rely on condition (b) at all. We chose to include a (slightly different) proof here to encourage understanding and increase the self-containment of this work.*

□

4.2 Directing the Unfolding for Optimality

In this section we consider how to use *ERV-Fly* to obtain an optimal solution to REACHABILITY$_\Sigma$ with respect to a function $g : C \to \mathbb{R}^+$, where C is a configuration. Recall that through the formulation of Σ_R, REACHABILITY$_\Sigma$ is cast to the problem of finding an occurrence sequence including t_R. The *ERV-Fly* algorithm returns the first such sequence found, if one exists, and stops. If the process continued, it would not necessarily find an optimal occurrence sequence including t_R as the selection of cut-off events may prematurely terminate optimal paths to t_R. Thus, to find an optimal solution on-the-fly we must ensure it exists in the finite prefix *and* corresponds to the first event pulled from the queue such that $\varphi(e_R) = t_R$.

4.2.1 A Notion of Optimality

Let Unf_{Σ_R} be the maximal branching process of net system Σ_R. We denote by $\mathbf{e}_R$ the set of all local configurations $[e_R]$, such that $\varphi(e_R) = t_R$ and $e_R \in Unf_{\Sigma_R}$. Thus for

every $[e_R] \in \mathbf{e}_R$, the configuration $C = [e_R] \setminus \{e_R\}$ represents a solution to the reachability problem defined by Σ_R. We say that a configuration $C^* = [e_R^*] \setminus \{e_R^*\}$ is an *optimal solution* to REACHABILITY$_\Sigma$ if $[e_R^*] \in \mathbf{e}_R$ and $[e_R^*]$ is optimal among the members of $\mathbf{e}_R$, with respect to some criterion.

We first identify the conditions on $\prec$ which guarantee *ERV-Fly*$(\Sigma_R, \prec)$ to find an optimal solution to REACHABILITY$_\Sigma$ with respect to $\prec$. As is the case with optimal search in the state space, to perform an optimal search on-the-fly via unfolding we need to ensure that every event e in an optimal solution $[e_R^*]$ is processed before the last event of a non-optimal solution is found. Since events are queued with respect to $\prec$, we require that

$$e \in [e_R^*] \Rightarrow [e] \prec [e_R]$$

where $[e_R^*] \in \mathbf{e}_R$ is minimal and $[e_R] \in \mathbf{e}_R$ is a non-minimal with respect to $\prec$, i.e. $[e_R^*] \prec [e_R]$.

Theorem 4.2.1 (Optimal REACHABILITY$_\Sigma$ with respect to $\prec$)**.** *Let Σ_R define an instance of* REACHABILITY$_\Sigma$*. Let $\prec$ be a semi-adequate order on configurations. ERV-Fly($\Sigma_R, \prec$) solves* REACHABILITY$_\Sigma$*. Furthermore, if positive, it will identify a minimal solution with respect to $\prec$ if $\prec$ is transitive and $e \in [e_R^*] \Rightarrow [e] \prec [e_R]$, where $e_R^*, e_R \in \mathbf{e}_R$ & $[e_R^*] \prec [e_R]$.*

Proof. We need to show that all events in an optimal solution will be processed before a sub-optimal solution is found.

First, observe that if the solution to the reachability problem is positive, then for *every* $[e_R^*]$ which is a minimal member of $\mathbf{e}_R$ with respect to $\prec$, the queue will contain an event $e \in [e_R^*]$. If no such event is in the queue then one of the events in $[e_R^*]$ must have been identified as a cut-off. If $e \in [e_R^*]$ is a cut-off event then there exists some e', already in the built prefix, such that $\text{Mark}(e') = \text{Mark}(e)$ and $[e'] \prec [e]$. But this implies $[e'] \oplus \mathcal{E}' \oplus e_R = [e_R] \prec [e] \oplus \mathcal{E} \oplus e_R^* = [e_R^*]$ for some structurally isomorphic finite extensions $\mathcal{E}'$ and $\mathcal{E}$ (since $\prec$ is semi-adequate and thus preserved by finite extensions), which contradicts the minimality of $[e_R^*]$. So the queue always contains some event $e \in [e_R^*]$.

Now, by Theorem 3.3.5 and Theorem 4.1.1 we know if $\prec$ is a semi-adequate order then *ERV*$(\Sigma_R, \prec)$ returns a solution to REACHABILITY$_\Sigma$. By inspection of Algorithm 4, we see if REACHABILITY$_\Sigma$ is positive then the solution returned corresponds to the local configuration of the first event e_R taken from the queue such that $\varphi(e_R) = t_R$ (i.e. $[e_R] \in \mathbf{e}_R$). We need to prove that configuration $[e_R]$ is minimal with respect to $\prec$ among members of $\mathbf{e}_R$. For a proof by contradiction, assume that the configuration $[e_R]$ of the first event e_R found by *ERV-Fly* is not minimal.

Since $[e_R]$ is non-minimal and $\prec$ is transitive, there must be at least one $[e^*_R] \in \mathbf{e}_R$ such that $[e^*_R]$ is minimal with respect to $\prec$ and $[e^*_R] \prec [e_R]$[1]. Let $e \in [e^*_R]$ be an event in the queue when $[e_R]$ is dequeued. But, by the condition in this theorem, this means $[e] \prec [e_R]$ and consequently e_R could not have been dequeued before e. Thus, by contradiction, $[e_R]$ must be minimal with respect to $\prec$ among members of $\mathbf{e}_R$, and so $C = [e_R] \setminus \{e_R\}$ is a minimal solution to REACHABILITY$_\Sigma$ with respect to $\prec$. □

Evidently, this captures the reason why the original cardinality based order can be used to find a solution with minimal cardinality. If $\prec$ is a transitive adequate order on configurations then *ERV-Fly*$(\Sigma_R, \prec)$ will solve the reachability problem optimally with respect to $\prec$. To see this recall that property b of an adequate order states $[e] \subseteq [e'] \Rightarrow e \prec e'$. Considering the above theorem, $e \in [e^*_R]$ thus implies $[e] \prec [e^*_R]$, or $e = e^*_R$; furthermore since $[e^*_R] \prec [e_R]$, if $\prec$ is transitive then $[e] \prec [e_R]$. Thus, *ERV-Fly* implemented with the transitive, adequate order $C \prec C' \Leftrightarrow |C| < |C'|$ will find a solution to the reachability problem with minimal cardinality. Note that a transitive semi-adequate order does not necessarily guarantee the condition in Theorem 4.2.1 holds, since property (b) is weaker for semi-adequacy than adequacy.

If $\prec$ is not a total order then for the purpose of optimisation we may wish to further discriminate between pairs of configurations which are not comparable with respect to $\prec$. However this extra discrimination may not be appropriate for the selection of cut-offs. So rather than ordering the queue of possible events by $\prec$, we could instead employ an order $\prec_q$ that refines $\prec$, i.e. $C \prec C' \Rightarrow C \prec_q C'$, but still use $\prec$ for the determination of cut-off events. We must furthermore require that the order $\prec_q$ is *asymmetric*, i.e. $C \prec_q C' \Rightarrow \neg(C' \prec_q C)$, to ensure $C \prec_q C' \Rightarrow \neg(C' \prec C)$. This order can thus provide further discrimination, without affecting the completeness of the generated prefix. *ERV*$(\Sigma, \prec, \prec_q)$ and *ERV-Fly*$(\Sigma, \prec, \prec_q)$ will denote the *ERV* and *ERV-Fly* algorithms implemented with a priority queue ordered by $\prec_q$, where Σ and $\prec$ refer to the input PT-net and the order on configurations used to determine cut-off events. Providing $\prec_q$ is asymmetric and refines $\prec$, the use of $\prec_q$ to order the queue simply allows control of previously arbitrary decisions. Thus, theorems involving *ERV* and *ERV-Fly* remain applicable.

Now, we want to find an optimal solution with respect to the function $g : C \to \mathbb{R}^+$. Again, we want to ensure that every event e in some optimal solution $[e^*_R]$ is processed before the last event of a non-optimal solution is found. Since events are now queued with respect to $\prec_q$, and we seek a minimal solution with respect to g, we require that:

[1]There could be other minimal members which are not comparable with $[e_R]$ with respect to $\prec$.

$$e \in [e^*_R] \Rightarrow [e] \prec_q [e_R]$$

where $[e^*_R] \in \mathbf{e}_R$ is minimal and $[e_R] \in \mathbf{e}_R$ is a non-minimal with respect to g, i.e. $g([e^*_R]) < g([e_R])$. Furthermore, we need to make sure there exists at least one optimal solution containing no cut-off events. We can do this by requiring that an event in a solution can only be cut off by an event which leads to a solution of equal or lesser cost. That is, if $e \in [e_R]$, where $[e] \oplus \mathcal{E} \oplus e_R = [e_R]$, then

$$[e'] \prec [e] \;\&\; \text{Mark}(e') = \text{Mark}(e) \Rightarrow g([e'] \oplus \mathcal{E}' \oplus e'_R) \leq g([e] \oplus \mathcal{E} \oplus e_R)$$

for $\mathcal{E}' \sim \mathcal{E}$.

More generally, we could require that the semi-adequate cut-off order implies the natural order defined by the cost function g, i.e. $[e'] \prec [e] \Rightarrow g([e']) \leq g([e])$. Then, since a semi-adequate order is preserved by finite extensions, it follows that $e' \prec e \;\&\; \text{Mark}([e']) = \text{Mark}([e]) \Rightarrow [e'] \oplus \mathcal{E}' \prec [e] \oplus \mathcal{E} \Rightarrow g([e'] \oplus \mathcal{E}') \leq g([e] \oplus \mathcal{E})$ for any $\mathcal{E}' \sim \mathcal{E}$. Observe that when seeking minimality with respect to a semi-adequate order $\prec$ this requirement is superfluous. Later, in the construction of a semi-adequate order for use in the optimisation of a particular cost function g, we will seek to meet this more general requirement as it is more intuitive.

Theorem 4.2.2 (Optimal REACHABILITY$_\Sigma$ wrt g)**.** *Let Σ_R define an instance of* REACHABILITY$_\Sigma$*. Let $\prec$ be a semi-adequate partial order on configurations. Let $\prec_q$ be an asymmetric partial order on configurations such that $C \prec C' \Rightarrow C \prec_q C'$. ERV-Fly($\Sigma_R, \prec, \prec_q$) solves* REACHABILITY$_\Sigma$*. Furthermore, if* REACHABILITY$_\Sigma$ *is positive, it will identify an optimal solution with respect to $g : C \to \mathbb{R}+$ if*

(a) $e \in [e^*_R] \Rightarrow [e] \prec_q [e_R]$*, where* $e^*_R, e_R \in \mathbf{e}_R \;\&\; g([e^*_R]) < g([e_R])$*; and*

(b) $[e'] \prec [e] \;\&\; \textit{Mark}(e') = \textit{Mark}(e) \;\&\; [e] \oplus \mathcal{E} \oplus e_R = [e_R] \in \mathbf{e}_R \Rightarrow g([e'] \oplus \mathcal{E}' \oplus e'_R) \leq g([e] \oplus \mathcal{E} \oplus e_R)$ *for* $\mathcal{E}' \sim \mathcal{E}$.

Proof. In the proof for minimality with respect to $\prec$ we first showed that for *every* $[e^*_R]$ which is a minimal member of $\mathbf{e}_R$ with respect to $\prec$, the queue will contain an event $e \in [e^*_R]$. Whilst this is still true, we are now interested in members of $\mathbf{e}_R$ which are minimal with respect to g. We show that for *at least one* $[e^*_R]$ which is a minimal member of $\mathbf{e}_R$ with respect to g, the queue will contain an event $e \in [e^*_R]$. If the queue does not contain such an event, then one of the events in $[e^*_R]$ was identified as a cut-off. If $e \in e^*_R$ was a cut-off event

then there exists some e', in the built prefix, such that $\text{Mark}(e') = \text{Mark}(e)$ and $[e'] \prec [e]$. But by condition b of this theorem, this implies $g([e'] \oplus \mathcal{E}' \oplus e'_R) \leq g([e] \oplus \mathcal{E} \oplus e^*_R)$ for some structurally isomorphic finite extensions $\mathcal{E}'$ and $\mathcal{E}$. So $[e'] \oplus \mathcal{E}' \oplus e'_R = [e'_R]$ must be a minimal member of $\mathbf{e}_R$ with respect to g, and the queue must contain an event $e'' \in [e'_R]$ in the extension of $[e']$, i.e. $[e'] \subset [e''] \subseteq [e'_R]$. If not, then we apply the same reasoning again. Since $\prec$ is semi-adequate and thus well-founded, eventually we end with an event $e \in [e^*_R]$ in the queue, where $[e^*_R]$ is minimal with respect to g.

We now follow the same reasoning as in the previous proof to show that all events in an optimal solution must be processed before a non-optimal solution is found. We need to prove that the local configuration of the first event e_R found by *ERV-Fly* is minimal with respect to g among members of $\mathbf{e}_R$. For a proof by contradiction, assume that the configuration $[e_R]$ for the first event e_R found by *ERV-Fly* is not minimal with respect to g.

Let $e \in [e^*_R]$ be an event in the queue when $[e_R]$ is dequeued, where $[e^*_R]$ is minimal among $\mathbf{e}_R$ with respect to g. So $e \in [e^*_R]$ and $g([e^*_R]) < g([e_R])$. But, by condition a of this theorem, this means $[e] \prec_q [e_R]$ and consequently e_R could not have been dequeued before e. Thus, by contradiction, $[e_R]$ must be minimal with respect to g among members of $\mathbf{e}_R$, and so $C = [e_R] \setminus \{e_R\}$ is a minimal solution to $\text{REACHABILITY}_\Sigma$ with respect to g. □

4.2.2 Notions of Cost

We now extend the unfolding algorithm to consider positive cost functions for transitions and events, and then combine them in additive and parallel manners to define different cost functions for configurations.

An extended net is a tuple $\langle P, T, F, c' \rangle$ where $\langle P, T, F \rangle$ is a PT-net and $c' : T \to \mathbb{R}^+$ is a cost function defined on transitions. To unfold an extended net system, we need to deal with extended prefixes $\beta = \langle ON, \varphi \rangle$ where $ON = \langle B, E, G, c \rangle$ is an extended occurrence net with the cost function $c(e) = c'(t)$ if $\varphi(e) = t$, where φ is the usual homomorphism that maps conditions/events in ON onto places/transitions in the PT-net.

This cost function can be extended over a configuration C under additive criteria as follows:

Definition 8 (Additive Cost)**.**

$$c_+(C) = \sum_{e \in C} c(e)$$

In other words, the additive cost of a configuration is the sum of the costs of all events in the configuration. Clearly, if $c(e) = 1$ then the additive cost of a configuration is the number of events in the configuration, i.e. $c_+(C) = |C|$.

Alternatively, the cost function can be extended over a configuration C under parallel criteria:

Definition 9 (Parallel Cost)**.**

$$c_{||}(C) = \max_{\sigma \in C} \sum_{e \in \sigma} c(e)$$

where σ is a causal chain of events in C, and the maximum is over all such chains.

Consider all the causal chains in C and assign each a value equal to the sum of the costs of the events it contains: the parallel cost of C is the maximum over all these values. For a more intuitive understanding of parallel cost, consider that the cost of an event is equal to its time duration. Then, the parallel cost of a configuration is the minimum possible time it can take to execute, and the causal chain defining this minimum time is the critical path.

4.2.3 Optimal Cost Reachability Analysis

In order to use the *ERV-Fly* algorithm to find optimal configurations with respect to our notion of cost, we need to define appropriate orderings to direct the search and select cut-off events. We consider the cases of additive and parallel costs separately.

Additive Cost

The case for optimal additive cost reachability analysis is quite simple. We define an adequate partial order on configurations $\prec_+$ which implies and is implied by the natural order defined by the additive cost function c_+. This order $\prec_+$ is used both for queueing and identifying cut-offs.

Here we first define an order on configurations based on their additive cost, then show it is an adequate order, and from this finally prove it can be used to make *ERV-Fly* find optimal additive cost solutions to REACHABILITY$_\Sigma$.

Definition 10 ($\prec_+$)**.**

$$C \prec_+ C' \Leftrightarrow c_+(C) < c_+(C')$$

Proposition 4.2.3 (Adequacy of $\prec_+$)**.** *The order on configurations defined by $\prec_+$ is adequate.*

Proof.

(a) *Well-founded.* For a proof by contradiction, let $C_1 \succ_+ C_2 \succ_+ \cdots$ be an infinite descending chain of *finite* configurations. Since the cost of each event is finite and positive, and the number of events in a configuration is finite, therefore $\infty > c_+(C_1) > c_+(C_2) > \cdots \geq 0$. Which is not possibly infinite.

(b) *Refines the subset operator.* $C_1 \subset C_2 \Rightarrow C_2 = C_1 \cup \mathcal{E}$ for some set of events $\mathcal{E}$. Consequently, $c_+(C_2) = \sum_{e \in C_1 \cup \mathcal{E}} c(e) = c_+(C_1) + c_+(\mathcal{E})$. Since each $c(e)$ is positive, $c_+(C_1) < c_+(C_2)$.

(c) *Preserved by finite extensions.* Let $C_1 \prec_+ C_2$ and $\text{Mark}(C_1) = \text{Mark}(C_2)$. Consider the configurations. $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$where $\mathcal{E}_1$ and $\mathcal{E}_2$ are finite sets of events such that $\mathcal{E}_1 \backsim \mathcal{E}_2$. By the definition of structural isomorphism, for every event in $\mathcal{E}_1$ there is an event in $\mathcal{E}_2$ φ-labelled by the same transition (and vice versa). Therefore $c_+(\mathcal{E}_1) = c_+(\mathcal{E}_2)$. We can thus make the succession of implications: $C_1 \prec_+ C_2 \Rightarrow c_+(C_1) < c_+(C_2') \Rightarrow c_+(C_1) + c(\mathcal{E}_1) < c_+(C_2') + c(\mathcal{E}_2) \Rightarrow c_+(C_1 \oplus \mathcal{E}_1) < c_+(C_2 \oplus \mathcal{E}_2) \Rightarrow C_1 \oplus \mathcal{E}_1 \prec_+ C_2 \oplus \mathcal{E}_2$.

□

Corollary 4.2.4 (Optimal Additive Cost)**.** *Let Σ_R define an instance of* REACHABILITY$_\Sigma$. *ERV-Fly($\Sigma_R, \prec_+$) solves* REACHABILITY$_\Sigma$*, and if positive identifies a minimum additive cost solution.*

Proof. Theorem 4.2.3 states $\prec_+$ is an adequate order. Thus by Theorem 4.2.1 *ERV*($\Sigma_R, \prec_+$) solves REACHABILITY$_\Sigma$. Furthermore, it satisfies the condition which guarantees a positive solution will be optimal with respect to $\prec_+$. To see this recall that property (b) of an adequate order states $[e] \subset [e'] \Rightarrow e \prec e'$. Considering Theorem 4.2.1, $e \in [e^*_R]$ thus implies $[e] \prec_+ [e^*_R]$, or $e = e^*_R$; furthermore since $[e^*_R] \prec_+ [e_R]$, and $\prec_+$ is clearly transitive, then $[e] \prec_+ [e_R]$. Thus *ERV-Fly*($\Sigma_R, \prec_+$) finds a minimal solution to REACHABILITY$_\Sigma$ with respect to $\prec_+$. By definition of $\prec_+$, this solution has minimum additive cost. □

We have now shown how unfolding can be directed on-the-fly to obtain a minimal additive cost solution to REACHABILITY$_\Sigma$.

Parallel Cost

The case of optimal parallel cost reachability analysis is not so forthright. We will step through the ideas and challenges that led to its development.

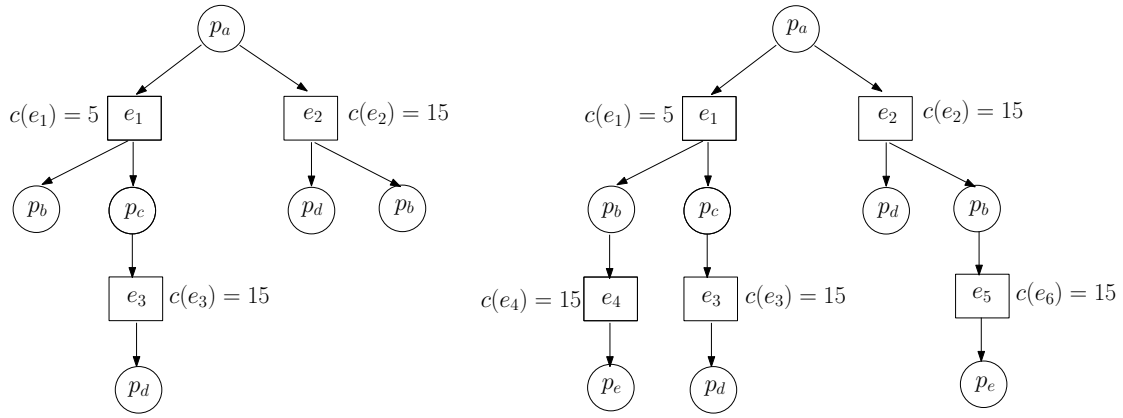


Figure 4.1: The branching process on the left has configurations $C_1 = \{e_1, e_3\}$ and $C_2 = \{e_2\}$, where Mark$(C_1) = \{a, b\} =$ Mark(C_2), $c_{||}(C_1) = 20$ and $c_{||}(C_2) = 15$. Thus $C_1 \succ_1 C_2$. It is then extended by events e_4 and e_5, each mapping to transition t such that ${}^\bullet t = b$, $t^\bullet = c$ and $c(t) = 15$, to become the branching process on the right. Consequently $c_{||}(C_1 \oplus e_4) = 20 \prec_1 c_{||}(C_2 \oplus e_5) = 30$. Clearly $\prec_1$ is not preserved by finite extensions

We want to construct a semi-adequate order on configurations which implies the natural order defined by the parallel cost function (but not necessarily the reverse). We naturally begin with an order based solely on $c_{||}$:

$$C \prec_1 C' \Leftrightarrow c_{||}(C) < c_{||}(C')$$

Unfortunately this order is not adequate, nor is it semi-adequate, as it violates condition (c) of Definition 7: $\prec_1$ is not preserved by finite extensions, thus does not guarantee a prefix generated by *ERV* $(\Sigma, \prec_1)$ will be complete. We show this by example. On the left of Figure 4.1 is a branching process for some PT-net system Σ. Let $C_1 = \{e_1, e_3\}$ and $C_2 = \{e_2\}$. Thus Mark$(C_1) = \{p_b, p_d\} =$ Mark(C_2). Furthermore, $c_{||}(C_1) = 20$ and $c_{||}(C_2) = 15$ so $C_1 \succ_1 C_2$. Now suppose there is a transition $t \in \Sigma$ such that ${}^\bullet t = p_b$, $t^\bullet = p_e$ and $c(t) = 15$. In the branching process on the right, C_1 and C_2 are extended by events e_4 and e_5 respectively, each corresponding to an instance of t (hence $e_4 \backsim e_5$). Consequently $c_{||}(C_1 \oplus e_4) = 20 \prec_1 c_{||}(C_2 \oplus e_5) = 30$. Thus the order is not preserved by finite extensions.

The nature of the inconsistency prompts us to reckon with the fact each place p in the final marking of C has its own parallel cost, corresponding to the maximal-cost chain leading to its assertion in the configuration C. In symbols:

$$c(C, p) \;=\; \max_{\sigma_p \in C} \sum_{e \in \sigma_p} c(e)$$

where the max is over all chains σ_p that *support* p. For example in Figure 4.1, considering

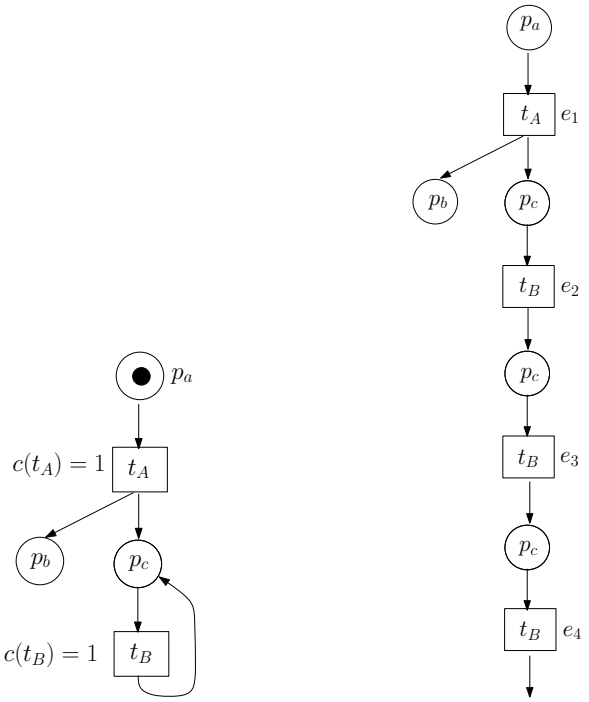


Figure 4.2: A PT-net system Σ (left) and the branching process generated by *ERV*$(\Sigma, \prec_1)$.

C_1 and C_2 as defined previously, $c(C_1, p_b) = 5, c(C_1, p_d) = 20, c(C_2, p_b) = 15, c(C_2, p_d) = 15$. Observe that $c_{||}(C) = \max\{c(C, p) : p \in \text{Mark}(C)\}$.

We subsequently consider a strict order on the costs of corresponding places in two configurations with the same final marking:

$$C \prec_2 C' \quad \text{iff} \quad \begin{cases} c(C,p) < c(C',p)) \; \forall p \in \text{Mark}(C) & \text{when Mark}(C) = \text{Mark}(C') \\ c_{||}(C) < c_{||}(C') & \text{otherwise.} \end{cases}$$

This order now satisfies condition (c), but can not ensure that in an infinite chain of configurations, $C_1 \subset C_2 \subset \cdots$ there will be two configurations such that $C_i \prec_2 C_j$ for some $i < j$ (condition (b) of Definition 7). Thus *ERV*$(\Sigma, \prec_2)$ may attempt to build an infinite prefix. To see this consider Figure 4.2, which shows a PT-net system Σ and the branching process that would be generated by *ERV*$(\Sigma, \prec_2)$. Let $C_1 = \{e_1\}, C_2 = \{e_1, e_2\}, C_3 = \{e_1, e_2, e_3\}, C_4 = \{e_1, e_2, e_3, e_4\}$, *etc.* Thus $C_1 \subset C_3 \subset C_3 \subset C_4 \subset \cdots$ is an infinite chain of configurations with the same final marking of $\{p_b, p_c\}$. Now $c(C_1, p_b) = c(C_2, p_b) = c(C_3, p_b) = c(C_4, p_b) = \cdots$ so it will never be the case that $c(C_i, p_b) < c(C_j, p_b)$, for some $i < j$, and thus it can not be that $C_i \prec_2 C_j$.

This new problem suggests we require a non-strict order, i.e. $c(C, p) \leq c(C', p) \; \forall p \in \text{Mark}(C)$ when $\text{Mark}(C) = \text{Mark}(C')$. However a non-strict order is not well-founded and thus violates condition (a) of Definition 7. This means a prefix generated by *ERV* may not be complete. This problem can be amended however, by imposing a further comparison

on conditions that *is* strict. It appears the ideal solution would be to require that at least one single strict comparison on the cost of two corresponding places must hold, i.e. when $\text{Mark}(C) = \text{Mark}(C')$ then $c(C,p) \leq c(C',p)) \; \forall p \in \text{Mark}(C)$ *and* $\exists p' \in \text{Mark}(C)$ such that $c(C,p') < c(C',p')$. While now satisfying (a), the existence of a single strict comparison is not preserved by finite extensions (condition (c) of Definition 7), suffering the same inconsistency identified previously for $\prec_1$. After much consideration we find ourselves forced to look to unrelated comparisons on configurations, to refine the non-strict order to one which is well-founded. We could use the Parikh vector or Foata normal form described in [59] for example. For simplicity, we use here the cardinality comparison.

The order is now:

$$C \prec_3 C' \quad \text{iff} \quad \begin{cases} c(C,p) \leq c(C',p)) \; \forall p \in \text{Mark}(C) & \\ \textit{and } |C| < |C'| & \text{when Mark}(C) = \text{Mark}(C') \\ & \\ c_{||}(C) < c_{||}(C') & \text{otherwise.} \end{cases}$$

This is an adequate order on configurations. The downfall of this solution is that cardinality does not imply parallel cost; this means there may be instances when two configurations have the same marking but one has greater parallel cost and the other has greater cardinality - so neither can be cut-off. To reduce the impact of this, we can enable a further comparison to be made when a strict ordering between place costs does in fact hold. The violation made by the strict comparison alone (i.e. as identified for $\prec_2$) is remedied by the non-strict case. We finally propose the following semi-adequate order on configurations:

Definition 11 ($\prec_{||}$)**.**

$$C \prec_{||} C' \quad \textit{iff} \quad \begin{cases} c(C,p) < c(C',p) \; \forall p \in \textit{Mark}(C) \textit{ or} & \\ & \\ c(C,p) \leq c(C',p) \; \forall p \in \textit{Mark}(C) & \\ \textit{and } |C| < |C'| & \textit{when Mark}(C) = \textit{Mark}(C') \\ & \\ c_{||}(C) < c_{||}(C') & \textit{otherwise.} \end{cases}$$

Proposition 4.2.5 (Semi-adequacy of $\prec_{||}$)**.** *The order on configurations defined by* $\prec_{||}$ *is semi-adequate.*

Proof.

(a) *Well-founded.* For a proof by contradiction, let $C_1 \succ_{||} C_2 \succ_{||} \cdots$ be an infinite descending chain of *finite* configurations. Since the number of markings is finite, there are only a finite number of subchains $C'_1 \succ_{||} C'_2 \succ_{||} \cdots$ such that $\text{Mark}(C'_1) = \text{Mark}(C'_j)$ for all $j > 0$. Each of these will consist of at least one of two possible subchains. The first possible chain contains configurations which *are not* strictly comparable based on individual place costs, i.e. $c(C''_1, p) \geq c(C''_2, p) \geq \cdots$. This implies $|C''_1| > |C''_2| > \cdots > 0$, which can not be an infinite chain because the configurations are finite. The second possible chain contains configurations which *are* strictly comparable based on individual place costs, i.e. $c(C'''_1, p) > c(C'''_2, p) > \cdots$. Since the cost of each event is finite and positive, and a configuration is finite, this implies $\infty > c_{||}(C'''_1) > c_{||}(C'''_2) > \cdots \geq 0$, which also can not be infinite.

(b) *Causal chains.* Let $C_1 \subset C_2 \subset \cdots$ be an infinite chain of finite configurations. Since the number of markings is finite, there is a subchain of configurations with the same final markings; let this be $C'_1 \subset C'_2 \subset \cdots$. Clearly $|C'_1| < |C'_2| < \cdots$. Furthermore, all chains in C'_1 must also be in C'_2, and all chains in C'_2 must also be in C'_3; thus $c(C'_1, p) \leq c(C'_2, p) \leq c(C'_3, p)$ for all $p \in \text{Mark}(C'_1)$. Therefore $C'_1 \prec_{||} C'_2 \prec_{||} \cdots$.

(c) *Preserved by finite extensions.* We examine the case when $C_1 \prec_{||} C_2$ and $\text{Mark}(C_1) = \text{Mark}(C_2)$. It follows from the definition of $\prec_{||}$ that either (1) $c(C_1, p) \leq c(C_2, p)$ for all $p \in \text{Mark}(C_1)$ *and* $|C_1| < |C_2|$; or (2) $c(C_1, p) < c(C_2, p)$ for all $p \in \text{Mark}(C_1)$. Consider the configurations $C'_1 = C_1 \oplus \mathcal{E}_1$ and $C'_2 = C_2 \oplus \mathcal{E}_2$ where $\mathcal{E}_1$ and $\mathcal{E}_2$ are structurally isomorphic extensions of unit length, which map to the transition t. With respect to C'_1, the cost of any place p' in the postset of t, i.e. $p' \in t^\bullet$, will be

$$c(C'_1, p') = \max_{p'' \in {}^\bullet t} c(C_1, p'') + c(\mathcal{E}_1)$$

Similarly for the same place in C'_2. Considering case (1), since $c(C_1, p'') \leq c(C_2, p'')$ for all $p'' \in \text{Mark}(C_1)$, and $c(\mathcal{E}_1) = c'(t) = c(\mathcal{E}_2)$, then $c(C'_1, p') \leq c(C'_2, p')$ for all $p' \in t^\bullet$. Places in $\text{Mark}(C_1 \oplus \mathcal{E}_1)$, but not in the postset of t, will have the same chains supporting them as in the original configurations, hence $c(C'_1, p) = c(C_1, p)$ for all $p \in \text{Mark}(C'_1) \setminus t^\bullet$ (and likewise for C'_2 and C_2). Thus $c(C'_1, p) \leq c(C'_2, p)$ for all $p \in \text{Mark}(C'_1)$. Clearly the difference between the cardinality of each configuration is preserved by these finite extensions. Hence $C'_1 \prec_{||} C'_2$.

In case (2), the cost of places in the postset of t is calculated as above. Then, since $c(C_1, p) < c(C_2, p)$ for all $p \in \text{Mark}(C_1)$, and $c(\mathcal{E}_1) = c'(t) = c(\mathcal{E}_2)$, it follows that $c(C'_1, p') < c(C'_2, p')$ for all $p' \in t^\bullet$. Places not in the postset of t are unaffected. Hence $C'_1 \prec_{||} C'_2$. We could extend these configurations again by another single

transition and follow the same reasoning, thus showing the order defined by $\prec_{||}$ is preserved by finite extensions.

□

Whilst this order prefers configurations with lower parallel cost, i.e. $C \prec_{||} C' \Rightarrow c_{||}(C) \leq c_{||}(C')$, it does not guarantee that if $c_{||}(C) < c_{||}(C')$ then $C \prec_{||} C'$. For example it could be that $[e_R^*]$ has a lower parallel cost than $[e_R]$ but the configurations are incomparable with respect to $\prec_{||}$ because they have the same final marking M and $c([e_R^*], p) > c([e_R], p)$ for some place $p \in M$ which does not contribute to the overall parallel cost of e_R^*. They will thus be ordered arbitrarily in the queue and we can not guarantee to remove $[e_R^*]$ before $[e_R]$. Furthermore, the comparisons made by this order are quite expensive, and we can make some simpler calculations for the purpose of ordering events in the queue. For example, if $\text{Mark}(C) = \text{Mark}(C')$ then determining that C' could never be smaller than C with respect to $\prec_{||}$, by observing that $c_{||}(C) < c_{||}(C')$, is easier than determining that C is definitely smaller than C' via place-wise comparison. With this in mind, in conjunction with Theorem 4.2.2, we propose a simple asymmetric order on configurations $\prec_{q||}$, where $C \prec_{||} C' \Rightarrow C \prec_{q||} C'$ and $c_{||}(C) < c_{||}(C') \Rightarrow C \prec_{q||} C'$.

Definition 12 ($\prec_{q||}$)**.**

$$C \prec_{q||} C' \quad \textit{iff} \quad \begin{cases} c_{||}(C) < c_{||}(C) \ \textit{or}, \\ \\ c_{||}(C) = c_{||}(C') \ \textit{and} \ |C| < |C'|. \end{cases}$$

Corollary 4.2.6 (Optimal Parallel Cost)**.** *Let Σ_R define an instance of* REACHABILITY$_\Sigma$*. ERV-Fly($\Sigma_R, \prec_{||}, \prec_{q||}$) solves* REACHABILITY$_\Sigma$*, and if positive identifies a minimum parallel cost solution.*

Proof. Theorem 4.2.5 states $\prec_{||}$ is a semi-adequate partial order on configurations. That $\prec_{q||}$ is an asymmetric partial order on configurations satisfying $C \prec_{||} C' \Rightarrow C \prec_{q||} C'$, is obvious from the definitions of $\prec_{||}$ and $\prec_{q||}$. Thus, by Theorem 4.2.2 *ERV-Fly*($\Sigma_R, \prec_{||}, \prec_{q||}$) solves REACHABILITY$_\Sigma$. Furthermore it satisfies those conditions which ensure it finds a minimal solution with respect to $c_{||}$:

(a) $e \in [e_R^*] \Rightarrow [e] \prec_{q||} [e_R]$, where $c_{||}([e_R^*]) < c_{||}([e_R])$. First observe that $[e] \subseteq [e_R^*] \Rightarrow c_{||}([e]) \leq c_{||}([e_R^*])$. This follows from the definition of $c_{||}$: all chains in $[e]$ must be in

$[e^*_R]$ since $[e] \subseteq [e^*_R]$, thus the maximum cost of any chain in $[e^*_R]$ must be at least the maximum cost of any chain in $[e]$. Since $c_{||}$ is clearly transitive, it thus follows that $c_{||}[e] < c_{||}([e_R])$. By definition of $\prec_{q||}$ this implies $[e] \prec_{q||} [e_R]$.

(b) $[e'] \prec_{||} [e]$ & $\text{Mark}(e') = \text{Mark}(e)$ & $[e] \oplus \mathcal{E} \oplus e_R = [e_R] \in \mathbf{e}_R \Rightarrow c_{||}([e'] \oplus \mathcal{E}' \oplus e'_R) \leq c_{||}([e] \oplus \mathcal{E} \oplus e_R)$ for $\mathcal{E}' \sim \mathcal{E}$. Since $\prec_{||}$ is a semi-adequate order, it is preserved by finite extensions. Thus $[e'] \prec_{||} [e]$ & $\text{Mark}(e') = \text{Mark}(e) \Rightarrow [e'] \oplus \mathcal{E}' \oplus e'_R = [e'_R] \prec_{||} [e] \oplus \mathcal{E} \oplus e_R = [e_R]$ for $\mathcal{E}' \sim \mathcal{E}$. By definition of $\prec_{||}$ this implies $c_{||}([e'_R]) \leq c_{||}([e_R])$.

Thus *ERV-Fly*$(\Sigma_R, \prec_{||}, \prec_{q||})$ solves REACHABILITY$_\Sigma$, and if positive identifies a minimum parallel cost solution.

□

Our solution for an order on configurations which enables *ERV-Fly* to optimise parallel cost is perhaps unnecessarily inefficient; as just mentioned, cardinality does not imply parallel cost. To make the best of the situation, one should try to use a refining order (i.e. here we have used cardinality) which *often* implies parallel cost for the given domain - if not always.

4.3 Directing the Unfolding with Heuristics

In the previous section, we sought sound strategies for directing *ERV-Fly* to build minimum-cost configurations first; this ensures that the first configuration to contain an event e_R is cost optimal. Further to this, we can favor those configurations which appear "closer" to including an event e_R, as their systematic exploration should result in a more efficient strategy for reachability analysis. Inspired by heuristic search in artificial intelligence, we incorporate problem specific information, in the form of a heuristic function, into strategies for directing the unfolding process. We are particularly interested in state based heuristics, which as mentioned have significantly impacted the scalability of automated planning. A heuristic is an estimate of the minimum cost, or shortest distance, from one particular state to another. For now, let us simply assume that a heuristic function h applied to a configuration C will approximate the minimum cost of extending C to include some event e_R.

As is standard in state based heuristic search, we seek strategies informed by the value of a function f that is composed of a cost function g and a heuristic function h:

$$f(C) = g(C) + h(C) \text{ where } C \text{ is a finite configuration.}$$

As before, let C_R denote a configuration containing an event e_R, such that $\varphi(e_R) = t_R$. The idea is that function f, applied to configuration C, estimates the *total cost* of a configuration

C_R where $C_R \supseteq C$, i.e. $f(C) \approx g(C_R)$. Technically, g and h are non-negative functions on configurations, such that $h(C) = 0$ if $t_R \in C$. Conceptually, the function g maps a configuration C to its cost and and the function h estimates the cost of extending C to some C_R, i.e. $h(C) \approx g(C_R) - g(C)$. We define the *optimal* heuristic function $h^*(C) = g(C_R) - g(C)$, where C_R is the minimal configuration with respect to g such that $C_R \supseteq C$. If no such configuration $C_R \supseteq C$ exists, then $h^*(C) = \infty$. We say that h is an *admissible* heuristic function if $h(C) \leq h^*(C)$ for all finite configurations C. That is, an admissible heuristic never over-estimates the cost of a solution to REACHABILITY$_\Sigma$. Admissible heuristics are required to guarantee optimality; when optimality is superfluous a non-admissible heuristic is more appropriate. Non-admissible heuristics generally provide better guidance than admissible heuristics, as will be shown by empirical results presented later in the chapter.

4.3.1 Direct Translation

A direct translation from standard heuristic search suggests the following order on configurations be used to direct the unfolding:

$$C \prec_4 C' \quad \text{iff} \quad \begin{cases} f(C) < f(C') & \text{or} \\ g(C) < g(C') & \text{if } f(C) = f(C'). \end{cases}$$

Notice that taking $g(C) = |C|$ and $h \equiv 0$ makes $\prec_4$ the adequate order used by the typical breadth-first implementation of the *ERV* algorithm. Furthermore, by breaking ties appropriately, $\prec_4$ becomes the total order defined in [59]. In addition, we could let $g \equiv c_+$ and require Mark$(C) =$ Mark$(C') \Rightarrow h(C) = h(C')$. The result would be a semi-adequate order on configurations which can be used to solve reachability and, when the heuristic function h is admissible, find an optimal solution with respect to c_+. Unfortunately however this framework is limited in the range of cost functions that can be soundly entailed. If we let $g \equiv c_{||}$, for example, the resulting order does not guarantee completeness. The reasoning follows easily from that given in Section 4.2.3, with respect to why the parallel cost function can not be used directly to define a semi-adequate order. We keep in mind however, that this order may be appropriate for queueing events.

4.3.2 Generic Framework for Heuristic Guidance

We prefer to construct a generic framework that can be instantiated with

(a) A cost function $g : C \to \mathbb{R}+$;

(b) A semi-adequate order $\prec_g$ such that $C \prec_g C \Rightarrow g(C) \leq g(C')$; and

(c) A heuristic function $h : C \to \{0\} \cup \mathbb{R}+$.

Then, providing g, $\prec_g$ and h satisfy certain conditions, can be used to solve the reachability problem optimally with respect to g.

With this in mind, in conjunction with Theorem 4.2.2, we propose a simple asymmetric order on configurations $\prec_f$, then identify specific conditions on the instantiation of g, h and $\prec_g$, which ensure *ERV-Fly*$(\Sigma_R, \prec_g, \prec_f)$ solves the reachability problem optimally with respect to g.

Definition 13 ($\prec_f$)**.** *Let $f(C) = g(C) + h(C)$ where $g : C \to \mathbb{R}+$ and $h : C \to \{0\} \cup \mathbb{R}+$.*

$$C \prec_f C' \quad \textit{iff} \quad \begin{cases} f(C) < f(C') & \textit{when } \textit{Mark}(C) \neq \textit{Mark}(C'); \\ g(C) < g(C') & \textit{when } \textit{Mark}(C) \neq \textit{Mark}(C') \textit{ and } f(C) = f(C'), \\ & \quad \textit{or } \textit{Mark}(C) = \textit{Mark}(C'); \\ |C| < |C'| & \textit{when } \textit{Mark}(C) \neq \textit{Mark}(C') \textit{ and } f(C) = f(C') \\ & \quad \textit{and } g(C) = g(C'), \textit{ or} \\ & \quad \textit{Mark}(C) = \textit{Mark}(C') \textit{ and } g(C) = g(C'). \end{cases}$$

Theorem 4.3.1 (Optimal REACHABILITY$_\Sigma$ wrt g, with heuristic guidance)**.** *Let Σ_R define an instance of* REACHABILITY$_\Sigma$*. Let $\prec_g$ be a semi-adequate partial order on configurations such that $C \prec_g C' \Rightarrow C \prec_f C'$. ERV-Fly$(\Sigma_R, \prec_g, \prec_f)$ will solve* REACHABILITY$_\Sigma$*. Furthermore, it will find a minimal solution with respect to g if*

(a) The heuristic function h is admissible[2]*, and $h(C) = 0$ if $e_R \in C$;*

(b) $e \in [e^*_R] \Rightarrow g([e]) \leq g([e^*_R])$, where $[e^*_R]$ is minimal among members of $\mathbf{e}_R$, with respect to g; and

(c) $[e] \prec_g [e'] \Rightarrow g([e]) \leq g([e'])$.

Proof. That $\prec_f$ is asymmetric is clear from its definition. Thus if $\prec_g$ is a semi-adequate partial order on configurations such that $C \prec_g C' \Rightarrow C \prec_f C'$, as supposed by this theorem,

[2]In order for h to be non-negative *and* admissible the cost function g must be increasing, i.e. $C \subseteq C' \Rightarrow g(C) \leq g(C')$.

then by Theorem 4.2.2, *ERV-Fly*$(\Sigma_R, \prec_g, \prec_f)$ solves REACHABILITY$_\Sigma$. Furthermore if g and h meet the conditions for optimality outlined above then *ERV-Fly*$(\Sigma_R, \prec_g, \prec_f)$ satisfies the conditions of Theorem 4.2.2 which ensure it finds a minimal solution with respect to g:

(a) $e \in [e^*_R] \Rightarrow [e] \prec_f [e_R]$, where $e^*_R, e_R \in \mathbf{e}_R$ and $g([e^*_R]) < g([e_R])$. By condition (b) of this theorem, $e \in [e^*_R] \Rightarrow g([e]) \leq g([e^*_R])$. Thus $g([e]) < g([e_R])$. Furthermore, since h is admissible (by condition (a)), $g([e]) + h([e]) = f([e]) \leq g([e^*_R])$. Also by condition (a), since $h(C) = 0$ when $e_R \in C$, so $g([e^*_R]) = f([e^*_R])$ and $g([e_R]) = f([e_R])$. Thus $f([e]) \leq f([e_R])$. Hence by definition of $\prec_f$, $[e] \prec_f [e_R]$ as required.

(b) $[e'] \prec_g [e]$ & Mark(e') = Mark(e) & $e \oplus \mathcal{E} \oplus e_R = [e_R] \in \mathbf{e}_R \Rightarrow g([e'] \oplus \mathcal{E}' \oplus e'_R) \leq g([e] \oplus \mathcal{E} \oplus e_R)$ for $\mathcal{E}' \sim \mathcal{E}$. Since $\prec_g$ is semi-adequate and thus preserved by finite extensions, $[e'] \prec_g [e]$ & Mark(e') = Mark$(e) \Rightarrow [e'] \oplus \mathcal{E}' \oplus e'_R = [e'_R] \prec_g [e] \oplus \mathcal{E} \oplus e_R = [e_R]$. By condition c this implies $g([e_R]) \leq g([e'_R])$, as required.

Thus *ERV-Fly*$(\Sigma_R, \prec_g, \prec_f)$ finds a minimal solution with respect to g, subject to the conditions of this theorem.

□

4.3.3 Specific Instantiations

We can now define heuristically informed strategies that direct the *ERV-Fly* algorithm to solve REACHABILITY$_\Sigma$ optimally with respect to a particular cost function. Applying this framework to additive and parallel cost problems is straight forward.

We relax the orders $\prec_+$ and $\prec_{||}$ so that they are only defined for configurations with equal markings. These orders are semi-adequate, as a closer look at the conditions for semi-adequacy reveals they need only to be met by configurations with equal markings. The relaxation enables more freedom in the ordering of configurations with non-equal markings; in particular we can consider their heuristic value, even if the heuristic function is not admissible, and thus solve reachability problems more efficiently. If the heuristic is admissible, we can solve reachability problems optimally with respect to the additive or parallel cost of a solution.

Heuristic Guidance with Additive Cost Function

We first define a semi-adequate order on configurations $\prec_{f+}$ such that $C \prec_{f+} C' \Rightarrow C \prec_f C'$ when g is instantiated with the the additive cost function, i.e. $g \equiv c_+$. It then follows

that *ERV-Fly*($\Sigma_R, \prec_{f+}, \prec_f$) solves the reachability problem defined by Σ_R and furthermore finds an optimal solution with respect to additive cost subject to certain conditions on the heuristic function.

Definition 14 ($\prec_{f+}$)**.**

$$C \prec_{f+} C' \Leftrightarrow c_+(C) < c_+(C') \ \ \& \ \ \mathit{Mark}(C) = \mathit{Mark}(C')$$

Proposition 4.3.2 (Semi-adequacy of $\prec_{f+}$)**.** *The partial order on configurations defined by $\prec_{f+}$ is semi-adequate.*

Proof. The proof that $\prec_+$ is well founded and preserved by finite extensions holds also for $\prec_{f+}$ (see parts a and c of the proof for Proposition 4.2.3). It remains to show that in any infinite causal chain $C_1 \subset C_2 \subset \cdots$ there exists $i < j$ such that $C_i \prec_{f+} C_k$. Let $C_1 \subset C_2 \subset \cdots$ be an infinite chain of finite configurations. Since the number of markings is finite, there is a subchain of configurations with the same final markings; let this be $C'_1 \subset C'_2 \subset \cdots$. C'_2 contains all the events in C'_1, and at least one more. Thus, since the cost of each event is positive, $c_+(C'_1) < c_+C_2$. Therefore, since the markings are equal, $C'_1 \prec_{f+} C'_2$. Thus $\prec_{f+}$ is semi-adequate. □

Corollary 4.3.3. *Considering the definition of $\prec_f$, let $g \equiv c_+$. Let Σ_R define an instance of* REACHABILITY$_\Sigma$. *ERV-Fly($\Sigma_R, \prec_{f+}, \prec_f$) solves* REACHABILITY$_\Sigma$ *and furthermore finds an optimal solution with respect to c_+ if the heuristic function h is admissible and $h(C) = 0$ if $e_R \in C$.*

Proof. Proposition 4.3.2 states that $\prec_{f+}$ is semi-adequate. Let $g \equiv c_+$ in the definition of $\prec_f$. That $C \prec_{f+} C' \Rightarrow C \prec_f C'$ is clear from the definitions of $\prec_{f+}$ and $\prec_f$. Thus by Theorem 4.3.1 *ERV-Fly*($\Sigma_R, \prec_{f+}, \prec_f$) solves REACHABILITY$_\Sigma$. Furthermore, it satisfies the conditions (b) and (c) of Theorem 4.3.1:

- (b) $e \in [e^*_R] \Rightarrow c_+([e]) \leq c_+([e^*_R])$, where $[e^*_R]$ is minimal among members of $\mathbf{e}_R$, with respect to c_+. This follows from the definition of the additive cost function and the fact every event in $[e]$ must also be in $[e^*_R]$ (in fact the inequality will be strict except when e is e^*_R).
- (c) $[e] \prec_{f+} [e'] \Rightarrow c_+([e]) \leq c_+([e'])$. This is a direct consequence of the definition of $\prec_{f+}$ (in fact the inequality will be strict in this case).

Thus, if h is admissible and $h(C) = 0$ when $e_R \in C$ (condition a of Theorem 4.3.1) then *ERV-Fly*($\Sigma_R, \prec_{f+}, \prec_f$) (with $g \equiv c_+$) will find a minimal solution with respect to the additive cost function. □

This strategy is particularly useful when optimality is superfluous and the priority is to solve a reachability problem as quickly as possible. In this case, one should simply set the cost of every transition to 1 (i.e. $c_+(C) = |C|$) and use and an informative non-admissible heuristic.

Heuristic Guidance with Parallel Cost Function

We now define a semi-adequate order on configurations $\prec_{f||}$ such that $C \prec_{f||} C' \Rightarrow C \prec_f C'$ when g is instantiated with the the parallel cost function, i.e. $g \equiv c_{||}$. It then follows that *ERV-Fly*($\Sigma_R, \prec_{f||}, \prec_f$) solves the reachability problem defined by Σ_R and furthermore finds an optimal solution with respect to parallel cost if the heuristic function is admissible.

Definition 15 ($\prec_{f||}$)**.**

$$C \prec_{f||} C' \quad \textit{iff} \quad \begin{cases} c(C,p) < c(C',p)\ \forall p \in \textit{Mark}(C)\ \textit{or} & \\ c(C,p) \leq c(C',p)\ \forall p \in \textit{Mark}(C) & \\ \textit{and}\ |C| < |C'| & \textit{when Mark}(C) = \textit{Mark}(C') \end{cases}$$

Proposition 4.3.4 (Semi-adequacy of $\prec_{f||}$)**.** *The partial order on configurations defined by $\prec_{||}$ is semi-adequate.*

Proof. The proof that $\prec_{||}$ is semi-adequate holds also for $\prec_{f||}$ (see that parts (a), (b) and (c) of the proof for Proposition 4.2.5 depend only on the case when markings are equal). □

Corollary 4.3.5. *Considering the definition of $\prec_f$, let $g \equiv c_{||}$. Let Σ_R define an instance of* REACHABILITY$_\Sigma$. ERV-Fly*($\Sigma_R, \prec_{f||}, \prec_f$) solves* REACHABILITY$_\Sigma$ *and furthermore finds an optimal solution with respect to $c_{||}$ if the heuristic function h is admissible and $h(C) = 0$ if $e_R \in C$.*

Proof. Proposition 4.3.4 states that $\prec_{f||}$ is semi-adequate. Let $g \equiv c_{||}$ in the definition of $\prec_f$. That $C \prec_{f||} C' \Rightarrow C \prec_f C'$ is clear from the definitions of $\prec_{f||}$ and $\prec_f$. Thus by Theorem 4.3.1 *ERV-Fly*($\Sigma_R, \prec_{f||}, \prec_f$) solves REACHABILITY$_\Sigma$. Furthermore, it satisfies conditions (b) and (c) of this same theorem:

(b) $e \in [e^*_R] \Rightarrow c_{||}([e]) \leq c_{||}([e^*_R])$, where $[e^*_R]$ is minimal among members of $\mathbf{e}_R$, with respect to $c_{||}$. This follows from the definition of the parallel cost function and the fact every chain in $[e]$ must also be in $[e^*_R]$ (in fact the inequality will be strict, except when $e = e^*_R$, as all chains can otherwise be extended by e^*_R).

(c) $[e] \prec_{f||} [e'] \Rightarrow c_{||}([e]) \leq c_{||}([e'])$. This is a direct consequence of the definition of $\prec_{f||}$

Thus, if h is admissible and $h(C) = 0$ when $e_R \in C$ (condition (a) of Theorem 4.3.1) then *ERV-Fly*$(\Sigma_R, \prec_{f||}, \prec_f)$ with $g \equiv c_{||}$ will find a minimal solution with respect to the parallel cost function. □

The conditions on h allow it to depend on a configuration C, rather than just is marking Mark(C). It will be shown that this is particularly useful when crafting a heuristic function based on the parallel cost of a configuration.

Summary

Whilst focusing on additive and parallel cost criteria, we sought a general formulation for a heuristically informed semi-adequate order on configurations that is amenable to use with other cost functions one may deem useful. We emphasise that *directed unfolding* can:

(a) Be used to find an optimal solution to REACHABILITY$_\Sigma$;

(b) Be incorporated with an admissible heuristic function to find an optimal solution more quickly; and

(c) Employ non-admissible heuristics, when optimality is superfluous, to solve positive reachability problems more efficiently.

Clearly the benefits of these applications are concentrated on positive solutions to the reachability problem. This then brings us to consider the effect of using directed unfolding when the solution to REACHABILITY$_\Sigma$ is negative.

4.4 Size of the Finite Prefix

When the solution to REACHABILITY$_\Sigma$ is positive, then the efficiency of a heuristically guided strategy depends on how informative the heuristic function is. When the solution is negative, then efficiency depends on the size of the finite prefix that must be generated to conclude no reachable marking can enable t_R.

The breadth-first order on configurations proposed by Esparza *et al* [59], based on McMillan's partial order comparing cardinality and refined to a total order using Parikh-vector

and Foata normal form comparisons, is considered to produce a *minimal* complete finite prefix when applied with the *ERV* algorithm. Whilst no formal definition of minimality is provided, it appears to suggest that the prefix is as small as possible whilst maintaining completeness.

The heuristically guided strategies presented here are not total orders, and furthermore do not necessarily refine the subset operator. When a total order on configurations is used to implement the *ERV* algorithm, then the number of non cut-off events, in the generated prefix, is bounded by the number of reachable markings. Alternatively, a partial order implies there may be instances in which two configurations have the same final marking but neither can be identified as minimal and subsequently serve as a cut-off. In these situations, i.e. a partial order on configurations, if the order refines the subset operator then we can at least bound the length of a causal chain to the number of reachable markings. Thus, the absence of a total order and further failure to refine the subset operator suggests the heuristically guided strategies presented in this thesis will be less efficient than the original breadth-first approach, when applied to reachability problems for which the solution is negative. However experimental results, presented in Section 4.6, show this is not the case.

The reason for this comes back to the unique considerations we can make when generating a prefix for the purpose of reachability analysis. Namely, in the same way it is not necessary to generate a complete prefix in order to identify a positive solution, it is also not necessary for the conclusion of a negative solution. If we take an event e from the queue, and can conclude that there is no configuration C_R such that $C_R \supseteq [e]$ (where $e_R \in C_R$ and $\varphi(e_R) = t_R$), then it is unnecessary to extend $[e]$ any further. That is, for the purposes of reachability analysis e can be a cut-off event. In the context of a heuristically guided strategy, this is implied by $h([e]) = \infty$ (and thus $f([e]) = \infty$), when h is a *safely pruning* heuristic. A heuristic function is safely pruning [67] if and only if $h(C) = \infty$ implies that there is no configuration $C_R \supseteq C$ with $e_R \in C_R$ (i.e. $h^*(C) = \infty$). Pruning safety is a weaker property than admissibility as it pertains only to a subset of configurations (namely the "dead-end" configurations). All the heuristic functions considered in this thesis are safely pruning. Furthermore, the ordering of the queue in these strategies tells us that all other events in the queue must also have a heuristic value of infinity: thus we can cease unfolding completely when an event e with $f([e]) = \infty$ is dequeued, and conclude that the solution to REACHABILITY$_\Sigma$ is negative.

We illustrate the behaviour of a safely pruning heuristic on the small example in Figure 4.3. A heuristic such as $h^{\max}$ (see next section) estimates that from the initial marking $\{p_a\}$, a marking which includes the places in ${}^\bullet t_R = \{p_d, p_e\}$ is reachable in 2 steps (the maximum

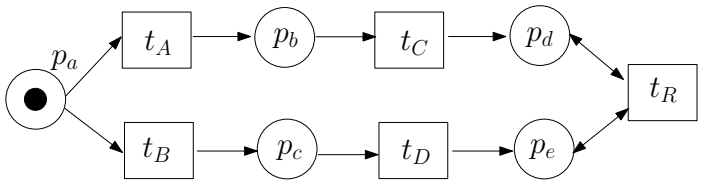


Figure 4.3: PT-net system where transition t_R can not be enabled.

cost of the two paths, assuming all transitions have unit cost). An event mapping to transition t_A (or t_B) is then added to the branching process, leading to a configuration with final marking p_b (or p_c). The $h^{\max}$ value of the marking p_b (or p_c) is ∞, since there is then no way to reach one of the places in ${}^\bullet t_R$, namely p_e (or p_d).

To summarise, when solving REACHABILITY$_\Sigma$ on-the-fly via unfolding, the size of the prefix generated by directed unfolding is generally related to the informativeness of the guiding heuristic. However, in the particular case when REACHABILITY$_\Sigma$ is negative the size of the prefix is determined by the pruning power of the heuristic, i.e. its ability to quickly recognise dead-end configurations. As mentioned in Chapter 7 future work may involve a more informed, quantitative complexity analysis of unfolding in general and the size of prefixes required for on-the-fly reachability analysis in particular.

4.5 Heuristic Functions

A common approach to constructing heuristic functions, both admissible and inadmissible, is to define a *relaxation* of the problem, such that the relaxed problem can be solved (at least approximately) efficiently [132]. That is, a less complex problem is obtained from the original one by making simplifying assumptions and relaxing constraints. The cost of a solution to the relaxed problem can be used as an estimate of the cost of the solution to the real problem, i.e. as the heuristic value [132].

To implement $\prec_{f_+}$ or $\prec_{f_{||}}$ in the *ERV-Fly* algorithm, we need heuristic functions that estimate the minimum cost, with respect to c_+ and $c_{||}$, of extending a configuration C to a configuration $C_R \supseteq C$, such that $e_R \in C_R$ and $\varphi(e_R) = t_R$. In the context of the original Petri net, this is equivalent to the problem of determining the the minimum cost of an occurrence poset $\sigma_\triangleright$ satisfying $\text{Mark}(C) \xrightarrow{\sigma_\triangleright} M_R$ where ${}^\bullet t_R \subseteq M_R$. The additive and parallel cost functions defined previously over configurations can be applied equivalently to occurrence posets. The additive cost of an occurrence poset $\sigma_\triangleright = \langle T', \triangleright \rangle$ is the sum of the costs of the transitions in T'. A transition t occurring in T' n times will contribute $n \times c(t)$ to the

additive cost. The parallel cost of $\sigma_{\triangleright}$ is the maximum sum of the costs of transitions in a chain defined by the ordering constraints, e.g $c(t_j)+c(t_k)+\ldots+c(t_m)$ where $t_j \triangleright t_k \ldots \triangleright t_m$.

This thesis looks at relaxing the problem in the context of the original Petri net, to construct heuristic functions. To simplify discussion we will avoid reference to the occurrence poset and speak of a heuristic estimating the cost from one marking to another. We now present heuristic functions which consider additive and parallel cost functions (respectively).

4.5.1 Heuristic Functions For Additive Cost

We have experimented with heuristic functions derived from two different relaxations, both developed in the area of AI planning. The first relaxation is to consider each place in the preset of a transition independently of the others. For a transition t to be enabled, each place in ${}^\bullet t$ must be marked. Thus, the cost from a given marking M to a marking that enables t is *at least* the cost from M to mark a single place in ${}^\bullet t$. So, a lower bound on the cost from marking M to a marking enabling t is $d^{\max}(M, {}^\bullet t) = \max_{p \in {}^\bullet t} d^{\max}(M, \{p\})$, where $d^{\max}(M, \{p\})$ denotes the cost from M to any marking that includes $\{p\}$. For a place p to be included in the target marking M – if it is not already – at least one transition in ${}^\bullet p$ must occur. Thus, $d^{\max}(M, \{p\}) = \min_{t \in {}^\bullet p} d^{\max}(M, {}^\bullet t) + c'(t)$. Whilst this defines the exact cost from M to a marking including $\{p\}$, it is an under-estimate since our valuation of $d^{\max}(M, {}^\bullet t)$ is a lower bound. Combining these two facts we obtain an equation $d^{\max}(M, M')$ which specifies a lower bound on the additive cost from a marking M to M'. The solution to this equation can be computed in polynomial time using dynamic programming [67]. We use this to define a heuristic function, denoted by $h^{\max}$, which is equivalent to the h_{max} heuristic considered in [18, 19]:

Definition 16.

$$h^{max}(C) = d^{max}(\mathit{Mark}(C), {}^\bullet t_R)$$

where

$$d^{max}(M, M') = \begin{cases} 0 & \textit{if } M' \subseteq M \\ \min_{t \in {}^\bullet p} c'(t) + d^{max}(M, {}^\bullet t) & \textit{if } M' = \{p\} \\ \max_{p \in M'} d^{max}(M, \{p\}) & \textit{otherwise.} \end{cases} \tag{4.1}$$

Note that $h^{\max}$ is also equivalent to the h^m heuristic, defined in [76], when $m = 1$.

Since $d^{\max}(\text{Mark}(C), {}^\bullet t_R)$ is a lower bound on the minimum additive cost of extending a configuration C to a configuration $C_R \supseteq C$, such that $e_R \in C_R$, the $h^{\max}$ heuristic is admissible.

In many cases, however, $h^{\max}$ is too weak to effectively guide the unfolding process. To avoid over-estimating costs, admissible heuristics tend to be conservative and consequently less discriminating between different states. Conversely inadmissible heuristics have greater freedom in assigning values are are thus more informative with respect to the relative values being a stronger indicator of how "promising" a path is. An inadmissible, but often more informative version of the $h^{\max}$ heuristic, can be obtained by estimating the cost from marking M to a marking enabling t as the *sum* (as opposed to the maximum) of the costs to mark each place in ${}^\bullet t$ independently. The resulting heuristic, which we will refer to as h^{sum}, is based on the h_{add} heuristic defined in [18, 19]:

Definition 17.

$$h^{sum}(C) = d^{sum}(\mathit{Mark}(C), {}^\bullet t_R)$$

where

$$d^{sum}(M, M') = \begin{cases} 0 & \text{if } M' \subseteq M \\ \min_{t \in {}^\bullet p} c'(t) + d^{sum}(M, {}^\bullet t) & \text{if } M' = \{p\} \\ \sum_{p \in M'} d^{sum}(M, \{p\}) & \text{otherwise.} \end{cases} \tag{4.2}$$

The second relaxation condered in this thesis is known as the *delete relaxation*. In the context of Petri nets, the simplifying assumption made in this relaxation is that a transition requires the presence of a token in each place in its preset to be enabled, but does not consume any of these tokens when it occurs. This means that once a place is marked it will remain so. Every marking that is reachable in the original net is a subset of a marking that is reachable in the relaxed problem. The delete-relaxed problem has the property that a solution (if one exists) can be found in polynomial time [86]. The heuristic function h^{FF} (named after FF [86], the first planning system to employ this heuristic) maps a configuration C to the cost of extending it to contain e_R when the delete relaxation is employed. The relaxed problem is solved via its *relaxed planning graph* representation, which is essentially a complete prefix of the unfolding of the relaxed problem, seeded by conditions mapping to the final marking of the configuration in question without considering forward

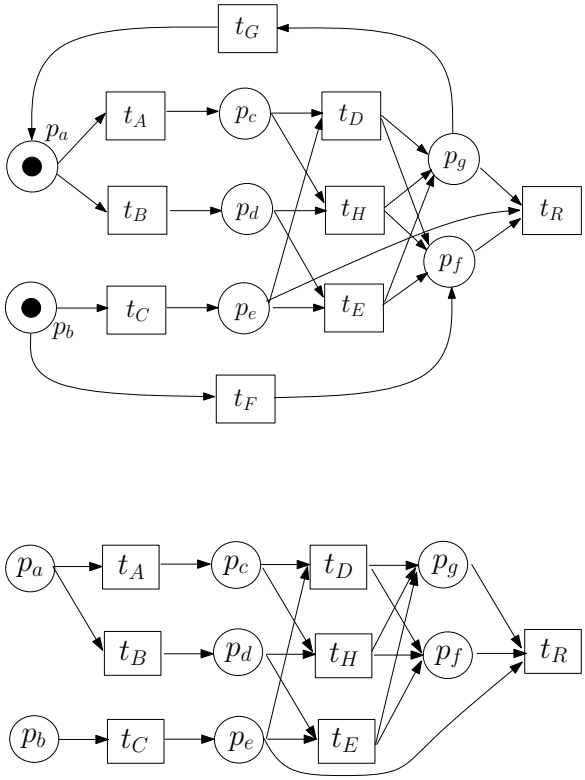


Figure 4.4: Marked PT-net (above) and corresponding relaxed planning graph.

conflict[3]. A description of the construction of a planning graph is given in Chapter 5; the delete-relaxation version of this is obtained by simply ignoring all the negative effects of actions (to be defined in the same chapter). The cost of a path to t_R can be easily extracted from the relaxed planning graph; in the case there are multiple transitions feeding a place, one is chosen arbitrarily. Figure 4.4 shows a marked PT-net and the relaxed planning graph corresponding to this marking: assuming all actions have unit cost a minimum cost solution for the relaxed problem is the sequence t_A, t_C, t_D; another solution is t_A, t_B, t_H, t_C.

The h^{FF} heuristic satisfies the requirement that $h^{\text{FF}}(C) = 0$ when $e_R \in C$, but because an arbitrary solution is extracted it is not admissible. The heuristic defined by the cost of the *minimal* solution to the delete-relaxed problem, known as h^+, is admissible, but solving the relaxed problem optimally is NP-hard [31].

[3]As a consequence the delete relaxation can technically destroy the 1-safeness of the net. However, since tokens are never consumed the exact number of tokens in a place does not matter, but only whether the place is marked or not.

4.5.2 Heuristic Function For Parallel Cost

A heuristic $h(C)$ that reflects the parallel cost function, must estimate the cost of reaching t_R from Mark(C) while considering the *relative cost* to mark each of the places $p \in$ Mark(C) via configuration C. We thus define the relative cost of place p with respect to configuration C as $t(C, p)$:

$$t(C, p) = c(C, p) - c_{||}(C)$$

Observe that these quantities are always less than or equal to zero.

We consider a temporal version of the $h^{\max}$ heuristic, based on the temporal h^m_T heuristic described by Haslum and Geffner in [77] with $m = 1$:

Definition 18.

$$h^{par}(C) \ = \max\{0, d^{par}(C, t_R{}^\bullet)\}$$

where

$$d^{par}(C, M') \ = \begin{cases} \max_{p \in M'} t(C, p) & if M' \subseteq \mathit{Mark}(C) \\ \min_{t \in {}^\bullet p} c'(t) + d^{par}(M, {}^\bullet t) & M' = \{p\},\ p \notin M \\ \max_{p \in M'} d^{par}(M, \{p\}) & \mathit{otherwise.} \end{cases} \tag{4.3}$$

h^{par} is solved by means of dynamic programming seeded at the relative place times.

4.6 Experimental Results

We extended MOLE to implement:

- ◇ *ERV-Fly*$(\Sigma_R, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and admissible heuristics $h \equiv 0$ and $h^{\max}$, and inadmissible heuristics h^{FF} and h^{sum};
- ◇ *ERV-Fly*$(\Sigma_R, \prec_{f||}, \prec_f)$, with $g \equiv c_{||}$ and admissible heuristics $h \equiv 0$ and h^{par}.

Here we compare directed unfolding and "blind" unfolding, in the context of solving REACHABILITY$_\Sigma$ using the *ERV-Fly* algorithm. Clearly directed unfolding provides an additional guarantee on the quality of a positive solution, when considering the additive and parallel cost functions for configurations. Conversely, what we wish to illustrate here is the benefit of applying heuristically informed strategies to problems that can (theoretically) be solved

using a breadth-first strategy. Consequently we limit the scope to additive cost problems, where the cost of every transition is 1. The results demonstrate that directed unfolding clearly solves reachability problems more quickly.

For the purpose of on-the-fly reachability analysis, we found that the additional tie-breaking comparisons used by MOLE to achieve a *total* order on configurations inhibited all versions (including the original). We consequently disabled the Parikh vector and Foata normal form based comparisons [59] to obtain the results which follow[4]. All experiments in this chapter were conducted on a Pentium M 1.7 GHz with 1GB of memory.

4.6.1 Petri Net Benchmarks

The developers of MOLE provided a set of standard Petri net benchmarks representative of Corbett's examples [42]. Recall that for on-the-fly reachability analysis via unfolding, REACHABILITY$_\Sigma$ is cast as the problem of identifying an occurrence sequence that enables t_R. Thus to assess the performance of directed unfolding we systematically considered each transition in each benchmark to be t_R. DARTES, a PT-net system modelling the communication skeleton of an Ada program, was the only benchmark which proved to be challenging. MOLE is unable to decide, in reasonable time, whether certain transitions in DARTES can be enabled. For the other benchmarks, MOLE can generate a complete finite prefix of the unfolding of the Petri net model in a matter of seconds.

Figure 4.5 compares the performance of the original version of MOLE with directed versions employing each of the heuristic functions $h^{\max}$, h^{sum}, and h^{FF}. For each version, and for each of the 253 DARTES transitions, we recorded the time taken to decide whether this transition could be enabled. The graph shows the percentage of problems solved by each version, within computation time limits ranging from 0.01 to 300 seconds. The original breadth-first version of MOLE performs well on the simplest problems - 50% of the problems are solved within 0.1 seconds. In particular, within a 0.01 second time limit the original MOLE solves slightly more problems than the versions employing the $h^{\max}$ and h^{sum} heuristics - here the overhead in computing the heuristic values outweighs the benefit - however is clearly less efficient than the version employing the h^{FF} heuristic. Within the 0.03 to 300 second time limits, the breadth-first strategy is systematically outperformed by all of the directed versions. The benefit of the heuristically informed strategies become exceptionally clear for larger problems: given a 300 second time limit the blind version

[4]This means the original version of MOLE implements McMillan's cardinality order on configurations only [119].

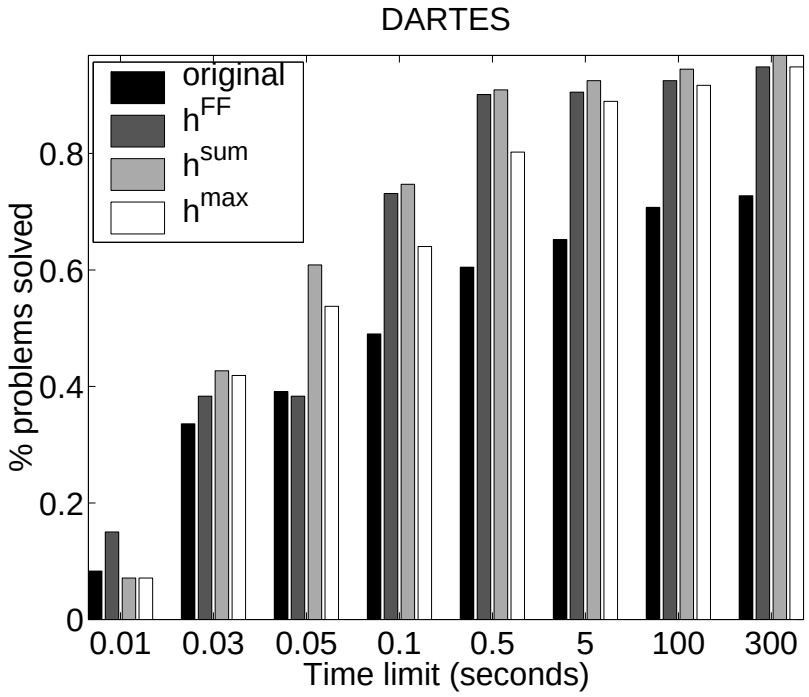


Figure 4.5: Results for DARTES

solve just under 75% of the problems whilst the directed versions solve just over 95% of them. DARTES contains transitions which can only be enabled by occurence sequences of over 90 transitions. Within the given time-limit, the breadth-first strategy could only solve problems where a positive solution contains less than 60 transitions. Specifically, within the time limitations, the original version solves 185 of the 253 problem instances (73%), whereas the version using h^{sum} solves 245 (97%). The instances solved by each of the directed versions is a strict superset of those solved by the original. Unsurprisingly, all solved problems were positive solutions (the transitions could be enabled).

4.6.2 Random Problems

To investigate the scalability of directed unfolding, we first turned to the problem generators provided with the PEP distribution[5] (bufgen, pargen, philgen, phillgen, slotgen). Unfortunately, these failed to provide challenging benchmarks for unfolding based reachability analysis, and rather appeared to be designed to illustrate its strength over state space analysis. MOLE can construct the finite prefix for very large instances due to a massive degree of concurrency. More importantly, the length of paths to any transition is very short (2 for philgen to 12 for slotgen) and does not increase with problem size.

[5] http://parsys.informatik.uni-oldenburg.de/ pep/PEP2.0/

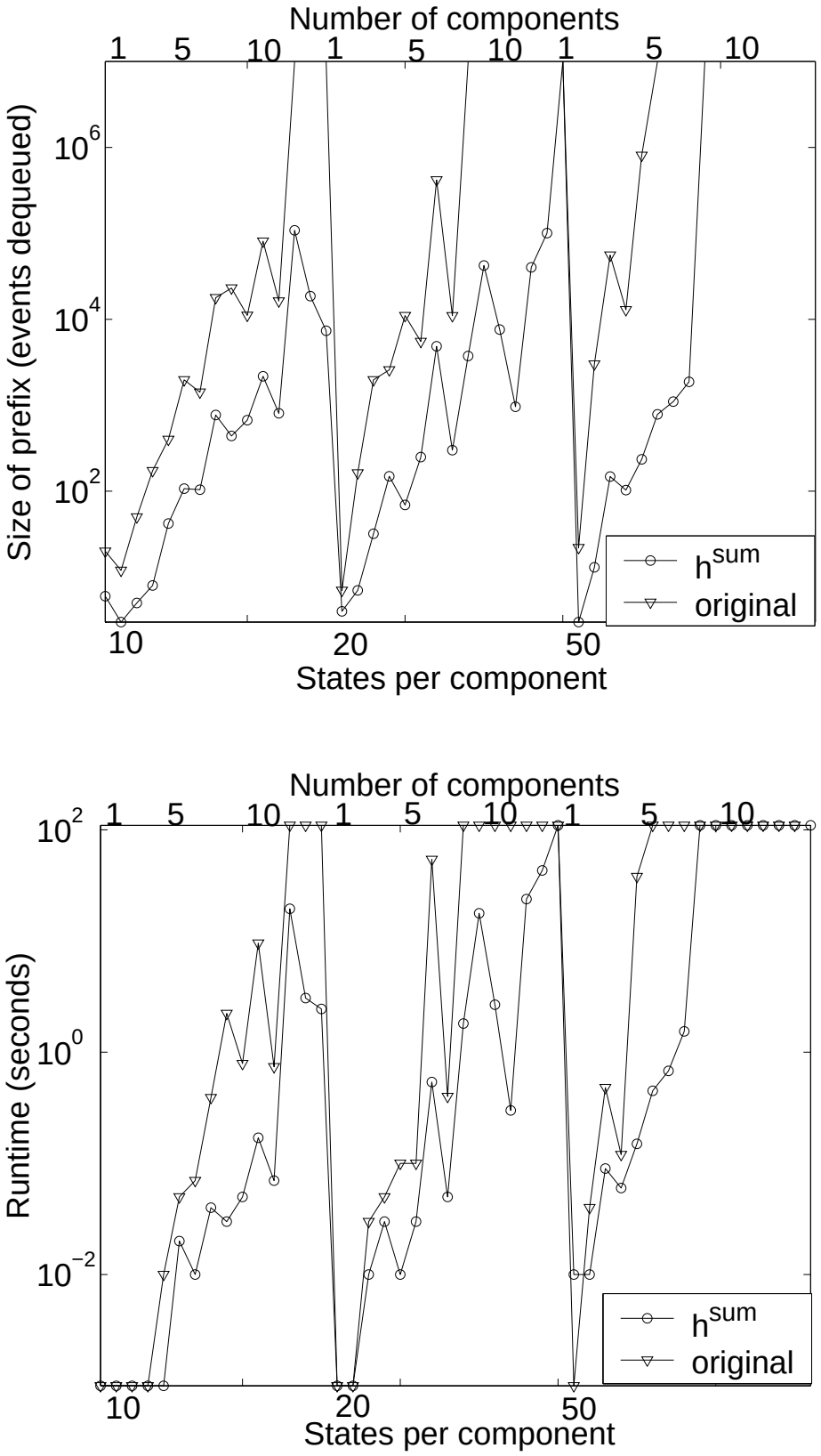
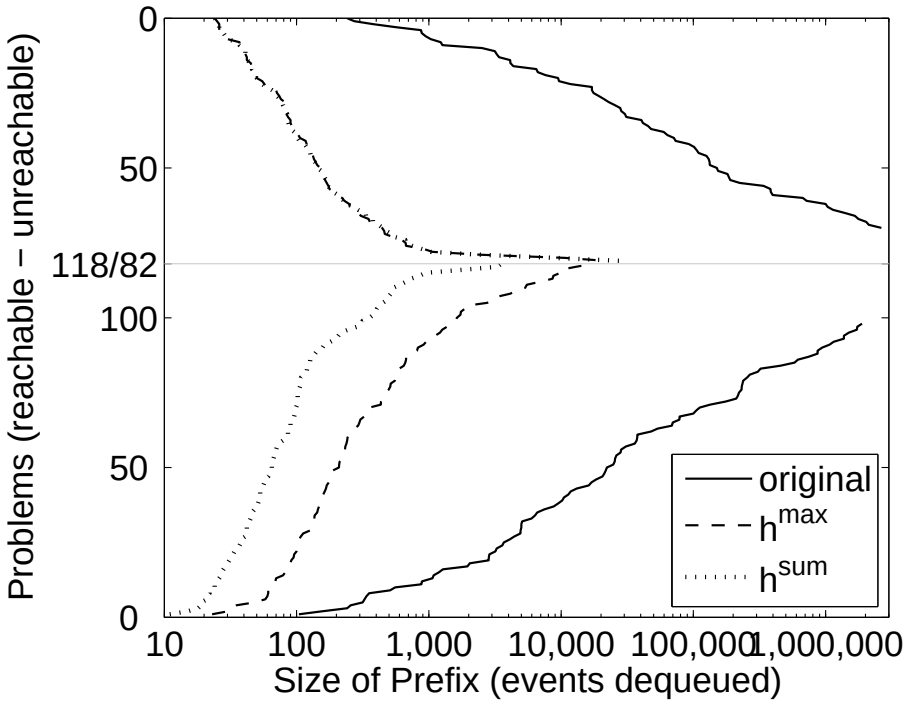


Figure 4.6: Results for Random PT-nets



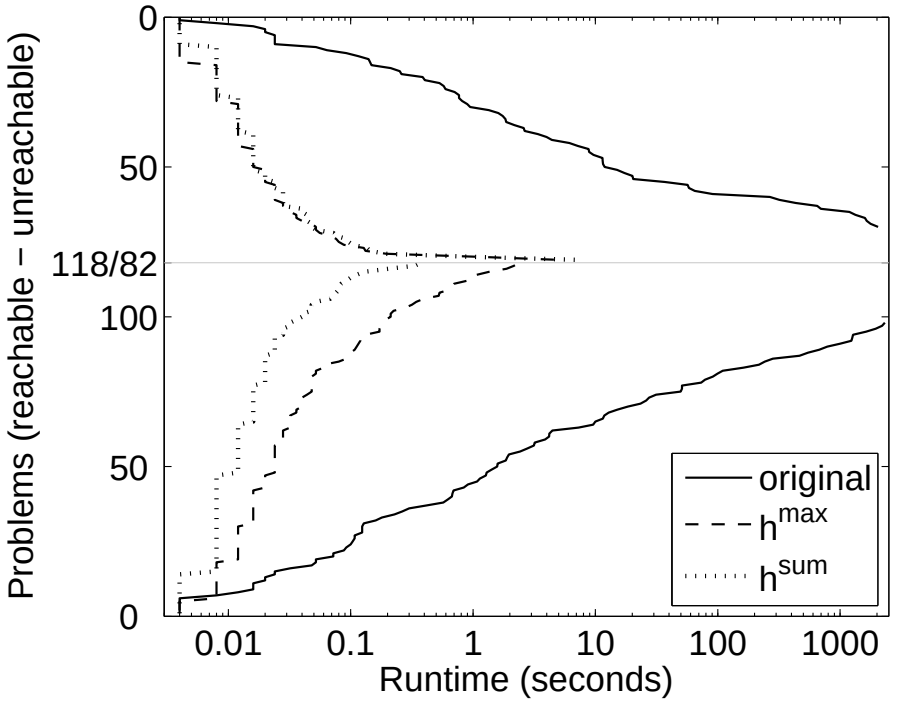


Figure 4.7: Results for Random PT-nets: reachable versus unreachable problems

We consequently decided to implement our own generator of random Petri nets. Conceptually, the generator creates a set of component automata, and connects them in an acyclic dependency network. The transition graph of each component automaton is a sparse, but strongly connected, random directed graph. Synchronisations between pairs of component automata are such that only one (the dependent) automaton involved in the synchronisation changes state, but can only do so when the other component automaton is in a particular state. Synchronisations are chosen randomly, in such a way that dependencies between components form a tree. Reachability problems are defined by the random and independent choice of states for the various automata. The construction of the component automata and their synchronisations ensures that every choice of states is reachable. We generated random problems featuring $1 \ldots 15$ component automata of 10, 20, and 50 states each. The resulting Petri nets range from 10 places and 30 transitions to 750 places and over 4000 transitions.

The bottom graph in Figure 4.6 shows the runtime for each experiment; the top graph shows the number of events taken from the queue of possible extensions. To avoid cluttering these graphs, we only show the performance of the worst and best strategies, namely breadth-first (i.e. original), and *ERV-Fly*($\Sigma_R, \prec_{f+}, \prec_f$), implemented with $g \equiv c_+$ and $h \equiv h^{\text{sum}}$. Evidently, directed unfolding can solve much larger problems than the original blind unfolding. For the largest problem instances considered, the difference in performance reached over two orders of magnitude in speed and three in size. The original version could merely solve the easier half of the problems, while directed unfolding only failed on the six larger instances (with 50 states per component).

In this experiment, optimal solutions reach lengths of several hundred events. For the problems which could be solved optimally using the h^{max} implementation, h^{FF} produced solutions within a couple transitions of the optimal. In all the problems considered, solutions obtained with h^{sum} were slightly longer than those obtained with h^{FF}.

We then changed the transition graph of each component automaton into a (directed) tree-like structure instead of a strongly connected graph, thus enabling the random generator to produce problems in which randomly chosen states have a fair chance of being unreachable. To explore the effect of directing the unfolding in this case, we generated 200 such instances (each with 10 components of 10 states per component), of which 118 turned out to be reachable and 82 unreachable. Figure 4.7 shows the results, in the form of distribution curves (prefix size on the top and runtime on the bottom; note that scales are logarithmic). The lower curve is for solvable problems, while the upper, "inverse" curve, is for problems where REACHABILITY$_\Sigma$ is negative. Thus, the point on the horizontal axis where the two

curves meet on the vertical is where, for the hardest problem instance, the reachability question has been answered.

As expected, the version using h^{sum} solves the problems with a positive solution much faster than that using h^{max}, which is in turn much faster than blind unfolding. However, also in those instances where REACHABILITY$_\Sigma$ is negative, the prefix generated by directed unfolding is significantly smaller than that generated by the original algorithm. In this case, the results from the h^{sum} and h^{max} versions are nearly indistinguishable. This is due to the fact that their pruning power (ability to detect dead end configurations) is the same.

4.6.3 Planning Benchmarks

To assess the performance of directed unfolding on a wider range of problems with realistic structure, we now consider benchmarks from the two last editions of the International Planning Competition (IPC-4 and IPC-5). These benchmarks are described in PDDL (the Planning Domain Definition Language), which we translate into 1-safe PT-nets as explained in Chapter 6.

In Figure 4.8, we present results for the first 26 IPC-4 instances of AIRPORT, a ground air-traffic control problem. Both the optimal and non-optimal AIRPORT planning problems are known to be PSPACE-complete [80]. The corresponding Petri nets range from 78 places and 18 transitions (instance 1) to 4611 places and 1711 transitions (instance 26). Optimal solution lengths (with respect to additive unit cost) range from 8 to over 200.

As before, the top graph shows the number of events pulled out of the queue, and the bottom graph shows the runtime. To avoid cluttering the graphs, we do not show the performance of h^{sum}. Its results comprise of those for h^{FF} and h^{max}. For small instances, the relatively small gain (1 order of magnitude fewer nodes) in unfolding size does not compensate for the overhead incurred in computing the heuristic function. However, for larger instances, directed unfolding reduces both size and runtime by over two orders of magnitude. The original version of MOLE is unable to solve six of the instances within a 600 second time limit. These instances describe ground traffic control problems over the topology of half of Munich airport. h^{max} fails to solve the two larger instances, but h^{FF} solves them easily.

In Figure 4.9 we present results for OPENSTACKS, a production scheduling problem. Optimal OPENSTACKS is NP-complete [106], while the problem becomes polynomial if optimality is not required. We consider the instances Warwick 91-120, which feature 10 products, 10 orders and an increasing ratio of three to five products per order. The IPC-5 "propositional" version of OPENSTACKS disables concurrency. In contrast, while still re-

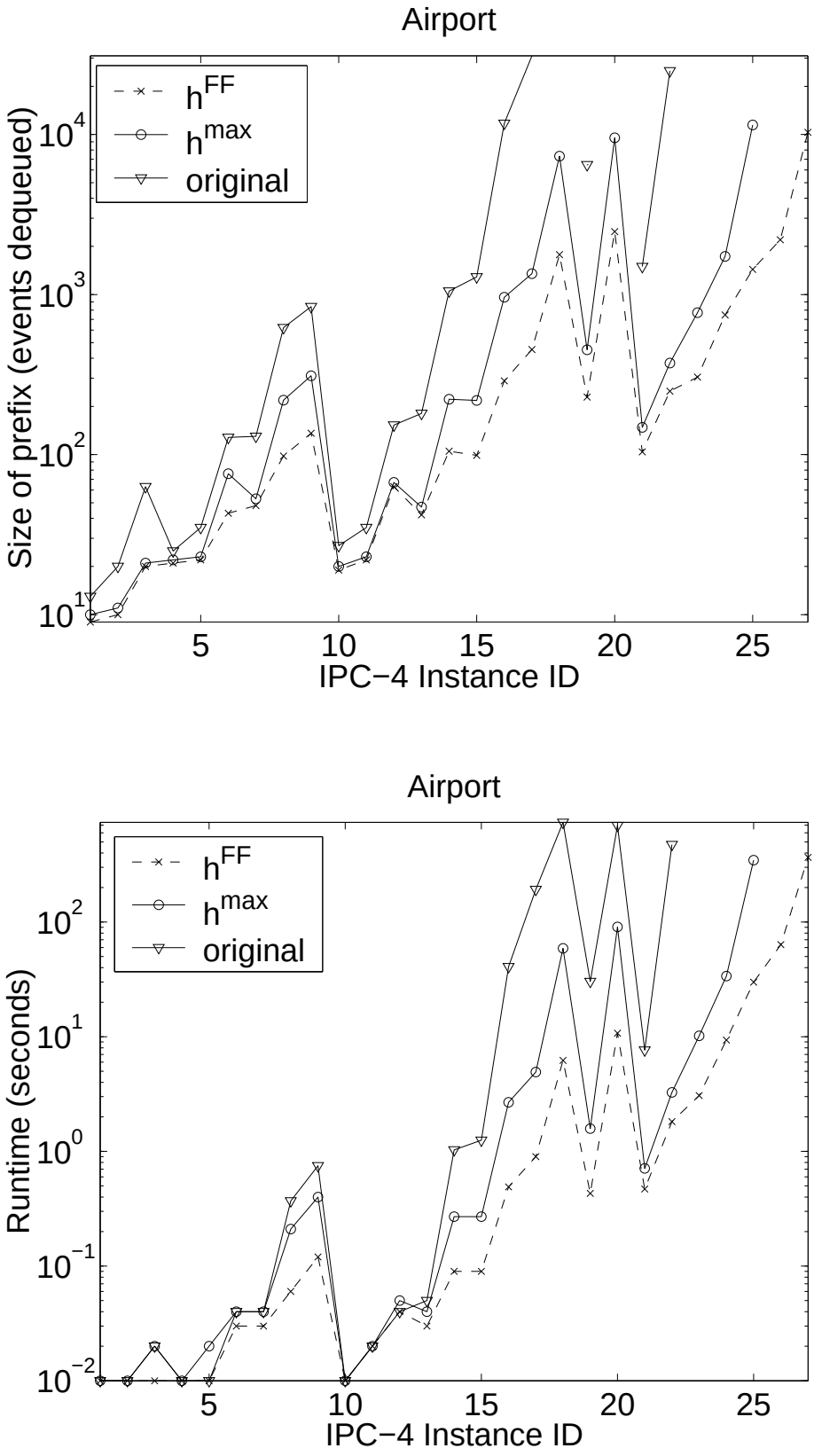


Figure 4.8: Results for AIRPORT planning problems.

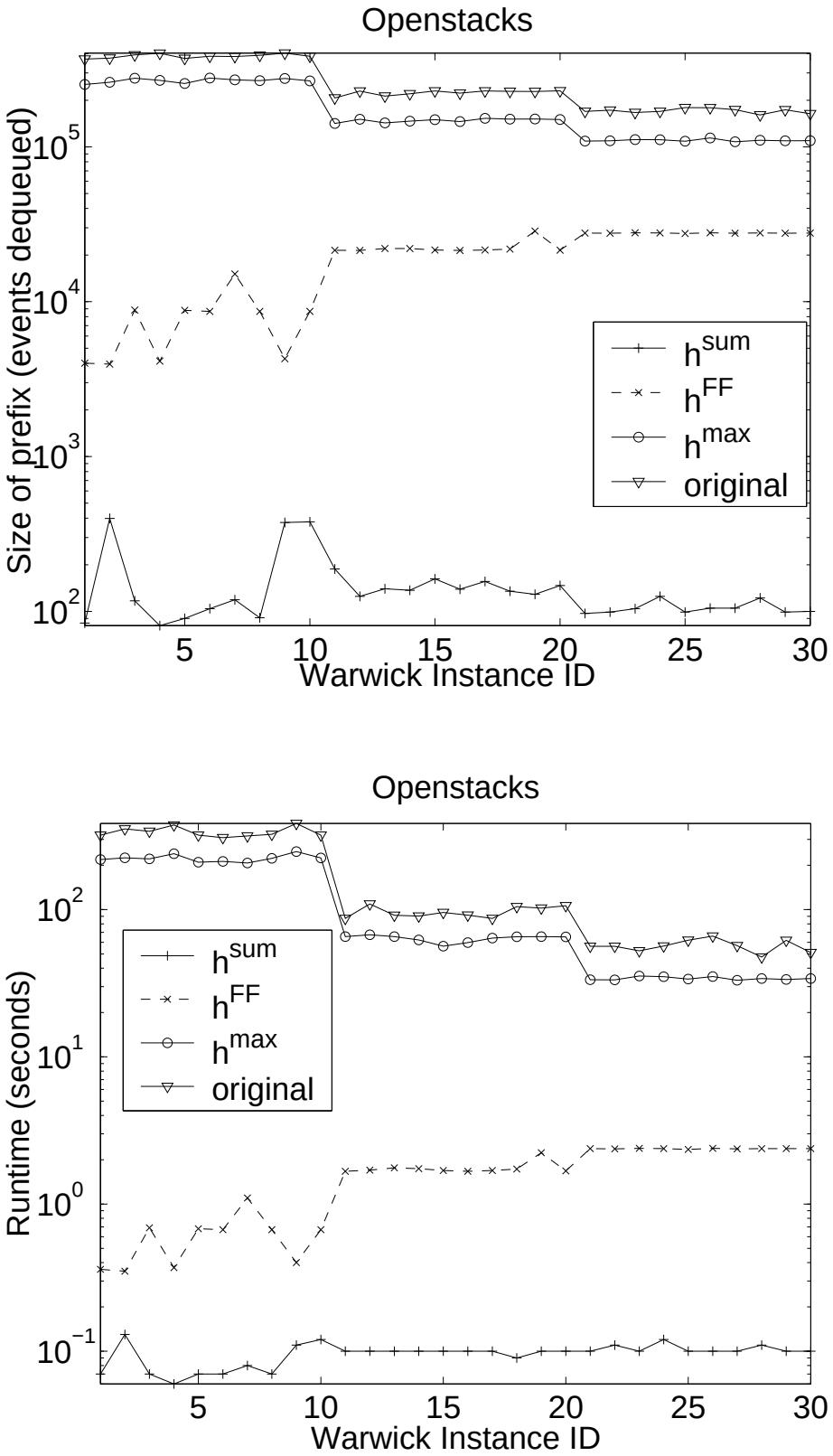


Figure 4.9: Results for OPENSTACKS planning problems.

taining the IPC-5 optimality criterion, we use the natural encoding of OPENSTACKS which allows several products to be produced in parallel. The corresponding Petri nets all have 65 places and 222 transitions, but differ in their initial markings. The optimal solution length varies between 35 and 40 action instances. In OPENSTACKS, the gap between directed and breadth-first unfolding is spectacular. The version guided by the h^{sum} heuristic consistently spends around 0.1 sec solving the problem: that is over 3 orders of magnitude less than the breadth-first version. The version directed by h^{FF} has runtimes ranging from 0.3 sec (instance 91) to 2.8 sec (instance 120). This shows that directed unfolding, which unlike breadth-first search is not confined to optimal solutions, is able to exploit the fact that non-optimal OPENSTACKS is an easy problem.

4.7 Conclusion

This chapter presented the theory of *directed unfolding*: controlling the unfolding process with informative strategies, for the purpose of optimality and increased efficiency.

The original contributions in this chapter are:

(1) Recognising and developing the potential of unfolding Petri nets to solve optimisation problems by:

- ◇ Identifying conditions which guarantee the *ERV-Fly* algorithm will identify an optimal solution to REACHABILITY$_\Sigma$ with respect to a particular cost function; and
- ◇ Developing partial orders on configurations which consider transition costs, in both additive and parallel formulations, and can be used to direct *ERV-Fly* to an optimal solution to REACHABILITY$_\Sigma$.

(2) Evolving the unfolding to be a principled method for solving the reachability problem, by enabling the process to be directed with problem specific information in the form of heuristics. This includes:

- ◇ Developing a framework for combining an arbitrary cost function with a heuristic function, such that the cost function can be optimised if the heuristic function is admissible;
- ◇ Implementing this framework for the case of additive and parallel cost functions, using results from (1);

- ◇ Providing the option, within this framework, to prioritise efficiency or solution optimality by using non-admissible or admissible heuristics respectively; and
- ◇ Demonstrating that suitable heuristic functions can be automatically extracted from the original Petri net.

We have extended the MOLE software to perform the following implementations of directed unfolding:

- ◇ *ERV-Fly*($\Sigma_R, \prec_{f+}, \prec_f$), with $g \equiv c_+$ and admissible heuristics $h \equiv 0$ and $h^{\max}$, and inadmissible heuristics h^{FF} and h^{sum};
- ◇ *ERV-Fly*($\Sigma_R, \prec_{f||}, \prec_f$), with $g \equiv c_{||}$ and admissible heuristics $h \equiv 0$ and h^{par}.

Furthermore, this work has impacted members of the Petri net community by inspiring (re-)consideration regarding:

- ◇ The formative assumptions underlying the *ERV* algorithm (e.g. semi-adequate orders); and
- ◇ The potential of unfolding, both in its capabilities (e.g. optimisation, fast reachability analysis) , and possible areas of application (e.g. planning).

In Part II of this thesis, we apply directed unfolding to the problem of automated planning; it will be shown that this work has also influenced the AI planning community.

4.7.1 Personal Contribution and Collaboration

The theory of directed unfolding and its application to automated planning has been previously published, to some extent, in [83], [84], and [20]. It is the result of both my own independent work, and collaboration with other researchers.

Langford White first suggested applying PT-net unfolding to solve planning problems with concurrent actions. This led me to investigate how the unfolding could be directed to find a minimal cost solution to the reachability problem, if transitions have arbitrary costs and their costs accumulate in an additive manner, i.e. optimal reachability analysis with respect to the additive cost function via on-the-fly unfolding. I motivated and presented this work at a DPOLP meeting, along with the challenges in translating a planning problem to a Petri net (see Chapter 6), and ideas for modelling and analysing actions with probabilistic effects.

This sparked the interest of Sylvie Thiébaux and Jussi Rintanen, both from the AI planning community, and we began collaborating.

Thiébaux suggested associating the cost function with an admissible heuristic function, and using the original Petri net to calculate the heuristic. This proved to be the crucial difference between directed unfolding being an interesting theory and a useful application, as it significantly increased efficiency. Furthermore, Thiébaux, Rintanen and myself were able to overcome the translation challenges I had identified (namely logical consistency via 1-safety and modelling negative preconditions and effects) and propose the translation from a classical planning problem to a Petri net, as presented in Chapter 6. Thiébaux implemented the translation in PETRIFY, a PDDL to PT-net translator written in Standard ML (SML). I extended the MOLE software to perform directed unfolding with respect to additive cost and admissible heuristic guidance. During this process I was in communication with Stefan Schwoon, one of the creators of MOLE who was able to provide valuable insight into the implementation, direct us to suitable test suites, and eliminate the "bugs" we encountered through our particular usage of the software (acknowledgement can be found on their website[6]). Rintanen, Thiébaux, White and I consolidated the translation, directed unfolding with respect to additive cost and admissible heuristic functions, and some preliminary experimental results, and submitted a paper to Mochart-06 [82] and the 2007 International Joint Conference on Artificial Intelligence (IJCAI-07) [84]. Rintanen subsequently presented our work at the Mochart workshop. I presented at IJCAI-07, where we had been nominated for a best paper award.

In the meantime I also presented my original work on directing the unfolding for optimal additive cost, and possible probabilistic extensions, at the Doctoral Consortium of the 2006 International Conference on Automated Planning and Scheduling (ICAPS-06) [83]. This caught the attention of Blai Bonet, and we decided to collaborate further.

Bonet believed it could be possible to use inadmissible heuristics. They clearly did not lead to adequate orders on configurations, which led to further investigation as to whether adequacy was in fact necessary. Bonet and Thiébaux recognised that the property of an adequate order refining the subset operator is stronger than required, thus leading to semi-adequate orders. This opened the door to a new family of strategies that can be used to direct the unfolding. Bonet revived the entire MOLE code, including parts of the original software, our extensions, and the heuristic functions. His choice of data structures significantly improved the efficiency of the program with respect to both time and memory usage. Patrik Haslum, who had contributed to discussions and experiments with respect

[6] `http://www.fmi.uni-stuttgart.de/szs/tools/mole/Readme`

to the IJCAI paper, became more heavily involved at this time; in particular he designed and implemented the random problem generator discussed earlier in this chapter. Bonet, Haslum, Thiébaux and I consolidated the work on directed unfolding with admissible and non-admissible heuristics, and unit cost actions, for UFO-07, the unfolding workshop held in conjunction with the 28th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency. Haslum presented at this workshop. We are aware that in response to the paper [20], presentation and subsequent discussions some members of the Petri net community are further investigating the implications of this research, having found it challenged their previous understanding of unfolding.

Concurrent to this, wanting to broaden the application scope to temporal planning, I was considering whether makespan (i.e. parallel cost) could be optimised rather than additive cost. I came to many dead ends, some of which have been discussed in this chapter. Thiébaux rose to the challenge with new ideas, and together we formulated an order on configurations very similar to that presented here as $\prec_{||}$. Our original formulation was not well-founded however, a problem remedied by the cardinality comparison included here. The $h_{||}$ heuristic presented in Chapter 4, and its implementation, is the result of collaboration between Haslum, Bonet and myself.

The particular formulation of directed unfolding found this chapter is my own work. The framework is more general and transparent than in preceding publications. Identification of the conditions which ensure optimality with respect to various criteria, and the formulation of ERV-Fly with two different orders on configurations (the queueing order and the cut-off order) is my own work, though this later idea has definitely been a topic of discussion amongst Bonet, Haslum, Thiébaux and myself. The framework combining a generic cost function g, semi-adequate order $\prec_g$ and heuristic function h is my own work, but was inspired by the high ideals of Bonet. The presentation of directed unfolding with respect to parallel cost, both with and without heuristic guidance, has not appeared previously; it has developed passed initial expectations to the extent presented here through valuable discussions with Haslum.

Part II

Planning Via Directed Unfolding

This page left blank.

Chapter 5

Automated Planning

PLANNING IS FOREMOST AN EXERCISE IN CONTROLLING COMBINATORIAL EXPLOSION
Russel and Norvig[1]

Part I of this thesis presented the theory of directed unfolding, a technique oriented for reachability analysis of concurrent systems which furthermore enables optimisation with respect to various cost criteria. In Part II we present the second contribution of this thesis: the application of directed unfolding to automated planning.

This chapter serves as an introduction to automated planning: what, why and how? The first section establishes the concept of automated planning, and consolidates its connection with reachability analysis. It begins with some first intuitions on the notion of planning, then outlines the theoretical and practical motivations for automated planning in general and domain independent planning in particular. It then presents a conceptual model for a planning problem which, subject to various assumptions, is equivalent to the REACHABILITY problem discussed in Chapter 2 and, importantly, demarcates the *classical* planning problem. This leads to the second section: classical planning. Here we explain our interest in what is probably the oldest and most restricted domain-independent planning problem. We formulate the classical planning problem, and summarise the main approaches to analysis: state space search, plan space search, planning as satisfiability and Graphplan. The discussion of analysis techniques includes comment on the development and impact of suitable heuristic functions for searching the state and plan spaces. The third section examines extensions to the classical planning problem, such that actions may be associated with an arbitrary cost or duration; we also briefly consider the possibility of actions with probabilistic effects. The chapter concludes with a summary.

[1] [147, p.407]

5.1 Automated Planning

Planning is central to the achievement of goals through intelligent behaviour. It is, generally speaking, a process of deliberation which considers how one can actively influence the world to achieve a particular result. Whilst the comprehensive notion of planning encompasses a wide range of problems, we can identify two defining requisites:

(a) A *world model*: planning requires a model of the world as a dynamic system with transformation rules capturing the anticipated changes caused by possible actions; and,

(b) A *goal*: planning requires some concept of what one desires to achieve in this world.

Automated planning is the act of formalising the planning process computationally. It is a key component of Artificial Intelligence (AI), arising in the mid-1960s - around one decade after the term AI was first coined by John McCarthy [113]; automated planning is also referred to as AI planning. The terms *planning system* and *planner* will be used to refer to technology which performs automated planning of some form.

5.1.1 Practical and Theoretical Motivation

One motivation for automated planning is purely practical: there is need for information processing tools to support humans in solving planning problems. The initial stimulant for this thesis - the Australian Defence Science and Technology Organisation's investment in the development of automated planning tools, via the DPOL Project[2] - is one example of the current demand for such technology, in the context of military operation planning. Another domain with critical safety and efficiency requirements, where interested parties are seeking automated planning support, is that of air traffic control. In 2004 various European stakeholders including Air France and the Dutch National Aerospace Laboratory sponsored the LEONARDO project[3], which investigated the feasibility of assimilating various planning support systems to increase the efficiency of air traffic control. The results indicated a clear benefit could be obtained through the proposed integration of planning tools.

To think more laterally on the practical applications of automated planning, consider that a planning problem is captured by a set of rules defining how the world can be manipulated

[2] http://nicta.com.au/research/projects/dynamic_planning,_optimisation_and_learning

[3] ec.europa.eu/transport/air_portal/research/doc/rtd_5_leonardo.pdf

and a goal which may be defined by an objective state: automated planning technology is relevant to a plethora of problems from playing bridge [156] to designing a manufacturing process, e.g. [128], to determining the shortest possible sequence of rearrangements between genome pairs [54].

A second motivation for automated planning is theoretical: if we qualify intelligence by the ability to act rationally [147], then planning - *"the reasoning side of acting"*[67, p.1] - is a critical component. Whilst research in automated planning is concerned primarily with the use of the computer as a tool for solving planning problems, it is arguable that advances in applied AI contribute to our understanding of the nature of intelligence. This is of particular interest to researchers in the interdisciplinary field of Cognitive Science and proponents of Searle's notion of strong AI [151]. For example, a 2001 Cognitive Science study on the planning abilities of humans found that *"adults and older children exhibited performance on planning tasks of varying complexity which matched that of artificial partial order planners"* [141, p. 941].

Ghallab *et al*, [67, p.2], identify that an important combination of these practical and theoretical motivations is the development of autonomous intelligent machines. Autonomy ultimately depends on continually answering the question of what to do next; the ability to compute effective plans is thus a key component of autonomous machines. Consequently NASA, for example, has been a strong force behind the development of AI planning, with complex artifacts such as the Deep Space One space probe [127], and space-based observatories like the Hubble Space Telescope [122] successfully equipped with automated planning technology. Furthermore, in accordance with the aforementioned theoretical perspective, the study of autonomous intelligent machines contributes to our understanding of intelligent behaviour.

5.1.2 Domain Independent Planning

Despite the variety of problems in planning, the balance of research in AI planning weighs heavily toward *domain independent* approaches. That is, where the planning system is separate from the world model, with the later given as input in conjunction with the goal, in a suitable formalism. Whilst there is benefit in addressing each application domain with representations and techniques focused specifically on its unique qualities, the general motivation for AI planning establishes firm ground for domain independent planning.

Firstly from a practical perspective, as mentioned at the outset of Chapter 2, it is desirable to avoid constructing a new modelling concept and analysis method for every different

situation because addressing each planning problem anew is more costly. In addition, the impact footprint is larger when research outcomes are are broadly applicable to a variety of domains. Secondly, from a theoretical perspective, studying the commonalities between all forms of planning helps to understand the process of planning as an aspect of intelligence. Finally, the study and design of autonomous intelligent machines necessitates domain independent planning .

Domain independence must come at a cost: the problem representation and analysis techniques are *"expected to work for a reasonably large variety of application domains"*[81, p 61] - the variation in problems is a real and practical difficulty. For this reason, in most cases where AI planning technology has been applied to real-world problems, the planning system has been enhanced with domain-specific knowledge. There are however examples of applications where this has not been required, e.g. [146] apply domain independent planning techniques to solve a real manufacturing problem.

Whilst this thesis stems from a project with a clear application domain - military operations planning - we chose to adopt a domain independent approach. The reason is that, as mentioned in Chapter 1, this thesis aims to develop and exploit the connection between AI planning and systems engineering: a domain independent approach is thus appropriate. Other project members have focused on the development of planning support specific to the application domain, e.g. [175]

5.1.3 Conceptual Model of the Planning Problem

Planning requires a model of the world as a dynamic system with transformation rules capturing the anticipated changes caused by possible actions, i.e. a *world model*. Thus a general world model for planning is the state transition system (see Definition 1), also known as the discrete event system (DES).

This is a very high-level abstraction however and encompasses a range of planning problems too large for automated computation within our current means. Recall the restrictive assumptions we made, in Chapter 2, about the state transition system $\Omega = \langle S, A, E, \gamma \rangle$. We make these same assumptions with respect to the world model of a planning problem, and impose further assumptions regarding the goal of a planning problem and the knowledge available to an automated planning system:

1. The set of states S is finite.

2. The result of every action and event is deterministic, i.e. for every $s \in S$, $a \in A$ and

$e \in E$, $|\gamma(s, a, e)| \leq 1$.

3. The system remains static unless a controlled transition, i.e. an action, takes place. That is, the set of events E is empty.

4. Actions and events are instantaneous state transitions.

5. The system is fully observable: this means the planning system has complete knowledge about the state

6. A goal can be specified as a set of states $S_g \subseteq S$.

7. Planning occurs offline: this means the planning system is not concerned with any change that may occur in the system while it is planning.

Conceptually, a world model for a planning problem satisfying Assumptions 1, 2 and 3 is a restricted state transition system $\Gamma = \langle S, A, \gamma \rangle$ (see Definition 2. Assumptions 4 - 7 then reduce planning to the problem of REACHABILITY (see Chapter 2, Section 2.2), with $S_R \equiv S_g$. That is, given these assumptions, a planning problem **P** is defined by a restricted state transition system $\Gamma = \langle S, A, \gamma \rangle$, an initial state $s_0 \in S$ and set of goal states S_g. A *solution plan* for **P** is a sequence of action instances $\langle a_1, a_2, a_3, ..a_n \rangle$ corresponding to a sequence of state transitions $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_1 = \gamma(s_0, a_1), s_2 = \gamma(s_1, a_2) \ldots, s_n = \gamma(s_{n-1}, a_n)$ and $s_n \in S_g$. The task of a planning system is to synthesise a solution plan for a given problem **P**$= \langle \Gamma, s_0, S_g \rangle$.

5.2 Classical Planning

The real life success of a plan that has been synthesised by an automated planner is constrained by computational power: modelling and exhaustively considering all information is not feasible. It is necessary to make limiting assumptions about the planning problem. One generally aims for restrictions which:

(a) Depart from reality so infrequently that they are relatively negligible; and/ or

(b) Facilitate the development of technology which may subsequently be extended to incorporate more real-world considerations.

A significant proportion of work in AI planning addresses what is loosely referred to as *classical planning*, so called because it arose through the earliest efforts toward automated

planning, by Green in 1969 [72] and Fikes and Nilsson in 1971 [63], for example. Classical planning is also referred to as *STRIPS planning*, in reference to the STanford Research Institute Problem Solver (STRIPS), one of the first planners to operate in this environment [63].

The boundaries of classical planning are somewhat imprecise, and we look here to Ghallab, Nau and Traverso's statement that *"Classical planning refers generically to planning for restricted state transition systems"*[67, p.17], which they substantiate with the set of assumptions just listed [67, p.9-10].

Whilst classical planning is very restrictive with respect to exspressivity, and consequently largely unrealistic, it has proved to be an effective baseline for the development of algorithms and techniques from which more expressive models can be developed. Furthermore, despite its limitations, many real-world planning problems have been modelled and solved in the classical environment. Note also that even with its restrictive assumptions classical planning is still a hard problem. In terms of complexity the problem specified shortly is PSPACE complete [67].

Since classical planning is the baseline for AI planning, and an objective of this thesis is to build on the connection between systems science and automated planning at a foundation level, this thesis addresses the classical planning paradigm. It is practical to develop the means to analyse a restricted world before considering more realistic (and thus increasingly complex) problems. We do take some small steps outside of the classical planning environment, relaxing some of the restrictive assumptions, and in this way consider the potential for directed unfolding to be extended to more expressive problems. As discussed in Chapter 7 however, future work must push the boundaries much further into reality.

The central components of classical planning are:

(a) Representation: how to represent states, actions and the state-transition function symbolically (i.e. without explicitly enumerating S, A and γ).

(b) Analysis: how to efficiently search for a solution. This involves identifying a search space and determining how nodes within it will be transformed (i.e. the search algorithm and control techniques).

We now address each of these in turn.

5.2.1 Representation

It is impractical to explicitly represent the states in a world model for planning: we would immediately become victim to the state explosion problem (as explained in Chapter 2 Section 2.3). Scientists of AI planning seek an implicit representation of planning problems; this is why automated planning algorithms are generally symbolic (as defined in Chapter 2).

STRIPS Representation

In this thesis we employ a representation of the classical planning problem based on propositional logic. It is semantically equivalent to what Ghallab *et al* refer to as *set theoretic* representation [67, Chapter 2.2], and what is widely referred to in planning literature as the grounded STRIPS representation (in this case extended with negative preconditions) [63].

Let $V = \{v_1, v_2, \ldots, v_N\}$ denote a set of *propositions*. The set of *literals* over V is the disjoint union $L = V \cup \{\neg v | v \in V\}$. The *complement* $\bar{l}$ of a literal $l \in L$ is defined by $\overline{v} = \neg v$ and $\overline{\neg v} = v$ for $v \in V$. For a subset of literals $X \subseteq L$, we define its complement to be the set $\overline{X} = \{\overline{x} | x \in X\}$. A subset X of L is *consistent* if and only if $X \cap \overline{X} = \emptyset$[4]. We denote by 2^L_C the set of all consistent subsets of L. A subset X of L is a *state* (over V) if and only if $X \in 2^L_C$ and $|X| = |V|$. We denote the set of states over V as S_V. A *grounded planning operator* over V is a pair $\langle P_{re}, E_{ff} \rangle$ where $P_{re}, E_{ff} \in 2^L_C$. An operator represents an action; P_{re} and E_{ff} capture its preconditions and effects respectively. An operator $o = \langle P_{re}, E_{ff} \rangle$ is *applicable* to a state $\phi \in S_V$ if $P_{re} \subseteq \phi$ (i.e. the preconditions hold). The state transition function is defined as $\gamma_o(\phi, o) = E_{ff} \cup (\phi \setminus \overline{E_{ff}})$, if o is applicable to ϕ, and is otherwise undefined. Let us demonstrate that γ_o indeed maps to a state:

Lemma 5.2.1. *Let the operator $o = \langle P_{re}, E_{ff} \rangle$ (over V) be applicable to state $\phi \in S_V$. Then $\gamma_o(\phi, o) = E_{ff} \cup (\phi \setminus \overline{E_{ff}}) = \psi$ is an element of S_V.*

The following proof has been adapted from personal correspondence with White [105].

Proof. We first prove that ψ is consistent, and then show it contains $|V|$ literals.

(a) $\psi \cap \overline{\psi} = (E_{ff} \cup (\phi \setminus \overline{E_{ff}})) \cap (\overline{E_{ff} \cup (\phi \setminus \overline{E_{ff}})})$
$= (E_{ff} \cup (\phi \setminus \overline{E_{ff}})) \cap \overline{E_{ff}} \cup (\overline{\phi \setminus \overline{E_{ff}}})$
$= (E_{ff} \cap \overline{E_{ff}} \cap (\overline{\phi \setminus \overline{E_{ff}}})) \cup ((\phi \setminus \overline{E_{ff}}) \cap \overline{E_{ff}} \cap (\overline{\phi \setminus \overline{E_{ff}}}))$
$= \emptyset$, because $(E_{ff} \cap \overline{E_{ff}} = \emptyset$ since E_{ff} is consistent,

[4] We use upper case letters to denote subsets of L and lower case letters to denote elements of L.

and $(\phi \setminus \overline{E_{ff}}) \cap \overline{E_{ff}} = \emptyset$.

Thus ψ is consistent.

(b) The literals in E_{ff} and their complements $\overline{E_{ff}}$ are the only elements added to and subtracted from the state ϕ when o is applied. For every literal $e \in E_{ff}$, either $e \subseteq \phi$ or $\overline{e} \subseteq \phi$. In the first case e will remain a member of the state after the application of o; since E_{ff} is consistent $\overline{e}$ is not in ϕ and can thus not be subtracted from it when the operator is applied. In the second case e will be added to the state and $\overline{e}$ will be removed. Thus the total number of literals in the state does not change, i.e. $|\psi| = |\phi| = |V|$.

□

Note that the semantics of planning operators include the condition that $P_{re} \cup E_{ff} = \emptyset$, i.e. the same literal does not appear both in the preconditions and effects of the same operator. This is because we assume that any literal in the preconditions of an operator remains true when the operator is applied, unless its complement appears explicitly in the set of effects.

We now give formal definitions of a *world model* for classical planning, the grounded classical *planning problem*, the *statement* of a classical planning problem, and a *solution plan*.

Definition 19 (World Model for Classical Planning)**.** *Let V be a finite set of propositions. The world model for classical planning is a restricted state transition system $\Gamma = \langle S, A, \gamma_o \rangle$ such that:*

(a) $S = S_V$*;*

(b) A *is a set of grounded planning operators over* V*; and*

(c) For $o = \langle P_{re}, E_{ff} \rangle \in A$ *and* $\phi \in S$, $\gamma_o(\phi, a) = E_{ff} \cup (\phi \setminus \overline{E_{ff}})$ *if* o *is applicable to* ϕ *and is undefined otherwise.*

This is also commonly referred to as a classical planning *domain*.

Definition 20 (Classical Planning Problem and Statement)**.** *A grounded classical planning problem is a triple* $\mathbf{P} = \langle \Gamma, I, G \rangle$ *where:*

- $\Gamma = \langle S_V, A, \gamma_o \rangle$ *is a world model.*
- $I \subseteq V$ *is the set propositions which are true in the initial state. The initial state is given by the set of literals* $\phi_0 = I \cup \{\neg v | v \in V \setminus I\}$.

- $G \in 2^L_C$ *is a consistent set of literals over* V*, i.e. the goal literals.*

The statement of a grounded classical planning problem is a four-tuple $\mathbf{P_s} = \langle V, I, A, G \rangle$. $\mathbf{P_s}$ *is a syntactic specification of* $\mathbf{P}$*; in particular the world model is specified by* V *and* A*.*

Observe that the statement of a classical planning problem avoids enumerating the set S, and the state transition function is implicit in the representation of actions as planning operators. In some literature the set of variables V is not included in the statement of the planning problem as it may be inferred from the initial state (providing this is fully specified, i.e. $I = s_0 \in S_V$). We have instead included V, and specified the initial state by $I \subseteq V$ for ease of translation from a planning problem to a PT-net reachability problem (see Chapter 6).

Definition 21 (Solution Plan)**.** *A plan* π *for a classical planning problem* $\mathbf{P} = \langle \Gamma, I, G \rangle$ *is a sequence of actions in* A*, i.e.* $\pi = \langle a_1, a_2, a_3, ..a_n \rangle$*. A plan is applicable to* $\mathbf{P}$ *if it corresponds to a sequence of state transitions* $\langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ *such that:*

- $\phi_0 = I \cup \{\neg v | v \in V \setminus I\}$*; and*
- $\phi_1 = \gamma_o(\phi_0, a_1),\ \phi_2 = \gamma_o(\phi_1, a_2) \ldots,\ \phi_n = \gamma_o(\phi_{n-1}, a_n)$.

Furthermore, this plan is a solution to $\mathbf{P}$ *if:*

- $G \subseteq \phi_n$.

It follows from these definitions that finding a solution plan π for a planning problem $\mathbf{P} = \langle \Gamma, I, G \rangle$ is equivalent to solving REACHABILITY for restricted state transition system $\Gamma = \langle S_V, A, \gamma_o \rangle$, with initial state $s_0 \equiv \phi_0 = I \cup \{\neg v | v \in V \setminus I\}$ and $S_R \equiv \{\phi \in S_V | G \in \phi\}$.

Whilst the statement of a classical planning problem is somewhat ambiguous, in that more than one classical planning problem can map to the same statement, it is sufficient for the specification of a problem because if two planning problems have the same statement then they also have the same set of reachable states and the same set of solution plans. Ghallab, Nau and Traverso [67, Sections 2.2.3 and 2.3.4] provide a thorough discussion of the relationship between a classical planning problem and its statement, including proof that the statement is sufficiently unambiguous to be used as the problem specification.

Given this, we can solve a classical planning problem $\mathbf{P}$ via its statement $\mathbf{P_s} = \langle V, I, A, G \rangle$. Note that the requirements of an applicable plan, and furthermore a solution plan, for $\mathbf{P} = \langle \Gamma, I, G \rangle$ can be checked using only the statement of $\mathbf{P}$. We will generally speak of a

planning statement as the actual planning problem. In particular we refer to a plan being a solution for a statement; technically this means it is a solution plan for the planning problem specified by the statement.

5.2.2 Analysis

The first steps of AI planning were taken by Newell, Shaw and Simon's General Problem Solver (GPS) [55], in 1957. GPS was an attempt to simulate the problem solving abilities of humans; it was (theoretically) able to solve formalised problems, such as puzzles and chess playing, via state space search guided by estimated differences between the current state and goal propositions. The late 1960's saw the emergence of technology intended specifically for planning, with Green casting planning as a theorem proving problem [72]. However the inefficiency of theorem-proving techniques at that time left this approach wanting, and research re-focused on planning as a search problem.

State Space Planning

Synthesising a plan through search involves defining a search space, then exploring this space to find a node/point or path that defines a solution plan. The early search based planners followed the lead of STRIPS [63], the first planner to directly formulate planning as search, and defined points in the search space as states in the planning world. This is commonly referred to as *state space planning*. The simplest way to explore the state space is to perform a forward search from the initial state. At a given state, applying any applicable operator leads to a different point in the search space. A solution plan is defined as any sequence of operators that can be applied to traverse the state space from the initial state to a state satisfying the goal propositions. Note that this is equivalent to the forward state space search described in Chapter 2. Alternatively, a backward exploration begins from the states satisfying the goal propositions, applies the inverses of operators to traverse to different points, and stops when a point is the initial state. Forward and backward exploration of the state space is often referred to as progressive and regressive search respectively.

There is argument for exploring the state space in either direction. On the one hand, the backward search concentrates only on those paths which lead to the goal, which can reduce the number of points traversed to from a given point (i.e. a lower branching factor). On the other hand, a forward search considers fully specified states (since the initial state is fully specified) whilst a backward search considers partial state descriptions (since the goal not necessarily a fully specified state); Bacchus and Kabanza [4] argue that effective strategies

for search control depend on the ability to evaluate the state of a plan, and view this in favor of forward search.

Obviously state space planning can suffer greatly from the state explosion problem (see Chapter 2); to reduce the number of states explored, heuristic functions are employed to estimate how far a given point is from a solution in order to explore "better" points first. The most common heuristic at this time was that of *means-end analysis* which, introduced by GPS, focuses on operators that appear likely to achieve the goal propositions. Other early planners using this technique include STRIPS [63] and Prodigy [32]. This was still insufficient in the face of the state explosion problem however; in addition it was difficult to extend to consider causal and temporal relationships between actions [103]. Consequently researchers were driven to look at the search space from another perspective.

Plan Space Planning

In the mid 1970's the dominant approach to classical planning changed to searching the space of partially constructed plans, commonly referred to as the *plan space*. This facilitates reasoning directly about the relationships between actions rather than states. Each point in the plan space is a partially constructed plan, which includes a set of actions (represented by operators) and ordering constraints on the actions. Furthermore, if the original operators are not grounded as presented here, i.e. the operators are specified using variables not literals, then a partially constructed plan may contain constraints on the binding of an action's variables. One point in the plan space is transformed to another by the inclusion of an operator or an ordering constraint between operators. An exploration of the plan space generally begins with a skeleton plan, where the first operator in the plan produces the initial state and the last operator in the plan has the goal propositions as preconditions. Plan transformations continue to be made until a solution is found. A solution is a set of actions and a set of constraints, such that any linearisation of the actions consistent with the constraints corresponds to a path in the state space from s_0 to some state $s \in S_g$.

NOAH, developed by Sacerdoti in 1974 [148], pioneered the technique of plan space planning; the transformation operations were somewhat ad hoc however, and in 1997 Tate introduced the notion of *causal links* in Nonlin [159][5]. A causal link connects a precondition of one operator with an effect of another operator, where the effect asserts the precondition.

[5]In addition to founding plan space planning as we know it today, NOAH and Nonlin are also considered the first planners to perform Hierarchical Task Network (HTN) planning, as they allow a partial plan to contain abstract operators which can be incrementally reduced to ground planning operators. This is outside the scope of classical planning however and not discussed further here.

Subsequently, for book-keeping purposes, a partial plan may also include a set of causal links. An *open goal* in a partial plan is a precondition, for an operator in the plan, which is not asserted via a causal link. An action is a *threat* if one of its effects may be inconsistent with a causal link. Ordering constraints must eliminate threats, ensuring that preconditions are not undone before the action requiring them takes place. For the purpose of plan synthesis, the transformation of a partial plan should achieve an open goal or remove a threat. This approach is often referred to as *partial order causal link* (POCL) planning. Various other strategies have been developed to ensure unordered actions do not interfere with each other, for example in TWEAK [34] Chapman employs what he calls the Modal Truth Criterion to check the truth of each precondition in the partial plan[6]. However most strategies are based on the POCL approach which (although introduced by Tate in 1977 [159]) was made popular by McAllester and Rosenblitt [112] in 1991 through their formulation of *systematic nonlinear planning* (SNLP). One of the most well known POCL planners is UCPOP [136], a descendant of Nonlin via SNLP.

Partial Order Planning and The Least Commitment Principle

Plan space planning facilitates *partial order planning*: searching through the space of partially ordered plans. Conversely, state space planning is more suited to *total order planning*, which is defined analogously. Note that despite this fact, the distinction between total and partial order planning is separate to the distinction between plan space and state space planning. There are some total order plan space planners, e.g. Waldinger's regression planner [167] and Warplan [168], and state space planning can be employed to synthesise a partially ordered plan [69].

Partial order planning is a partial order method, as defined in Chapter 2. In his original paper on partial order/ plan space planning [148] (referred to in some literature as nonlinear planning) Sacerdotti states "But plans themselves are not constrained by limitations of linearity". Recall that in Chapter 2 we considered how modelling can introduce "fictional" complexity to the analysis of a problem. In particular we noted that the arbitrary interleaving of concurrent actions contributes significantly to the state explosion problem, and that partial order methods attempt to alleviate this problem by considering the partial order model of system execution.

In fact, avoiding the interleaving of concurrent events is just one application of the *least commitment principle*. A key idea behind plan space planning is the principle of con-

[6]The Modal Truth Criterion has since been shown to be sufficient but unnecessary [64].

straining a plan as little as possible during its construction [169], i.e. least commitment. A forward or backward traversal of the state space, for example, requires making a commitment to an action's *relevance* (whether it will be part of a solution plan) and *position* (when, during execution of the solution plan, this action will be executed) [89]. Conversely, algorithms which traverse the plan space reason about the relevance of an action without necessarily fixing its position in the plan. Postponing commitment can reduce the need to backtrack over past decisions. This in itself can make planning more efficient; it also increases the chance of goals being *trivially serialisable*, which makes plan synthesis easy [89, 7]. A set of goals are trivially serialisable, with respect to a particular planning algorithm, if each goal can be solved sequentially in any order without violating past progress [7, 88].

The downfall is that reducing the level of commitment made during plan synthesis increases the difficulty of traversing from one point to another in the search space [89]. For instance it is easier to identify which actions are applicable in a given state (as in state space planning) than to determine how inconsistencies in a partial plan can be resolved. Furthermore, the level of commitment made by current plan space planners is such that a partial plan does not correspond to a state of the world. The consequence of this is highlighted by the revival of state space planning due to the development of powerful state based heuristic functions (to be discussed shortly). So, again, we see the compromise which must be made when modelling: restricting the information considered can both complicate and simplify analysis. Of course, the right balance depends on the particular problem. Thus it is beneficial to have another option; in the next chapter we explain why planning via directed unfolding may provide a compromise that lies somewhere between state space and plan space planning with respect to commitment.

Narrowing our scope back to commitment with respect to action ordering, let us note that even this appears to have implications beyond our current understanding. Barrett and Weld [7] contend that partial order planning is superior to total order planning because a planning problem is more likely to be trivially serialisable with respect to a partial order planning algorithm than a total order planning algorithm. Indeed, experimental results in [7] find no domain in which a total order planner performs better than a partial order planner, but several domains for which the partial order planner is exponentially more efficient. Minton, Bresina and Drummond [123] argue that the issue is more complicated than this. They compare a total order planner TO and partial order planner UA which are similar in all manner except their commitment to actions added during plan synthesis. TO orders a new action with respect to *all* other actions, whereas UA orders a new action only with respect to other interacting actions. Hence, for any plan produced by TO, UA produces a corresponding

plan that is less constrained with respect to ordering, or equivalent. Minton *et al* observe that the search space of the partial order planner is exponentially smaller than that of the total order planner. But, whilst UA can outperform TO, the difference in efficiency is highly dependent on the particular control strategy employed, and the density and distribution of solutions in each search space. For example, consider an extreme case where the space of partial order plans consists of unordered solution plans and totally ordered plans which are not solutions. Each unordered solution will correspond to an exponential number of solutions in the totally ordered space, and there is a one to one mapping between plans that are not solutions in the partial and total order spaces. Thus there will be a higher density of solutions in the space of totally ordered plans. The converse could also occur, where there are totally ordered solution plans combined with a large number of unordered plans which are not solutions, resulting in the space of partially ordered plans having a higher solution density. Minton *et al* also find that distribution sensitive search strategies like depth first search enable the partial order planner to outperform the total order planner when solutions are not distributed uniformly throughout the search space.

Soon after Minton *et al* published these results, the partial order versus total order planning appears to have temporarily subsided. The twenty year focus on plan space planning, and in particular its role in partial order planning, was ended by two developments: Kautz and Selman's impressive results casting planning as a propositional satisfiability (SAT) problem [93] and Blum and Furst's Graphplan algorithm [17].

Planning As Propositional Satisfiability

Improvements in the performance of SAT methods, e.g. [152], encouraged reconsideration of planning as theorem proving. Experimental results from Kautz and Selman in 1996 [93] suggested planning as SAT might yield the fastest classical planner to date [169].

Given a set of discrete variables, legal domains for each variable and a set of constraints on values groups of variables can take, the constraint satisfaction problem (CSP) is to find an assignment of values to all the variables such that none of the constraints are violated, or determine that no such assignment exists. A satisfying assignment is called a *model*. If the variables are Boolean, then a CSP is a SAT problem.

Planning as SAT involves guessing the length of a plan, translating the problem to a propositional formula, and trying to find a model. If no model is found then the length is increased and the process is repeated. Suppose we restrict the planning problem to the problem of finding a plan of length n; this is referred to as a *bounded planning problem*. Each

$i, 0 \leq i \leq n$ is a *step* of the planning problem. Many encodings have been proposed for the translation, but most introduce a propositional variable for every action at every step, and every proposition at every step. The bounded planning problem is then encoded as a conjunction of *clauses*, where a clause is a disjunction of literals. For the purpose of modeling a planning problem, a clause captures constraints such as: propositions true in the initial state are true at step 0; the goal propositions are true at step n; an action occurring at step k implies its preconditions are true at step k and its effects are true at step $k + 1$, etc. Fast simplification algorithms, such as unit propagation and literal elimination (e.g. [164]), are used to shrink the formula. A model can then be sought via a systematic or stochastic search of the space of partial assignments to the variables in the formula.

Planning as SAT synthesises a *parallel* plan. A parallel plan is a sequence of sets of actions. For example $\langle \{a_1, a_6\}, \{a_3, a_2, a_5\} \rangle$ represents all sequences beginning with a_1 and a_6 in any order followed by a_3, a_2 and a_5 in any order. A parallel plan is obviously less constrained than a totally ordered plan but more constrained than a partially ordered plan.

The most serious disadvantages of planning as SAT are the size of the encoding and the restriction to discrete time. Since all possible actions and propositions are represented explicitly for each step, the number of variables and clauses can be very large. As the encoding described is limited to steps representing discrete points in time, it can not handle actions with varying durations. In attempt to rectify this shortcoming, a *causal encoding* has been proposed [92], based on the causal link representation employed in plan space planning. There has yet to be an efficient implementation of this idea however. In addition, recall that the encoding requires the original planning problem to be bounded to the problem of finding a plan of known length n. If no model exists for the bounded problem, it does not mean that no solution plan exists for the original unbounded problem.

Graphplan

Parallel to the growing enthusiasm for planning as SAT, the Graphplan algorithm, unveiled by Blum and Furst [17] in 1995, attracted considerable attention. Graphplan first constructs a *planning graph*, to a certain depth, then attempts to extract a solution plan from it. A planning graph is a layered graph where arcs are only permitted from one layer to the next. The layers correspond to steps, similar to those defined above, which represent discrete points in time when actions may be executed. There are two types of layers: propositional layers and action layers. The nodes in a particular propositional layer map to propositions which could be true at that time point, and similarly the nodes in an action layer map to actions which could be executed at that time point.

The initial layer P_0 consists of the set of propositions describing the initial state s_0. The next layer is the first action layer, A_0. It consists of all actions whose preconditions appear in P_0. Arcs connect the propositions in P_0 to the actions they support in A_0. The following layer, P_1, is the set of all propositions in P_0 together with all the effects of the actions in A_0. There are two types of arcs between an action layer and the following proposition layer: *positive* arcs go from an action to its positive effects, and *delete* arcs go from an action to its negative effects.

During this construction, pairwise mutual exclusion (mutex) links are recorded. A mutex relation exists between two actions at a given level if any of the following conditions hold:

(a) *Inconsistent effects*: the effect of one action negates a positive effect of the other;

(b) *Interference*: the effect of one action negates a precondition of the other; or

(c) *Competing needs*: a precondition for one action is mutually exclusive with a precondition for the other.

A mutex relation exists between two propositions if every possible pair of actions that can achieve them are mutually exclusive.

Mutex propagation is a form of reachability analysis: the existence of all goal propositions in a layer is a necessary (but not sufficient) condition for the existence of actions in the preceding layers which can transform the system to a state satisfying the goal. Once a layer containing the goal propositions has been identified, Graphplan uses the planning graph to guide the search for a solution plan. Graphplan, like planning as SAT, synthesises a parallel solution plan.

Whilst the original Graphplan algorithm employs a search procedure for solution extraction tailored specifically for the planning graph structure, it is also possible to cast the problem of solution extraction to a propositional satisfiability problem. In fact by viewing the planning graph for a given problem as a SAT problem, the expansion phase corresponds to construction of the conjunctive formula and the graph extraction phase is the search for a model to satisfy it. The specific details of this process were originally described by Do and Kambhampati [44]. It was Kautz and Selman however who first observed that the simplification performed in the construction of a planning graph, via the propagation of pairwise mutexes, is more powerful than the unit propagation used in their previous SAT system [93]. Their BLACKBOX planner [94], released in 1998, generates a planning graph, then extracts a solution using SAT techniques.

Heuristic State Space Planning

In the 1998 International Conference on AI Planning and Scheduling (AIPS-98) Planning Competition [114], three out of four planners in the STRIPS track were based on the ideas of Graphplan and planning as SAT. The fourth, HSP [18], was a heuristic search planner that proved competitive with the other entrants, and can be accredited with the subsequent revival of state space planning. HSP performs a state space search guided by heuristics extracted automatically from the problem encoding. The significant advancement in state based heuristic functions since the first era of state space planning enabled state space planners to outperform the plan space planners which had claimed superiority decades earlier. One of the most successful classical planners today, FF (Fast Forward) [86], performs a forward state space search using the heuristic h^{FF} described in Chapter 4.

A Revival of Partial Order Planning?

In the paper "Reviving Partial Order Planning" [130] Nguyen and Kambhampati argue that considering heuristic state space planning and CSP-based planning (such as planning as SAT and Graphplan) superior to partial order planning is a misinterpretation of the advances in these areas. They contend that the flexible nature of partial order planning has benefits beyond the realm of these other approaches. Indeed, Smith *et al*'s [154] assessment of the current and ideal capabilities of planning systems concludes that a POCL framework is more easily extended to handle durative actions and temporal and resource constraints than other planning approaches. They speculate that the difficulty with POCL approaches lies in controlling the search. Nguyen and Kambhampati attempt to address this issue via REPOP, a descendant of UCPOP equipped with heuristics for ranking partial plans, and an improved ability to detect and resolve conflicts.

Vidal and Geffner [165] identify similar motivation for the development of CPT (Constraint Programming Temporal planner). Temporal planning is a variant of classical planning where actions have time durations. Vidal and Geffner recognise that POCL planning is particularly suited for temporal planning, but is limited by its weak ability to reduce (prune) the search space. They rectify this by combining a plan space search with powerful pruning rules which are implemented as constraints. The result, CPT, is probably the best optimal temporal planner today.

Whilst CPT is a partial order planner which uses constraint propagation for pruning, planning via directed unfolding is a forward partial order planning algorithm which employs the powerful heuristic functions used in state space search to guide and prune the search space.

5.3 Extending the Classical Planning Problem

As mentioned previously, the classical planning problem is very restrictive with respect to expressivity. We now look at two separate extensions to the classical planning world model: operators with arbitrary positive costs, which may represent the economic penalty or time duration of an action, and operators with probabilistic effects. We highlight current state of the art planners which address these extensions.

5.3.1 Action Costs

If more than one solution plan exists, as will most often be the case, it is desirable to be able to select the best plan with respect to some criteria. Here we consider a plan as optimal if it minimises a specified cost function.

We extend the classical planning model with a cost function $cost$ that maps operators to positive rational numbers. This may capture the economic cost or penalty incurred when applying an action, or the duration of an action, for example. For a given planning problem, we can talk about a solution plan with minimum overall cost as an optimal plan, and its cost as the optimal cost.

Additive Cost of a Plan

In the presence of costs, a plan π has an *additive cost* of:

$$cost_+(\pi) = \sum_{o \in \pi} cost(o)$$

Clearly, if all operator costs are equal to one, then an optimal solution is a plan with a minimal number of operator instances. In the literature on AI planning, the total number of operator instances in a plan is often referred to as the *length* of the plan.

It is necessary to distinguish between planners that attempt to optimise plan length (i.e. all operators have unit cost) and those which can consider operators with arbitrary, positive costs. Since it is unrealistic to think all actions in a plan will be equally demanding, the sum of the (arbitrary) costs of actions in a plan seems a more appropriate measure than its length. Despite this, most planners which consider additive cost, do so with respect to plan length. The only planner we are aware of that can find an optimal additive cost plan when

operators have arbitrary costs is HSP$_0^*$[7]. HSP* is a family of heuristic search planners, each utilising a different heuristic function. HSP$_0^*$ performs a backward state space search using an admissible heuristic function.

The new partial order planning algorithm presented in this thesis can be applied to synthesise optimal and suboptimal plans, with respect to the additive cost of operators with *arbitrary* positive costs.

Parallel Cost of a Plan

In the presence of costs, a plan π has a *parallel cost* of:

$$cost_{||}(\pi) \; = \; \max_{\sigma \in \pi} \sum_{o \in \sigma} cost(o)$$

where σ is a causal chain within π, and the maximum is taken over all such chains.

Clearly, if π is a totally ordered plan, then $cost_{||}(\pi) = cost_{+}(\pi)$.

The most common usage of parallel cost arises when the cost of an operator is its duration. Then $cost_{||}(\pi)$ corresponds to the *makespan* of π in automated planning literature. If all operators have the same duration then minimising $cost_{||}(\pi)$ minimises the number of time steps in the plan.

It is necessary to distinguish between planners that optimise makespan under the assumption all operators have equal duration and those which can consider arbitrary durations. Planners using Graphplan and/or SAT-based approaches reason about parallel plans, where time steps are defined at equal intervals arbitrarily set as one. Consequently these planners usually optimise the number of steps in a plan. SATPLAN06 [91], a SAT-based planner, represents current state-of-the art in this area. Conversely CPT[8] [165] and TP4[9] [77] can optimise makespan when operators have arbitrary durations. TP4 is from the HSP* family.

The new partial order planning algorithm presented in this thesis can be applied to synthesise a solution plan with optimal makespan, where operators have *arbitrary* durations.

5.3.2 Probabilistic Action Effects

We previously made the assumption that actions have deterministic effects. Determinism assumes that the world evolves in a fully predictable manner. In reality an action may have

[7] http://www.ida.liu.se/~pahas/hsps/
[8] http://www.cril.univ-artois.fr/~vidal/#cpt
[9] http://www.ida.liu.se/~pahas/hsps/

several different possible outcomes, none of which are certain, and an action may fail before completing. This can be modeled by a simple extension to the current representation. Instead of a single possible set of effects, a probabilistic operator has multiple sets of possible effects. We will refer to each of these sets as an *outcome*. Each outcome has a certain probability of occurrence, with the probabilities of all outcomes for a given action summing to one.

We shall refer to planning with probabilistic operators as probabilistic planning. Probabilistic planning aims to generate a conditional plan with an optimal or satisfactory probability of achieving the goal propositions, called the *probability of success*. A conditional plan is one that says "in this state, do this", i.e. it accounts for the fact the world state can not be predicted. The main approaches to probabilistic planning include:

- ◇ Casting the problem as a Markov Decision Process (MDP) and employing MDP analysis techniques to synthesise a plan;
- ◇ Planning based on model checking, using probabilistic extensions of some of the techniques described in Chapter 2; and
- ◇ Extending techniques for classical planning to deal with uncertainty. For example the Buridan planner [102] uses UCPOP techniques to put candidate plans together and assess them; Paragraph, a planner developed by Little and Thiébaux, addresses concurrent probabilistic planning problems in the Graphplan framework [108].

Probabilistic planning leads to more robust solution plans, but this comes at a great cost. Algorithms need to efficiently analyse all possible action outcomes, and generate a conditional plan. In attempt to avoid this, *replanning* has recently arisen as a seemingly effective way to deal with a non-deterministic planning world. Replanning involves generating a plan using a deterministic variant of the probabilistic problem, executing the plan until an unexpected outcome is observed, and then planning again from this state. There are two common approaches to making a deterministic variant of the probabilistic problem:

1. For each probabilistic action, ignore all but the most probable outcome. The drawback with this approach is that a the goal may become unreachable.
2. For each probabilistic action, create one deterministic action per outcome. This preserves all possible paths in the search space. The downfall is an increase in the number of actions.

In the Probabilistic Track of the 2004 International Probabilistic Planing Competition (IPC-04)[10] the winner was FF-REPLAN-4[11] [173], a replanner which compiles the problem to a deterministic variant using the first approach above, and employs the deterministic FF planner to plan and replan as necessary. FF-REPLAN-5, which uses the second approach above, then (unofficially) outperformed all the "real" probabilistic planners in the Probabilistic track of the following International Planning Competition, IPC-06[12].

The new partial order planning algorithm presented in this thesis can be applied to respond to probabilistic action effects using the second deterministic translation described above and planning/ replanning for the most likely path.

5.4 Conclusion

This chapter introduced the problem of automated planning in general, and the classical planning problem in particular. It was shown that the reachability problem for a restricted state transition system is equivalent to the classical planning problem, thus consolidating the connection between Part I and Part II of this thesis. After formalising the grounded classical planning problem, we surveyed the main approaches to plan synthesis and the issues that have driven the focus of planning research. We examined these techniques from the perspective of the search space, the level of commitment made when traversing the search space, and the structure of solution plans (e.g. totally ordered, partially ordered or parallel). Finally we looked at extensions to the classical planning problem which consider the cost of actions and probabilistic effects.

For more information on classical planning, Hendler *et al* [81] provide an overview of the first thirty years of research in AI planning. This includes a formulation of the planning problem similar to what is now considered the classical planning problem, and a chronology of the development of planning systems during this period. As mentioned, the dominant methods of analysis were means-end analysis (i.e. a form of state space search) and partial order planning; further to this, Weld [169] gives a dedicated introduction to partial order planning. Weld [170] describes the subsequent advances in AI planning: the planning graph and planning as a propositional satisfiability problem. There is a wide range of literature on CSPs. One concise summary of CSPs, including formulation and search techniques, can be found in [147]. The relevance of CSPs to planning is described in particular in [67] and

[10] http://ls5-web.cs.uni-dortmund.de/~edelkamp/ipc-4/
[11] http://www.public.asu.edu/~syoon10/ffreplan.html
[12] http://www.plg.inf.uc3m.es/icaps06/competition.htm

[163]. Bonet and Geffner *et al* [19, 18] describe the advancement of planning as heuristic search. Finally, Ghallab, Nau and Traverso [67] provide a comprehensive summary of AI planning - the problems, techniques and applications - including a detailed and up-to-date expose of approaches to the classical planning problem.

In the next chapter we illustrate how a solution plan can be synthesised via unfolding using the *ERV-Fly* algorithm. In particular, we apply the theory of directed unfolding to find partially ordered solution plans that are optimal with respect to their additive or parallel cost, trade optimality for efficiency to synthesise plans which are suboptimal with respect to additive cost, and address probabilistic planning via replanning for the most likely path.

Chapter 6

Planning Via Directed Unfolding

Petri nets are traditionally used for modelling and analysing distributed systems. As pointed out by Drummond [45], there is a close connection between Petri nets and classical planning models. Furthermore, the type of reasoning performed when analysing a Petri net is closely related to planning. This was exploited, for example, by Godefroid and Kabanza [69] to avoid considering all possible interleaves of actions in reactive planning, by Fabiani and Meiller [61] to avoid the explicit consideration of certain type of mutexes when constructing planning graphs, and by Silva, Castilho and Künzle [153] to recast plan extraction as a Petri net reachability problem.

Nevertheless, the connections between Petri nets and planning have not been developed far yet. Godefroid and Kabanza [69] perform a forward state space search which utilises the concurrency semantics of Petri nets to implement a form of Partial Order Reduction, thus limiting exploration to just one interleaving of concurrent actions. Meiller and Fabiani [61] use coloured Petri nets to implement a multi-valued version of the planning graph, merely obviating the need to explicitly consider certain types of permanent mutexes. Silva *et al* [153] translate a planning graph [17] into a Petri net, then apply integer programming (IP) methods to find a solution plan via reachability analysis.

Complementing a recent effort by Edelkamp and Jabbar to apply planning algorithms based on heuristic search to the analysis of Petri nets (the deadlock detection problem) [49], we contribute to bridging this gap by applying Petri net analysis techniques to planning.

This chapter applies the theory of directed unfolding to classical planning. The first two sections describe how to cast a classical planning problem as a PT-net $\text{REACHABILITY}_{\Sigma}$ problem. This involves translating the statement of a grounded classical planning problem to a PT-net system, by working through the challenges of maintaining logical consistency and representing operators' negative preconditions and effects. It is a small step from here

to formulate a classical planning problem as a Petri net REACHABILITY$_\Sigma$ problem. The *ERV-Fly* algorithm can then be used to synthesise a solution plan. In particular the theory of directed unfolding can be applied to find an optimal plan with respect to various cost criteria. We also consider the limitations of this translation and characterise the extent to which the notion of concurrency in the PT-net we obtain matches the independence based notion of concurrency commonly used in planning.

In the third section we compare and contrast planning via unfolding with the planning methods presented in the previous chapter. Of particular interest is the difference in the level of commitment made during state space planning, plan space planning and planning via unfolding.

Finally we present empirical results for the PUP SUITE a collection of planners which implement our approach to planning via directed unfolding.

6.1 Translating a Planning Problem to a PT-Net System

In this section we propose a translation from the statement of a classical planning problem $\mathbf{P_s} = \langle V, I, A, G\rangle$, to a PT-net system $pnet(\mathbf{P_s}) = \langle N, M_0\rangle$ with $N = \langle P, T, F\rangle$, such that the problem of finding a solution plan for $\mathbf{P_s}$ can then be cast to a problem of REACHABILITY$_\Sigma$ for $pnet(\mathbf{P_s})$.

We first informally depict the dynamics of a classical planning world defined by V and A, using the PT-net construct N. The set of places is $P = V$, i.e. each place $p \in P$ maps one-to-one to a proposition in the set V. For our purposes, the presence of a token in a place indicates the associated proposition is true. There set of transitions is $T = A$. That is, each transition $t \in T$ maps to an operator $o = \langle P_{re}, E_{ff}\rangle \in A$; the sets of places $P_{re} \cap V$ and $E_{ff} \cap V$ form the preset and postset of t respectively. Observe that we have not included the *negated* propositions in P_{re} and E_{ff}. In this way, t is enabled when the positive preconditions of o hold, i.e. $M(v) \geq 1\ \forall v \in P_{re} \cap V$. Furthermore, executing t causes the positive effects of o to become true and the positive preconditions to become false, by inserting and removing tokens in the postset and preset places respectively. Recall that the semantics of planning operators is such that a literal in the set of preconditions, which does not appear in the set of effects, is assumed to remain true. So let the postset of t also include the set of places $(P_{re} \setminus \overline{E_{ff}}) \cap V$.

This model raises the following questions:

1. *How to maintain logical consistency?*

   In this model, the presence of one or more tokens in a place represents the truth of the associated proposition, and exactly no tokens represents its falsity. Consider an operator $o = \langle P_{re}, E_{ff} \rangle$, such that $v \in E_{ff} \cap V$ and $\neg v \notin P_{re}$. That is, v is a positive effect, but not a negative precondition. When the transition $t = o$ is executed, it will put a token in the place v regardless of whether it contains one already. Multiple tokens can thus accumulate in a single place. Consequently, we must ensure that either:

   (a) A transition t can be enabled by at least one token in each of its preset places, and the execution of t causes *all* tokens in the set of preset places to be absorbed; or

   (b) A place never contains more than one token.

   The semantics required for (a) are not provided by PT-nets. Whilst it is possible to extend a PT-net to include a weight function that associates an arc from place p to a transition t with an integer weight w (see [144, 126]), the value w jointly indicates the number of tokens which must exist in p to enable t and the number of tokens that must be removed from p upon execution of t respectively[1]. Furthermore w can not be a variable. So, we must instead ensure condition (b). To do this we must look at the PT-net system, not simply the PT-net; for now we will just assume that the initial marking has no more than one token in a place. Essentially we want to ensure the PT-net system is 1-safe. This could be achieved using Boolean arithmetic, which is supported by a class of more expressive Petri nets called Boolean nets (see Kawamoto *et al* [95] for an introduction to Boolean nets). However the unfolding process, which we wish to use for analysis of our Petri net translation of a planning problem, has not yet been extended to such nets. As discussed in Chapter 7, extending the unfolding process and the notion of directed unfolding to higher level nets could be a subject of future work. Here, we choose instead to ensure 1-safety of the PT-net by making the original planning operators 1-safe. A planning operator $o = \langle P_{re}, E_{ff} \rangle$ is 1-safe if and only if $\overline{E_{ff}} \subseteq P_{re}$. This means a proposition can only be made true if it was previously false (and can only be made false if originally true). Note, this solution assumes that we can model or eliminate negative preconditions and negative effects.

2. *How to model the negative preconditions of an operator?*

   Currently, if the net has marking M then $M(v) = 0$ indicates proposition v is false.

[1] The definition of a PT-net given in Chapter 3 assumes every arc has a weight of one.

Consider an operator $o = \langle P_{re}, E_{ff} \rangle$ such that $\neg v \in P_{re}$. Enabling the transition $t = o$ depends on the presence, not the lack of, a token in each place in its preset: how can we ensure the precondition $\neg v$ holds in a marking that enables t? We identify two possibilities:

(a) Introduce the construct of *inhibitor arcs*. If a place v feeds into transition t via an inhibitor arc, then t can not be enabled when v contains a token. In this way we can model the requirement of a proposition being false for an operator to be enabled; or

(b) Eliminate negative preconditions from all planning operators in A.

Inhibitor arcs complicate the unfolding procedure, because an occurrence net does not explicitly represent local information regarding the absence of tokens in a place. Kleijn and Koutny [98] propose a modification of the standard unfolding algorithm which provides causality semantics for PT-nets with inhibitor arcs. In doing so they claim the partial order semantics based on causality are not adequate for this case and a notion of step sequences must instead be employed. A step sequence is comparable with the sequence of sets of actions generated by planning as SAT techniques, i.e. parallel plans. We have yet to deduce if/how directed unfolding can be applied in this context; furthermore we wish to avoid the notion of step sequences, as we are interested in using causal partial order semantics for planning.

Consequently we decide to instead remove negative preconditions from the planning operators. This is achieved simply by introducing a set of propositional variables $\hat{V}$ in a one-to-one correspondence with the set V. A given $\hat{v} \in \hat{V}$ is true if and only if the corresponding $v \in V$ is false. Thus the falsity of proposition v can be indicated by the presence of a token in a place mapping to $\hat{v}$ (rather than just the absence of a token in the place mapping to v).

3. *How to model the negative effects of an operator?*

Consider operator $o = \langle P_{re}, E_{ff} \rangle$. The occurrence of transition $t = o$ puts a token in each of its postset places. Currently, the postset of t is $\{E_{ff} \cup \{P_{re} \setminus \overline{E_{ff}}\}\} \cap V$. That is, when o is executed all propositions in the set of positive effects are asserted as true, as are all positive preconditions which do no appear negated in the set of effects. Further to this we need to remove tokens from the set of places $E_{ff} \cap \overline{V}$, i.e. model the fact that all propositions which appear negated in the set of effects become false when t occurs. PT-net semantics are such that the only way to remove a token from place p through the occurrence of transition t is to include p in the preset of t. If we do this however,

then p must necessarily contain a token for t to be enabled. A solution to this problem is provided by our solution to issue (1), i.e. making the planning operators 1-safe. Consider an operator $o = \langle P_{re}, E_{ff} \rangle$ with $\neg v \in E_{ff}$. From this we will derive at least one 1-safe operator $o^1 = \langle P^1_{re}, E^1_{ff} \rangle$ such that $\neg v \in E^1_{ff}$ and (by definition of 1-safety) $v \in P^1_{re}$. Thus, upon execution of the transition mapping to this operator, the token will be appropriately removed from the place mapping to v. Obviously this solution also depends on our ability to model (or in our case remove) negative preconditions, as we will also derive from o at least one 1-safe operator $o^2 = \langle P^2_{re}, E^2_{ff} \rangle$ such that $\neg v \in P^2_{re}$ (and $\neg v \notin E^2_{ff}$).

Considering our solutions to the aforementioned issues, our translation operates in three steps. The first two steps involve mapping the planning problem to an equivalent one where every operator is (1) 1-safe and (2) has no negative preconditions. In the third step, the resulting problem is mapped to a PT-net system. We now formalise the translation via these three steps, and prove equivalence with the original problem.

6.1.1 Establishing 1-safe Planning Operators

An operator $o = \langle P_{re}, E_{ff} \rangle$ is *1-safe* if and only if $\overline{E_{ff}} \subseteq P_{re}$. This means that every literal in the operator's effects is necessarily false in any state satisfying the operator's preconditions. This can be achieved by including in the preconditions the complement of every literal appearing in the effects. An "unsafe" operator can thus be converted into a collection of safe ones, by creating a new instance of the operator per combination of literals missing from the preconditions, and removing, in this new operator, any effect literal contained in the extended precondition. Formally, each operator $o = \langle P_{re}, E_{ff} \rangle$ is replaced with a set of operators $\chi^1(o)$ of cardinality $|2^{(E_{ff} \setminus \overline{P_{re}})}|$. Let $\xi = 2^{(E_{ff} \setminus \overline{P_{re}})}$ denote the class of all subsets of $E_{ff} \setminus \overline{P_{re}}$. If $\xi = \emptyset$ then we set $\chi^1(o) = \{o\}$. Otherwise, for each $D \in \xi$ we define a new element of $\chi^1(o)$ by $o' = \langle P'_{re}, E'_{ff} \rangle$, where

$$P'_{re} = P_{re} \cup \overline{D} \cup \left((E_{ff} \setminus \overline{P_{re}}) \setminus D \right) \text{ and}$$

$$E'_{ff} = D \cup (E_{ff} \cap \overline{P_{re}}).$$

D is the set of actual effects, per case.

Example

Consider, for example, the operator $o = \langle P_{re}, E_{ff} \rangle$ where $P_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}\}$ and $E_{ff} = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$. This operator is replaced with the four 1-safe operators $o_i = \langle P^i_{re}, E^i_{ff} \rangle$ given below along with the respective values for D^i.

$P_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}\}$	$E_{ff} = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$	
$P^1_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \neg\mathsf{e}\}$	$E^1_{ff} = \{\neg\mathsf{a}, \mathsf{b}\}$	$D^1 = \{\}$
$P^2_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \neg\mathsf{d}, \neg\mathsf{e}\}$	$E^2_{ff} = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}\}$	$D^2 = \{\mathsf{d}\}$
$P^3_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$	$E^3_{ff} = \{\neg\mathsf{a}, \mathsf{b}, \neg\mathsf{e}\}$	$D^3 = \{\neg\mathsf{e}\}$
$P^4_{re} = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \neg\mathsf{d}, \mathsf{e}\}$	$E^4_{ff} = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$	$D^4 = \{\mathsf{d}, \neg\mathsf{e}\}$

Equivalence

Each new operator is clearly 1-safe since $\overline{E'_{ff}} = \overline{D} \cup (\overline{E_{ff}} \cap P_{re}) \subseteq P'_{re}$. We now need to ascertain that the planning problems specified by $\mathbf{P_s} = \{V, A, I, G\}$ and $\mathbf{P'_s} = \{V, \bigcup_{o \in A} \chi^1(o), I, G\}$ have the same set of solution plans. First, we propose that operator o is applicable in state ϕ if and only if there exists exactly one operator in $\chi^1(o)$ which is applicable in ϕ, and the application of either operator will result in the same world state.

Proposition 6.1.1 (Equivalence of o and $\chi^1(o)$)**.** *Let o be a grounded planning operator over V. Then*

(1) *o is applicable to state $\phi \in S_V$ if and only if there exists exactly one operator $o' \in \chi^1(o)$ which is applicable to ϕ.*

(2) *If $o' \in \chi^1(o)$ and o are applicable to ϕ, then $\gamma_o(\phi, o') = \gamma_o(\phi, o)$. That is, the same state will result if either o or o' is applied in ϕ.*

Proof.

(1) Consider the operator $o = \langle P_{re}, E_{ff} \rangle$ over V. We show that $P_{re} \subseteq \phi \Leftrightarrow$ there exists exactly one $o' = \langle P'_{re}, E'_{ff} \rangle \in \chi^1(o)$ such that $P'_{re} \subseteq \phi$.

($\Rightarrow$) Let D be the maximal set in $\xi = 2^{(E_{ff} \setminus \overline{P_{re}})}$ such that $D \cap \phi = \emptyset$. Thus $\overline{D} \subseteq \phi$. As D is maximal, all other literals in $E_{ff} \setminus \overline{P_{re}}$ must be in ϕ, i.e. $(E_{ff} \setminus \overline{P_{re}}) \setminus D \subseteq \phi$. Thus $P_{re} \subseteq \phi$ implies $P'_{re} = P_{re} \cup \overline{D} \cup \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big) \subseteq \phi$. Note this reasoning still holds if $D = \emptyset$. Furthermore, no other operator in $\chi^1(o)$ is applicable to ϕ. To see

this, consider operator $o'' = \langle P''_{re}, E''_{ff} \rangle \in \chi^1(o)$ defined by some $D'' \in \xi$ which is not the maximal set. Then, there must be at least one literal in $(E''_{ff} \setminus \overline{P''_{re}}) \setminus D''$ which is not in ϕ. In which case $P''_{re} \nsubseteq \phi$.

($\Leftarrow$) The reverse implication is trivial since $P_{re} \subseteq P'_{re}$.

Thus, o is applicable to state ϕ if and only if there exists exactly one operator $o' \in \chi^1(o)$ which is applicable to ϕ.

(2) Suppose the current state is ϕ and the only operator in $\chi^1(o)$ to be enabled is $o' = \langle P'_{re}, E'_{ff} \rangle$, where $P'_{re} = P_{re} \cup \overline{D} \cup \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big)$ and $E'_{ff} = D \cup (E_{ff} \cap \overline{P_{re}})$. We now show that state $\psi = \gamma_o(\phi, o)$ is equal to state $\psi' = \gamma_o(\phi, o')$. The following proof is based on correspondence with White [105], with some minor ammendements for clarity.

$$
\begin{aligned}
\psi &= E_{ff} \cup (\phi \setminus \overline{E_{ff}}) \\
&= (E_{ff} \setminus \phi) \cup (\phi \setminus \overline{E_{ff}}) \text{ (since } E_{ff} \text{ is consistent)} \\
&= (E'_{ff} \setminus \phi) \cup \big((E_{ff} \setminus \phi) \setminus (E'_{ff} \setminus \phi)\big) \cup (\phi \setminus \overline{E_{ff}}) \text{ (since } E'_{ff} \subseteq E_{ff}) \\
&= (E'_{ff} \setminus \phi) \cup \big((E_{ff} \setminus E'_{ff}) \setminus \phi)\big) \cup (\phi \setminus \overline{E_{ff}}); \\
\psi' &= (E'_{ff} \cup (\phi \setminus \overline{E'_{ff}}) \\
&= (E'_{ff} \setminus \phi) \cup (\phi \setminus \overline{E'_{ff}}) \text{ (since } E'_{ff} \subseteq E_{ff} \Rightarrow E'_{ff} \text{ is consistent)} \\
&= (E'_{ff} \setminus \phi) \cup \big((\phi \setminus \overline{E'_{ff}}) \setminus (\phi \setminus \overline{E_{ff}}\big) \cup (\phi \setminus \overline{E_{ff}}) \text{ (since } (\phi \setminus \overline{E_{ff}} \subseteq (\phi \setminus \overline{E'_{ff}})).
\end{aligned}
$$

Thus we see that

$$\psi = \psi' \Leftarrow \big((E_{ff} \setminus E'_{ff}) \setminus \phi\big) = \big((\phi \setminus \overline{E'_{ff}}) \setminus (\phi \setminus \overline{E_{ff}})\big)\,.$$

We now prove the latter assertion. Indeed we prove more; we show that

$$\big((E_{ff} \setminus E'_{ff}) \setminus \phi\big) = \big((\phi \setminus \overline{E'_{ff}} \setminus (\phi \setminus \overline{E_{ff}}\big) = \emptyset\,,$$

and thus ii follows that

$$\psi = \psi' = (E'_{ff} \setminus \phi) \cup (\phi \setminus \overline{E_{ff}}).$$

Firstly, since $P'_{re} \subseteq \phi$,

$$\begin{aligned}\big((E_{ff} \setminus E'_{ff}) \setminus \phi\big) &\subseteq \big((E_{ff} \setminus E'_{ff}) \setminus P'_{re}\big)\\ &\subseteq (E_{ff} \setminus E'_{ff}) \setminus \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big)\\ &= \big(E_{ff} \setminus \big(D \cup (E_{ff} \cap \overline{P_{re}})\big)\big) \setminus \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big)\\ &= \big(\big(E_{ff} \setminus (E_{ff} \cap \overline{P_{re}}) \setminus D\big)\big) \setminus \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big)\\ &= \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big) \setminus \big((E_{ff} \setminus \overline{P_{re}}) \setminus D\big)\\ &= \emptyset.\end{aligned}$$

Now consider

$$\begin{aligned}\big((\phi \setminus \overline{E'_{ff}}) \setminus (\phi \setminus \overline{E_{ff}})\big) &= (\phi \cap \overline{E_{ff}}) \setminus \overline{E'_{ff}}\\ &= \emptyset \Leftrightarrow \overline{E'_{ff}} = (\phi \cap \overline{E_{ff}}) \text{ (since } \overline{E'_{ff}} \subseteq \overline{E_{ff}}).\end{aligned}$$

Now, noting

$$\begin{aligned}\overline{E_{ff}} &= \big((\overline{E_{ff}} \setminus P_{re}) \setminus \overline{D}\big) \cup \overline{D} \cup (\overline{E_{ff}} \cap P_{re})\\ &= \big((\overline{E_{ff}} \setminus P_{re}) \setminus \overline{D}\big) \cup \overline{E'_{ff}},\end{aligned}$$

we have

$$\begin{aligned}\phi \cap \overline{E_{ff}} &= \phi \cap \big(\big((\overline{E_{ff}} \setminus P_{re}) \setminus D\big) \cup \overline{E'_{ff}}\big)\\ &= \big(\phi \cap \big((\overline{E_{ff}} \setminus P_{re}) \setminus \overline{B}\big)\big) \cup (\phi \cap \overline{E'_{ff}})\\ &= \emptyset \cup \overline{E'_{ff}} \text{ (since } \big((\overline{E_{ff}} \setminus P_{re}) \setminus \overline{B}\big) \subseteq \overline{P'_{re}} \subseteq \overline{\phi} \text{ and } \overline{E'_{ff}} \subseteq P'_{re} \subseteq \phi)\\ &= \overline{E'_{ff}}\end{aligned}$$

as required.

□

This entails that in any sequence of operators, any $o' \in \chi^1(o)$ can be replaced by o, and o can be replaced by exactly one operator in $\chi^1(o)$. It thus follows that the planning problems specified by $\mathbf{P_s} = \{V, A, I, G\}$ and $\mathbf{P'_s} = \{V, \bigcup_{o \in A} \chi^1(o), I, G\}$ have the same set of solution plans.

Corollary 6.1.2 (Equivalent solution plans under 1-safety mapping)**.** *Let* $\mathbf{P_s} = \langle V, A, I, G\rangle$ *be the statement of a classical planning problem, and let* $\mathbf{P'_s} = \langle V, \bigcup_{o \in A} \chi^1(o), I, G\rangle$*. Then,* $\pi = (o_1, o_2, \ldots o_n)$ *is a solution plan for* $\mathbf{P_s}$ *corresponding to the sequence of states* $(\phi_0, \phi_1, \ldots \phi_n)$ *if and only if* $\pi' = (o'_1, o'_2, \ldots o'_n)$ *is a solution plan for* $\mathbf{P'_s}$*, where* o'_i *is the single member of* $\chi^1(o_i)$ *applicable to state* ϕ_{i-1}*.*

6.1.2 Eliminating Negative Preconditions

In the second step of the translation, a negative precondition $\neg v$ is eliminated in the usual way [65]: by replacing it with a corresponding positive precondition $\widehat{v}$ and forcing $\widehat{v}$ to be true if and only if v is false. For a given set V of state propositions, we introduce the set $\widehat{V} = \{\widehat{v} | v \in V\}$ of new state propositions. The idea is that $\widehat{v}$ is true exactly when v is false. Let χ^+ be the function which maps an operator $o = \langle P_{re}, E_{ff} \rangle$ over V to the operator $\chi^+(o) = \langle P'_{re}, E'_{ff} \rangle$ over $V \cup \widehat{V}$, where

$$P'_{re} = (P_{re} \cap V) \cup \{v \in \widehat{V} | \neg v \in P_{re}\}$$

$$E'_{ff} = E_{ff} \cup \{\neg\widehat{v} | v \in E_{ff} \cap V\} \cup \{\widehat{v} \in \widehat{V} | \neg v \in E_{ff}\}.$$

Example

For example, if operator $o_1 = \langle \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \neg\mathsf{e}\}, \{\neg\mathsf{a}, \mathsf{b}\} \rangle$ then operator $\chi^+(o) = \langle \{\mathsf{a}, \widehat{\mathsf{b}}, \mathsf{c}, \mathsf{d}, \widehat{\mathsf{e}}\}, \{\neg\mathsf{a}, \mathsf{b}, \widehat{\mathsf{a}}, \neg\widehat{\mathsf{b}}\} \rangle$.

Equivalence

Clearly an operator defined by P'_{re} and E'_{ff} has no negative preconditions, since $P'_{re} \subseteq V \cup \widehat{V}$. We now need to ascertain that there is a one-to-one mapping between solution plans for $\mathbf{P_s} = \{V, A, I, G\}$ and $\mathbf{P'_s} = \{V \cup \widehat{V}, \{\chi^1(o) | p \in A\}, I, G\}$.

Since we are extending the original set of world propositions V to include the set $\widehat{V}$, we need to define the notion of a state over $V \cup \widehat{V}$. Let ϕ be a subset of literals in $V \cup \widehat{V} \cup \{\neg v | v \in V \cup \widehat{V}\}$. Then ϕ is a state over $V \cup \widehat{V}$ if and only if:

(a) It is consistent with respect to the complement function: $\phi \cap \overline{\phi} = \emptyset$.

(b) It is fully specified: $|\phi| = |V \cup \widehat{V}|$.

(c) It is consistent with respect to the semantics of V and $\widehat{V}$: $v \in \phi \Leftrightarrow \widehat{v} \notin \phi$.

The first two conditions follow from the definition of a state over V. The third condition follows from the semantics we have associated with $\widehat{V}$ and V. Observe that there is a one-to-one mapping between states in the set S_V and states in the set $S_{V \cup \widehat{V}}$. The state $\phi \in S_V$ is equivalent to the state $\phi' \in S_{V \cup \widehat{V}}$, denoted by $\phi \equiv \phi'$, if and only if $\phi = \phi' \cap (V \cup \overline{V})$.

We can now show that the planning problems specified by $\mathbf{P_s} = \{V, A, I, G\}$ and $\mathbf{P'_s} = \{V \cup \widehat{V}, \{\chi^+(o)|o \in A\}, I, G\}$ have the same set of solution plans. We first propose that operator $o \in A$ is applicable in state $\phi \in S_V$, if and only if operator $o' = \chi^+(o)$ is applicable in state $\phi' \in S_{V \cup \widehat{V}}$, where $\phi \equiv \phi'$. Furthermore, the state resulting from the application of o in ϕ is equivalent to the state resulting from the application of o' in ϕ'.

Proposition 6.1.3 (Equivalence of o and $\chi^+(o)$)**.** *Let ϕ be a state over V and let ϕ' be a state over $V \cup \widehat{V}$ such that $\phi \equiv \phi'$. Let $o = \langle P_{re}, E_{ff}\rangle$ be a planning operator over V, with $\chi^+(o) = o' = \langle P'_{re}, E'_{ff}\rangle$. Then*

(1) $P_{re} \subseteq \phi \Leftrightarrow P'_{re} \subseteq \phi'$*; and*

(2) $\gamma_o(\phi, o) \equiv \gamma_o(\phi', o')$.

Proof.

(1) We first prove that $P_{re} \subseteq \phi \Leftrightarrow P'_{re} \subseteq \phi'$.

($\Rightarrow$) $\phi \equiv \phi'$ if and only if $\phi = \phi' \cap (V \cup \overline{V})$, therefore $\phi \equiv \phi' \Rightarrow \phi \subseteq \phi'$. Thus $P_{re} \subseteq \phi \Rightarrow P_{re} \subseteq \phi' \Rightarrow P_{re} \cap V \subseteq \phi'$. Furthermore, $\neg v \in P_{re} \Rightarrow \neg v \in \phi'$. From the definition of a state over $V \cup \widehat{V}$ it can be deduced that $\neg v \in \phi' \Leftrightarrow \widehat{v} \in \phi'$. Thus $\{\widehat{v} \in \widehat{V}|\neg v \in P_{re}\} \subseteq \phi'$, and subsequently $P_{re} \subseteq \phi \Rightarrow P'_{re} \subseteq \phi'$.

($\Leftarrow$) From the definition of a state over $V \cup \widehat{V}$, $\neg v \in \phi' \Leftrightarrow \widehat{v} \in \phi'$. This, in conjunction with the definition of P'_{re} tells us $P'_{re} \subseteq \phi' \Rightarrow P_{re} \cap \overline{V} \subseteq \phi'$. Also by definition of P'_{re}, $P_{re} \cap V \subseteq \phi'$. Therefore $P_{re} \cap (V \cup \overline{V}) \subseteq \phi'$, hence $P_{re} \cap (V \cup \overline{V}) \subseteq \phi' \cap (V \cup \overline{V}) = \phi$. So $P_{re} \subseteq \phi$.

(2) We now show that state $\psi = \gamma_o(\phi, o)$ is equivalent to state $\psi' = \gamma_o(\phi', o')$.

$$
\begin{aligned}
\psi &= E_{ff} \cup (\phi \setminus \overline{E_{ff}}) \\
&= E_{ff} \cup \left(\phi' \cap \left(V \cup \overline{V}\right) \setminus \overline{E_{ff}}\right) \\
&= E'_{ff} \cap (V \cup \overline{V}) \cup \left(\phi' \cap \left(V \cup \overline{V}\right) \setminus \overline{E'_{ff} \cap (V \cup \overline{V})}\right) \\
&= E'_{ff} \cap (V \cup \overline{V}) \cup \left(\phi' \cap (V \cup \overline{V}) \setminus \overline{E'_{ff}} \cap (\overline{V} \cup V)\right) \\
&= (E'_{ff} \cup (\phi' \setminus \overline{E'_{ff}})) \cap (V \cup \overline{V}) \\
&\equiv \psi'
\end{aligned}
$$

□

This result leads to the following corollary:

Corollary 6.1.4 (Equivalent solution plans under elimination of negative preconditions)**.** *Let* $\mathbf{P_s} = \langle V, A, I, G\rangle$ *be the statement of a classical planning problem, and let* $\mathbf{P'_s} = \{V \cup \widehat{V}, \{\chi^+(o)|o \in A\}, I, G\}$*. Then,* $\pi = (o_1, o_2, \ldots o_n)$ *is a solution plan for* $\mathbf{P_s}$ *if and only if* $\pi' = (\chi^+(o_1), \chi^+(o_2), \ldots \chi^+(o_n))$ *is a solution plan for* $\mathbf{P'_s}$*.*

6.1.3 Mapping to PT-Net System

We define $\chi(o) \triangleq \{\chi^+(o')|o' \in \chi^1(o)\}$ as the set of operators obtained from the grounded planning operator o by performing the above two steps. Instead of executing the operator o, we can always execute exactly one of the operators in $\chi(o)$ with the same result.

The third step in the translation involves mapping a planning problem to a PT-net system as follows. Let $\mathbf{P_s} = \langle V, A, I, G\rangle$ be the statement of a planning problem. We define a PT-net system $pnet(\mathbf{P_s}) = \langle P, T, F, M_0\rangle$ such that

- ◇ the places are $P = V \cup \widehat{V}$;
- ◇ the transitions are $T = \bigcup_{o\in A}\ \chi(o)$;
- ◇ the set F of arcs is obtained from $t = \langle P_{re}, E_{ff}\rangle \in T$ as

$$\{(v,t) \mid v \in P_{re}\} \cup \{(t,v) \mid v \in E_{ff} \text{ or } v \in P_{re} \text{ and } \neg v \notin E_{ff}\};$$

- ◇ for all $v \in V$, $M_0(v) = 1$ iff $v \in I$ and $M_0(\widehat{v}) = 1$ iff $v \notin I$, and
  for all $v \in V \cup \widehat{V}$, $M_0(v) = 0$ or $M_0(v) = 1$.

If P_s is an extension of the classical planning problem such that each operator $o \in A$ has cost $cost(o)$ then $pnet(\mathbf{P_s})$ is extended to include the cost function $c' : T \to \mathbb{R}^+$ such that $c' \equiv cost$.

Example

Figure 6.1 illustrates this mapping for a single operator, $\mathsf{x} = \langle\{\mathsf{a}, \neg\mathsf{b}\}, \{\neg\mathsf{a}, \mathsf{d}\}\rangle$. Note that x is first cast as two 1-safe operators with no negative preconditions, namely $\mathsf{x'}_1 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \mathsf{d}\}, \{\neg\mathsf{a}, \widehat{\mathsf{a}}\}\rangle$ and $\mathsf{x'}_2 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \widehat{\mathsf{d}}\}, \{\neg\mathsf{a}, \widehat{\mathsf{a}}, \neg\widehat{\mathsf{d}}, \mathsf{d}\}\rangle$.

Correctness

Let us first show that the translation does in fact result in a 1-safe PT-net system which is logically consistent with respect to the semantics of V and $\overline{V}$, i.e. for every marking M reachable from the initial state, $M(v) + M(\widehat{v}) = 1\ \forall v \in V$.

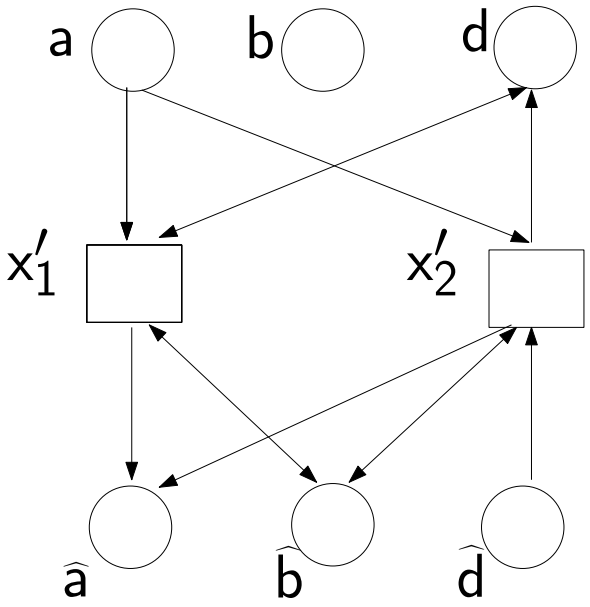


Figure 6.1: The translation of operator $\mathsf{x} = \langle\{\mathsf{a}, \neg\mathsf{b}\}, \{\neg\mathsf{a}, \mathsf{d}\}\rangle$ after transformation into two 1-safe operators with positive preconditions: $\mathsf{x}'_1 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \mathsf{d}\}, \{\neg\mathsf{a}, \widehat{\mathsf{a}}\}\rangle$ and $\mathsf{x}'_2 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \widehat{\mathsf{d}}\}, \{\neg\mathsf{a}\ \widehat{\mathsf{a}}, \neg\widehat{\mathsf{d}}, \mathsf{d}\}\rangle$.

Theorem 6.1.5 (Logical consistency of V and $\widehat{V}$)**.** *Let* $\mathbf{P_s}$ *be the statement of a planning problem. If* M *is a reachable marking of the PT-net system* $pnet(\mathbf{P_s})$ *then* $M(v) + M(\widehat{v}) = 1\ \forall v \in V$.

The following proof includes ideas taken from correspondence with Rintanen [145].

Proof. The proof is by induction in the length i of occurrence sequences leading to M.

Base case $i = 0$: By construction $M(v) + M(\widehat{v}) = 1$ for every $v \in V$.

Inductive case $i \geq 1$: Let M' be any marking that is reached by a sequence of i transitions in which the last transition is t and the second last marking is M. By the induction hypothesis $M(v) + M(\widehat{v}) = 1\ \forall v \in V$.

Let $v \in V$ be any place such that $M(v) = 1$. By the definition of t, if $v \in t^\bullet$ then either $\widehat{v} \in {}^\bullet t$ or $v \in {}^\bullet t$. In the first case there could not have been a token in v (by the induction hypothesis). Firing the transition puts one token into v where there previously was no token, thus $M'(v) + M'(\widehat{v}) = 1$. In the second case, there could not have been a token in $\widehat{v}$ (by the induction hypothesis); firing the transition takes a token from v and then puts it back. Again, $M'(v) + M'(\widehat{v}) = 1$. Similar reasoning can be applied to the situation where $M(\widehat{v}) = 1$ and $\widehat{v} \in t^\bullet$.

Therefore, for every marking M reachable from the initial state, $M(v) + M(\widehat{v}) = 1$ for every $v \in V$. It follows from this that $M(p) \leq 1$ for every place in $V \cup \widehat{V}$. □

It follows from this result that $M(p) \leq 1$ for every place in $V \cup \widehat{V}$. That is, the PT-net system $pnet(\mathbf{P_s})$ is 1-safe.

Now consider the statement of a planning problem $\mathbf{P_s} = \langle V, A, I, G\rangle$. By construction of $pnet(\mathbf{P_s})$, the plan σ is applicable to $\mathbf{P'_s} = \langle V \cup \widehat{V}, \cup_{o\in A}\chi(o), I, G\rangle$ and results in the world state ϕ if and only if σ is an occurrence sequence in $pnet(\mathbf{P_s})$ which leads to marking M, where $M(v) = 1$ if $v \in \phi \cap (V \cup \widehat{V})$ and $M(v) = 0$ otherwise. Then, considering the equivalence between the solutions of $\mathbf{P_s}$ and $\mathbf{P'_s}$ (established in Corollaries 6.1.2 and 6.1.4), we can identify the following relationship between a solution plan for $\mathbf{P_s}$ and an occurrence sequence in $pnet(\mathbf{P_s})$:

Corollary 6.1.6 (Equivalence of solution plan and occurence sequence)**.** *Consider the statement of a classical planning problem $\mathbf{P_s} = \langle V, A, I, G\rangle$ and its translation to PT-net system $pnet(\mathbf{P_s})$. $\pi = (o_1, o_2, \ldots o_n)$ is a solution plan for $\mathbf{P_s}$, corresponding to the sequence of states $(\phi_0, \phi_1, \ldots \phi_n)$, if and only if $\pi' = (o'_1, o'_2, \ldots o'_n)$ is an occurrence sequence in $pnet(\mathbf{P_s})$, corresponding to the markings $(M_0, M_1, \ldots M_n)$, where $M_i(v) = 1$ if $v \in \phi_i \cap V$ and $M_i(\widehat{v}) = 1$ if $v \in \phi_i \cap \overline{V}$, and o'_i is the single member of $\chi(o_i)$ enabled by marking M_{i-1}.*

6.1.4 Limitations of Translation

This translation from a grounded classical planning problem to a PT-net system has two main limitations: the number of operators and propositions defining the problem can increase significantly, and the notion of concurrency is different to the standard notion of concurrency in AI planning.

Size of Translation

An unsafe operator is converted into a collection of safe ones, by creating one "copy" of the operator per combination of values for missing preconditions, and removing, in this new operator, any effect literal contained in the extended precondition. The number of copies thus created is exponential in the number of missing preconditions. After the problem has been made 1-safe, negations are removed following the standard procedure of replacing each negative literal $\neg a$ with a new proposition $\hat{a}$ and modifying all operators so those that make a true make $\hat{a}$ false, and vice versa, thus ensuring that exactly one of a and $\hat{a}$ will hold

in every reachable state. This doubles the number of propositions representing the problem. Note that this does not contribute to the state explosion problem however, since the number of states remains the same.

Future work may consider reducing these increases by reasoning about their necessity in individual problems. For example it may be the case that an operator is safe even though for some of its effects no explicit negation appears in the preconditions. For example a precondition p may be mutually exclusive with effect e, i.e. there is no state in which p and e are both true. Thus it is unnecessary to reason about the truth of e in the preconditions. Also, there may be propositions which are implicitly each others negation, such as "the door is open" and "the door is shut". In such cases it is not necessary to introduce new propositions to represent their negations explicitly.

Notion of Concurrency

We are interested in the notion of concurrency that allows the simultaneous execution of several operators. The question arises as to whether the notion of concurrency inherent in the PT-net obtained by our translation coincides with the standard notion of concurrency in AI planning (as defined by Smith & Weld [155]). It turns out that this is not the case.

The standard notion of concurrency in planning is *independence*: two operators $\langle P_{re1}, E_{ff1}\rangle$ and $\langle P_{re2}, E_{ff2}\rangle$ are independent if and only if $P_{rei} \cap \overline{E_{ffj}} = \emptyset$ and $E_{ffi} \cap \overline{E_{ffj}} = \emptyset$ for $i, j \in \{1, 2\}$ and $i \neq j$. This captures the intuition that they can be executed in any order, yielding the same result in both cases.

Independence does not in general imply concurrency in the PT-net sense. For instance, consider the two independent planning operators $\langle\{\mathsf{a}\}, \{\mathsf{b}\}\rangle$ and $\langle\{\mathsf{a}\}, \{\mathsf{c}\}\rangle$. The corresponding Petri net transitions both take a token from a and therefore cannot occur concurrently.

For PT-nets in general, the converse implication does not hold either, i.e.. in some cases, transitions that could not occur simultaneously in the planning context can be simultaneous. For instance, consider two Petri net transitions t and t' such that ${}^\bullet t = \{\mathsf{a}\}$, $t^\bullet = \{\mathsf{b}\}$, ${}^\bullet t' = \{\mathsf{c}\}$, and $t'^\bullet = \{\mathsf{a}\}$. In markings in which places a and c contain a token these two transitions can occur in any order and concurrently. If these transitions are interpreted as planning operators $\langle\{\mathsf{a}\}, \{\neg\mathsf{a}, \mathsf{b}\}\rangle$ and $\langle\{\mathsf{c}\}, \{\neg\mathsf{c}, \mathsf{a}\}\rangle$, no concurrency is possible because the operators are dependent. However, unlike in the general case, the concurrency relation arising out of our translation is strictly stronger than independence:

Theorem 6.1.7 (Concurrency)**.** *Consider the statement of a classical planning problem* $\mathbf{P_s} = \langle V, A, I, G\rangle$ *and its translation to PT-net system* $pnet(\mathbf{P_s})$*. Let* o_1 *and* o_2 *be operators*

in A. If there are transitions $t_1, t_2 \in pnet(\mathbf{P_s})$ such that $t_1 \in \chi(o_1)$, $t_2 \in \chi(o_2)$ and t_1 and t_2 can occur simultaneously, then o_1 and o_2 are independent (and can be executed simultaneously).

The following proof is taken from correspondence with Rintanen [145], with some minor additions for clarity.

Proof. We prove the result contra-positively, assuming that o_1 and o_2 are not independent and showing that this implies t_1 and t_2 cannot fire simultaneously.

Let $t_1 = \langle P_1, E_1 \rangle$ and $t_2 = \langle P_2, E_2 \rangle$. Assume $o_1 = \langle P_{re1}, E_{ff1} \rangle$ and $o_2 = \langle P_{re2}, E_{ff2} \rangle$ are not independent. We analyse the possible reasons for this:

1. There is $a \in E_{ff1}$ and $\neg a \in P_{re2}$. This implies $\widehat{a} \in P_2$ and either $a \in P_1$ or $\widehat{a} \in P_1$:

   (a) If $a \in P_1$ then $a \in {}^\bullet t_1$ and $\widehat{a} \in {}^\bullet t_2$. By Theorem 6.1.5 $M(\widehat{a}) + M(a) = 1$ so t_1 and t_2 can not be enabled (nor occur) simultaneously;

   (b) If $\widehat{a} \in P_1$ then $\widehat{a} \in {}^\bullet t_1$ and $\widehat{a} \in {}^\bullet t_2$. The transitions can both be enabled, but can not occur simultaneously because $M(\widehat{a}) \leq 1$ by 1-safety (1-safety is entailed by Theorem 6.1.5).

   The case for $\neg a \in E_{ff1}$ and $a \in P_{re2}$ as well as the symmetric case with o_1 and o_2 interchanged are analogous.

2. There is $a \in E_{ff1}$ and $\neg a \in E_{ff2}$. Thus either $\widehat{a} \in P_1$ or $a \in P_1$, and either $a \in P_2$ or $\widehat{a} \in P_2$. Exactly one of the places a and $\widehat{a}$ can have a token, so either only one of the transitions is enabled, or they are both enabled but at most one of them can occur.

   The case for $\neg a \in E_{ff1}$ and $a \in E_{ff2}$ is analogous.

□

This models the semantics of the planning problem faithfully, in the sense that a plan is valid for the planning problem exactly when the corresponding occurrence sequence is a configuration of the Petri net, and the additive cost of the plan is the same. However, the parallel cost will not necessarily be the same, because there are cases where two operators are allowed to execute concurrently according to the planning semantics (as defined by Smith & Weld [155]), but the corresponding transitions in the Petri net are not concurrent. Consequently the causal chains may be different. This happens only when operators share a

precondition which is not negated in either of their sets of effects. An alternative translation which can overcome this problem uses additional places to enable such operators to occur simultaneously, thus preserving the planning semantics [166]. This is beyond the scope of this thesis and may be the subject of future work.

6.2 Planning as Reachability Analysis

We have translated the statement of a planning problem $\mathbf{P_s} = \langle V, A, I, G \rangle$ to a PT-net system $pnet(\mathbf{P_s})$. The problem of finding a solution plan for $\mathbf{P_s}$ can now be cast as a REACHABILITY$_\Sigma$ problem for $pnet(\mathbf{P_s})$ where the set of places we seek to mark is given by:

$$G' = (G \cap V) \cup \{\widehat{v} | v \in G \cap \overline{V}\}$$

We can now employ any of a number of approaches to solving REACHABILITY$_\Sigma$. For the reasons discussed in Chapter 3, our interest lies in on-the-fly reachability analysis via unfolding.

6.2.1 Planning via Directed Unfolding

Having cast a planning problem as a Petri net REACHABILITY$_\Sigma$ problem, we can synthesise a solution plan on-the-fly via unfolding. In particular we can use the theory of directed unfolding to increase efficiency and/or find optimal plans with respect to different criteria. Let $pnet(\mathbf{P_s}) = \langle P, T, F, M_0 \rangle$. To employ *ERV-Fly* (Algorithm 4) to solve REACHABILITY$_\Sigma$ for the PT-net $pnet(\mathbf{P_s})$ and set of places G' we need to define a new PT-net system:

$$\Sigma_{P_s} \triangleq \langle P, T \cup t_R, F \cup_{p \in G'} (p, t_R) \cup (t_R, p), M_0 \rangle$$

This new net defines the REACHABILITY$_\Sigma$ problem for $pnet(\mathbf{P_s})$ and G' (see Definition 6).

To synthesise an optimal solution plan with respect to additive cost, we can employ *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and an admissible heuristic function $h \equiv 0$ or $h^{\max}$. If efficiency is more important than optimality, then an inadmissible heuristic function such as h^{FF} or h^{sum} can instead be used.

To synthesise an solution plan with optimal parallel cost, i.e. minimal makespan, we can employ *ERV-Fly*$(\Sigma_{P_s}, \prec_{f||}, \prec_f)$, with $g \equiv c_{||}$ and an admissible heuristic function $h \equiv 0$ or

h^{par}.

In all cases, a partially ordered plan will be synthesised, as opposed to a totally ordered or parallel plan. In the next chapter we present empirical results for planning via directed unfolding, in each of these contexts.

6.2.2 Probabilistic Action Effects

We want to extend planning via directed unfolding, to problems where actions have probabilistic effects. However the benefits of Petri net unfolding, which are utilised in the deterministic case, can not necessarily be reaped in the probabilistic case. The main properties of Petri net unfolding which are exploited in planning via directed unfolding, are its concurrency semantics and factored state representation.

The concurrency semantics of Petri net unfolding allow us to avoid the arbitrary interleaving of concurrent actions. However when actions have probabilistic effects, the ordering of concurrent actions is not necessarily arbitrary. Whilst action a and action b may be independent, executing one before the other may increase our probability of achieving the goal. Consider the simple situation of two people, with two separate cars, in Alice Springs. The goal is for each person to drive their car along the Stuart Highway to Adelaide. It is realistic to assume the action of driving car a is independent of the action of driving car b, i.e. these are concurrent actions. Now, in a deterministic world we might assume that the action of driving will necessarily move a car (and driver) from one destination to another. Thus the order in which the cars leave Alice Springs is indeed arbitrary. Now, suppose that car a has a notable probability of getting a flat tire, and car b has a spare tire. Whilst the action of car a driving out of Alice Springs is independent of the action of car b driving out of Alice Springs, the order of these actions is definitely not arbitrary. If car a leaves first then there is a higher probability that both cars will reach Adelaide: if car a suffers a flat tire, then car b can provide a spare tire. Consequently, ignoring the ordering of apparently concurrent actions can decrease the probability that our plan with achieve the goal.

Another property of Petri net unfolding which is exploited by planning via directed unfolding, is its factored state representation. The factored state representation enables us to only ever consider the subset of states which are reachable via causally related actions, i.e. the final markings of local configurations. That is, in the deterministic case, we see the world form the perspective of driver a and from the perspective of driver b; it is not necessary to consider the different combinations of their world view. However we have just observed that in probabilistic case, considering the global state (i.e. having both cars in our

viewpoint) can enable us to increase the probability of achieving the goal. Consequently, limiting our consideration to "local" states can decrease the probability that our plan with achieve the goal.

This is just one angle of the complex problem of probabilistic planning with concurrent actions. Haar's work on probabilistic cluster unfolding [75] reflects the fundamental difficulties in consistently defining probabilities on concurrent systems. Indeed, we may decide that there is no omniscient being overseeing car a and car b, and we have no knowledge nor control over the order in which they leave. In this case a lack of information means we are in fact limited to "local" states and this leads to an arbitrary ordering decision where the concurrency semantics of unfolding may perhaps be utilised. The work of Benveniste *et al* [10] is relevant to this later problem. As mentioned in Chapter 2, they propose a structure called Markov Nets , which gives concurrency semantics to Markov Processes. A Markov net is generated by unfolding a Petri net, and propagating probability information such that concurrency implies probabilistic independence. Using their model is similar to having no global state information, i.e. driving car a and driving car b are indeed probabilistically independent.

Due to these complications, real probabilistic planning via directed unfolding is beyond the scope of this thesis. However it is still possible to consider operators with probabilistic effects by casting the problem to a deterministic one and replanning as necessary. The process is as follows:

(1) Split each probabilistic operator into one deterministic operator per possible outcome.

(2) Set the cost of an operator corresponding to outcome x to $-log(pr(x))$, where $pr(x)$ is the probability of outcome x.

(3) Synthesise an optimal or sub-optimal additive cost plan via *ERV-Fly*($\Sigma_{P_s}, \prec_{f+}, \prec_f$), with $g \equiv c_+$ and admissible heuristic function $h \equiv 0$ or $h^{\max}$, or an inadmissible heuristic function h^{FF} or h^{sum}. Minimising the additive cost function consequently maximises the product of the probabilities of action outcomes. That is, a plan which is optimal with respect to additive cost corresponds to the most likely sequence of outcomes.

(4) Replan when an unexpected outcome occurs.

Whilst this solution is not ideal, replanning can in fact be the only option for large problems where analysing all possible outcomes and synthesising a contingency plan, i.e. "real" probabilistic planning, is too computationally demanding.

6.3 Comparison with Classical Planning Methods

Weld comments that "the nature of the space being searched by an algorithm is (somewhat) in the eye of the beholder"[169, p. 33]. He supports this with the example of forward state space search, which could be viewed as searching the plan space by considering a node as the current path. Traversing to a neighbouring node involves appending a new applicable action to the end of the path. So let us begin by comparing the classical approaches to state space planning and plan space planning, with planning via directed unfolding, from the perspective of the plan space. The idea here is similar in spirit to Kambhampati's unifying framework of *refinement planning* [89, 88, 90], however we avoid the level of detail, formality and manipulation required to properly cast planning via directed unfolding as refinement planning.

Nodes in the search space are partial plans, consisting of a set of actions and a set of ordering constraints. Traversing from one node to another involves adding a new action or ordering constraint to the partial plan. As with refinement planning, there are two types of constraints between pairs of actions:

(a) A *precedence constraint* specifies that one action must be applied after another without precluding other actions to come between the two.

(b) A *contiguity constraint* specifies that one action must be applied immediately after the other.

In forward state space planning, a refinement involves adding an action a and a *contiguity* constraint specifying that a must come immediately after the last action added to the plan. In other words, the space of totally ordered plans is being traversed in a forward manner. We could explain backward state space planning analogously.

In both traditional plan space planning and planning via directed unfolding, a refinement involves adding an action a and possibly (but not necessarily) *precedence* constraints between a and other actions in the plan. In traditional plan space planning, a constraint may require a to come *before or after* an existing action. In planning via unfolding, a constraint will only ever require a to come *after* an existing action. Furthermore, when unfolding, a will necessarily be ordered with respect to all other interacting actions. In plan space planning such ordering is generally postponed until it is absolutely necessary. That is, the preset and postset conditions of an event in the unfolding can be considered to play the role of causal links in POCL planners, and an event (which maps to an action) is always included in a manner that does not threaten these causal links. As an aside, following this analogy

further, the complete unfolding reveals all possible ways in which a threat can be resolved. From a purely pragmatic perspective, planning via unfolding provides valuable insight to the planning process. From another perspective, the level of commitment is related to the fact planning via directed unfolding performs a *forward* traversal of the space of partially ordered plans whilst traditional partial order planning algorithms perform a non directional search.

We can conclude that planning via directed unfolding makes *less* commitments than state space planning, and *more* commitments that traditional plan space planning. Thus it is more likely that the goals in a problem will be trivially serialisable when planning via unfolding than state space planning, but less likely than for traditional plan space planning. Conversely, the commitments made when planning via unfolding make it easier to identify when a solution is found than in plan space planning. It is arguably on par with state space planning in this respect. However it is more difficult to perform a plan refinement operation when planning via unfolding than state space planning; the difference with refinement in plan space planning is less apparent and requires more detailed complexity analysis. Kambhampati [90] recommends selecting a planner "with the highest commitment, and with respect to whose class of (sub) plans, most goals in the domain are trivially serialisable". Thus, planning via unfolding provides a different balance on the scales of commitment and trivial serialisability.

In addition, the previous chapter highlighted the significant improvement in the scalability of state space planning, accredited to the power of state based heuristic functions. Traditional plan space planning algorithms can not utilise these heuristics, as their level of commitment and non-directional manner preclude associating a partial plan with a state of the world. Meanwhile, planning via unfolding makes sufficient commitments to map a node in the search space state to a state of the world, and thus *can* utilise the recent development in state based heuristics.

To further understand the spirit of planning via unfolding, one might find it useful to view the PT-net unfolding as a powerful planning graph, where conditions and events play the role of the graph's proposition and action nodes, respectively. There are a number of important differences, however. Firstly, whilst the planning graph performs an approximate reachability analysis, the unfolding computes reachability exactly: a by-product of the Petri net semantics is that all mutexes (not just binary ones) are propagated and accounted for when determining sets of possible events. Secondly, the unfolding duplicates nodes as needed to guarantee post-uniqueness, i.e. that conditions (proposition nodes) have a unique event (action node) as predecessor. A consequence of these differences is that plans can

be extracted from the unfolding in time linear in their length, while plan extraction from the planning graph requires search. Finally, there is no global notion of level in the unfolding. Instead, there is an asynchronous vision of time which confers on independent sub-problems their own local levels. Consequently, the unfolding lends itself more easily to the generation of partially-ordered plans with optimal additive cost, or optimal makespan, while the planning graph is better suited to producing step-optimal parallel plans.

6.4 The PUP SUITE

The PUP SUITE is a collection of planners which translate a classical planning problem $\mathbf{P}_s$ into a Petri net reachability problem defined by Σ_{P_s} and employ the *ERV-Fly* Algorithm to synthesise a solution plan, using the theory of directed unfolding for the purpose of optimisation and/ or efficiency. PUP is an acronym for Petri net Unfolding Planning. The term is similar to POP (Partial Order Planning) in reflection of the fact PUP is a form of partial order planning.

The *Planning Domain Definition Language* (PDDL, see [115] and [3]) unifies the STRIPS representation of classical planning, and a growing range of extensions such as durative actions and actions with probabilistic effects. It was developed primarily to facilitate the International Planning Competitions to be discussed shortly. The continuing extensions to PDDL have a secondary effect of directing the focus of automated planning research.

The first PDDL to PT-net translator, based on the theory presented in this chapter, was written by Sylvie Thiébaux. Thiébaux's translator, called PETRIFY, parses a PDDL *domain* file and PDDL *problem* file, and writes an *ll_net* (lower level net) file. The PDDL domain and problem files together describe the statement of a classical planning problem $\mathbf{P}_s$. The *ll_net* file describes the PT-net representation of $\mathbf{P}_s$, i.e. Σ_{P_s}, in the format required by MOLE. The *ll_net* format [14] was originally proposed by the creators of PEP tool[2] (Programming Environment based on Petri nets), and adopted by the creators of MOLE to facilitate interaction between the two tools. It was necessary for us to extend the original *ll_net* syntax to include a rational cost value in the the attributes of each transition. PETRIFY also parses Probabilistic PDDL (PPDDL, see [174]), which is an extension to PDDL that describes operators with probabilistic effects. In this case, PETRIFY splits each operator into several deterministic ones and maps their cost to the negative log of their probability (as described earlier in this chapter).

[2] `http://theoretica.infomatik.uni-oldenburg.de/~pep/`

Following the development of PETRIFY, Patrik Haslum wrote another PDDL to PT-net translator, also based on the theory in presented in this chapter. Haslum's translator is additionally able to parse the PDDL description of a planning problem with durative actions, mapping the cost of a transition to the duration of the action it represents. With respect to the planning problems considered in this thesis, the choice of translator is arbitrary, except that only PETRIFY processes probabilistic operators and only Haslum's translator handles durative actions.

As mentioned previously, we have extended MOLE to implement the various versions of *ERV-Fly* described in Chapter 4 (and referred to earlier in this chapter with respect to their application to plan synthesis). The PUP SUITE PDDL to PT-net TRANSLATOR incorporates both Thiébaux and Haslum's translators (with only one being applied for a given problem) and our extended version of MOLE, as shown in Figure 6.2. The user must specify whether to direct the unfolding using the additive or parallel cost function, and which heuristic function to employ. We will refer to the options as follows:

- ◇ PUP+ h^0: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and $h \equiv 0$;
- ◇ PUP+ $h^{\max}$: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and $h \equiv h^{\max}$;
- ◇ PUP+ h^{sum}: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and $h \equiv h^{\text{sum}}$;
- ◇ PUP+ h^{FF}: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and $h \equiv h^{\text{FF}}$;
- ◇ PUP|| h^0: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f||}, \prec_f)$, with $g \equiv c_{||}$ and $h \equiv 0$; and,
- ◇ PUP|| h^{par}: PUP SUITE synthesises a solution plan for $\mathbf{P_s}$ via *ERV-Fly*$(\Sigma_{P_s}, \prec_{f||}, \prec_f)$, with $g \equiv c_{||}$ and $h \equiv h^{\text{par}}$.

The output of the PUP SUITE is a partially ordered plan π represented by a set of operators and their ordering constraints. Following this is a mapping *start*: $o \rightarrow \{0\} \cup \mathbb{R}+$, for all $o \in \pi$, where *start*(o) is the time (with the origin at 0) at which operator o must be executed in order to minimise the total execution time of this particular plan. In addition, the length,

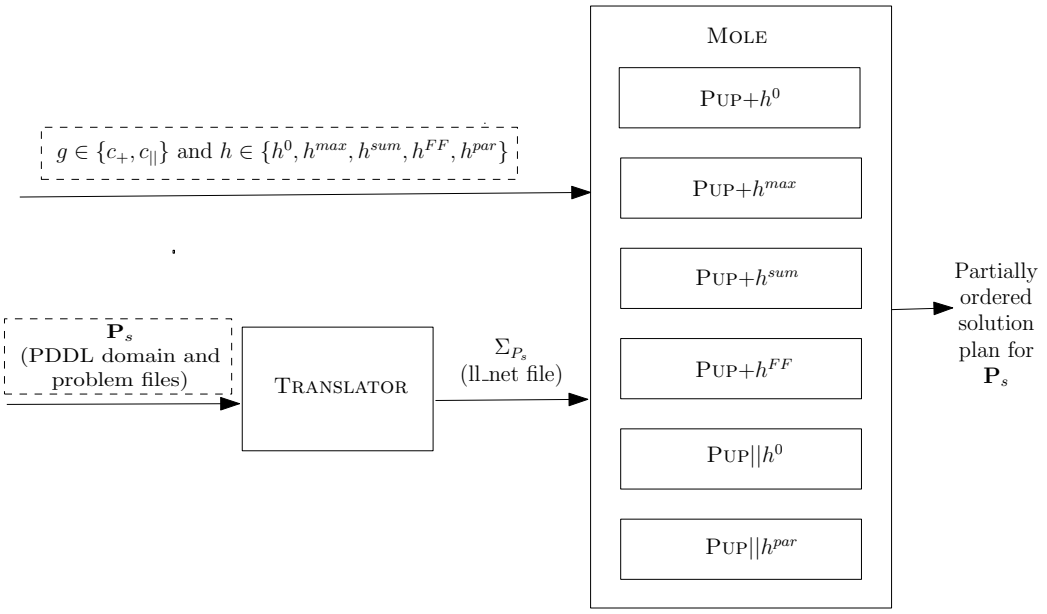


Figure 6.2: The PUP SUITE. User input is displayed inside dashed boxes. The user must provide PDDL domain and problem files describing the statement of a planning problem $\mathbf{P_s}$, and select which cost function g and heuristic function h to use to direct the unfolding. The TRANSLATOR casts $\mathbf{P_s}$ to a PT-net reachability problem defined by Σ_{P_s}, and writes an *ll_net* file describing Σ_{P_s}. The *ll_net* file is read by MOLE. In accordance with the user's specification of g and h, MOLE calls one of the PUP planners to solve the reachability problem defined by Σ_{P_s}, and subsequently prints a partially ordered a solution plan for$\mathbf{P_s}$.

unit-makespan, additive and parallel costs of the plan are all displayed regardless of how the unfolding was directed. Obviously, if actions have unit cost then the length is equal to the additive cost and the unit-makespan is equal to the parallel cost. The plan flexibility (a concept to be defined shortly) is also given.

REPUP is a version of the PUP SUITE that addresses probabilistic operator effects via re-planning. REPUP is a patch over MDPSIM[3], a client/ server style simulator used to evaluate planning systems in the Probabilistic Track of the International Planning Competitions. REPUP calls the translator and MOLE software as appropriate. To begin, REPUP invokes the translator to map the PPDDL description of the planning problem to a PT-net Σ_{P_s}. It then calls on MOLE to run *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$, with $g \equiv c_+$ and h as specified by the user (i.e. $h \equiv 0$, h^{sum}, h^{max} or h^{FF}). The probabilistic operator o corresponding to the first deterministic operator o' in the plan synthesised by MOLE is sent to the MDPSIM server to simulate execution. MDPSIM then indicates the new state s of the world; if s corresponds to the state resulting from execution of o' then the current plan continues to be applied. Otherwise, it must be that the execution of o resulted in a different outcome than expected. REPUP then changes the initial marking of Σ_{P_s} to s and re-calls MOLE to run *ERV-Fly*$(\Sigma_{P_s}, \prec_{f+}, \prec_f)$ in order to synthesise a new plan where the first action will necessarily be applicable to s.

We now present empirical results which illustrate the performance of the PUP SUITE across a range of planning problems.

6.4.1 Artificial Problems

Our first experiment supports the claim that planning via unfolding can be exponentially more efficient than planning via state space search, and that the difference lies in the level of concurrency. Consider an artificially constructed planning problem in which the goal is a conjunction of n sub-goals. The ith sub-goal is achievable by a sequence A_i of length i. The A_is are disjoint. Each action $a_{i,j}$ in A_i has a unique precondition $e_{i,j-1}$ which it negates, and one positive effect $e_{i,j}$, which acts as precondition of the next action $a_{i,j+1}$ and so on. Proposition $e_{i,0}$ is true in the initial state, $e_{i,i}$ is the ith sub-goal, and the $e_{i,j}$ are all different propositions. The degree c of concurrency in the problem varies from 1 (sequential) to n (fully concurrent) by making the ith sub-goal a precondition to the execution of the $i+1$th sequence (i.e., $e_{i,i}$ is a precondition of $a_{i+1,1}$), for $i = c \ldots n-1$.

The graph of Figure 6.3 shows the number of nodes expanded by forward state space search

[3] http://www.ldc.usb.ve/~bonet/ipc5/

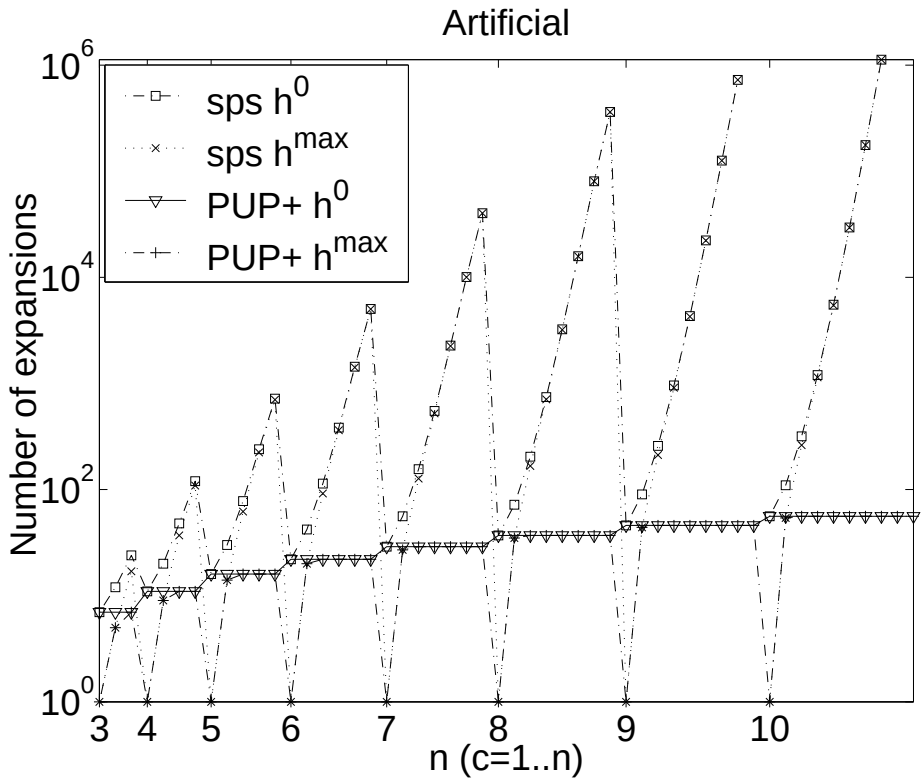


Figure 6.3: Artificial Problem

(sps) and unfolding (PUP+), each using the 0 and h^{max} heuristics, for $n = 3 \ldots 10$ and c varying from 1 to n in each case. To ensure fairness, we report the number of nodes expanded to prove optimality (to prove that there is no solution of cost less than the optimal) rather than to find the optimal solution. The figure clearly shows that, as c increases, the performance of state space search degrades exponentially, while the number of nodes expanded by the unfolding is constant (it equals $n(n+1)/2$, the number of actions in the plan). The h^{max} heuristic makes no significant difference except in the purely sequential case where it enables both techniques to prove optimality without search. State space search fails to solve some of the problems as early as $n = 9$, while unfolding solves all problems of size $n = 100$ (not shown in the figure) in a couple of minutes each, producing plans over 5000 actions long.

6.4.2 IPC Benchmarks

The International Planning Competition (IPC) [4] is a biannual event aimed to assist in benchmarking the latest planners. This empirical evaluation assists in identifying the strengths and weaknesses of various techniques. This comparison is limited however by the particu-

[4] http://ipc.icaps-conference.org

lar selection of benchmark problems. Furthermore, in general, it is difficult to decouple the different causes of complexity in the IPC benchmarks.

We ran STRIPS formulated experiments from IPC-1 (also known as the AIPS-98 planning competition), IPC-2 (the AIPS-00 planning competition), IPC-3, IPC-4 and IPC-5. In domains that fully disallow concurrency, such as the POWER SUPPLY RESTORATION (PSR) domain from IPC-4, and OPENSTACKS from IPC-5, the number of nodes expanded by unfolding and state space search is identical for a given heuristic and so unfolding gives no advantage. In domains that have some degree of concurrency, unfolding scales better than state space search however the later can often be faster since it is more expensive to process a node in the unfolding than in the state space. Unfortunately, in nearly all domains the level of concurrency is insufficient to exploit the benefits of unfolding, and we witnessed the unfolding growing exponentially in the size of the PT-net representation (i.e. the worst case complexity). This problem was often aggravated by the translation of operators to equivalent one-safe ones causing a significant explosion in the size of the PT-net representation. Consequently the PUP SUITE could generally tackle only small IPC problem instances and plan synthesis commonly failed for larger problems due to insufficient memory. Fortunately however there are two benchmark domains with a relatively high level of concurrency, which we can use to illustrate the benefits of planning via unfolding: AIRPORT and PIPESWORLD, both from IPC-4.

The AIRPORT [85] world model captures the topology of an airport relevant to ground traffic; the AIRPORT planning problem is to control the ground traffic such that inbound and outbound planes are sent to appropriate parking and runway positions respectively without endangering each other. Five scaling airport topologies appear in the IPC-4 benchmarks, with the second largest of these corresponding to one half the Munich airport MUC. As would be the case in real airport ground traffic control problems, planes can (and for efficiency should) be moved concurrently. Thus this is an appropriate domain for PUP.

PIPESWORLD [85] models a pipeline network for transporting oil derivative products. The planning problem is to transport products through the pipelines to their appropriate location. Logistical challenges include the fact that pipelines must always be filled with liquid; inserting one product at one end can cause another to come out at the other end, and certain products can not come into contact in the pipe. The IPC-4 benchmarks cover five scaling pipeline topologies. This is an appropriate domain for PUP because products can be put into different pipelines concurrently.

The nature of the operators in both the AIRPORT and PIPESWORLD domains is such that the notion of concurrency in the PT-net generated by our translation is guaranteed to co-

incide with that allowed in the original planning problem. That is, no operators share a precondition which is not negated in either of their sets of effects.

What is clear from the results which follow, is that given a domain with a sufficient level of concurrency, the PUP SUITE is consistently competitive with current state of the art planners with respect to suboptimal and optimal classical planning and optimal temporal planning. That is, it is irrelevant whether the cost function is additive or parallel and whether action costs are unitary or arbitrary positive numbers. This is quite uncommon if we consider other current state of the art planners, and is reflective of the fact the PUP approach sits somewhere between state space and plan space planning thus offering some of the benefits of each.

Translation Time

In the results which follow the *runtime* is the CPU time required for plan synthesis only; it does not include the time taken to translate a PDDL description of a planning problem to a PT-net.

For AIRPORT the translation time increases from 0.02 seconds to 24 seconds, for problems 1 -20. For problems 21 and above, which correspond to half the Munich Airport, the translation time increases from 3 minutes to nearly 1.5 hours. Translating the temporal version of problems 21 and above takes significantly more time, ranging from 1 hour to 50 hours.

For PIPESWORLD the translation time increases from 0.14 seconds to 69 seconds across the problem instances. There is only a slight increase for the temporal version, for which the longest translation time is 89 seconds.

Interestingly, whilst the PT-net representations of AIRPORT are significantly larger than those for PIPESWORLD, PUP's runtimes are generally faster for AIRPORT. We conjecture that the level of concurrency is much higher in AIRPORT, making it more amenable to planning via unfolding. This reflects of the scalability of PUP for highly concurrent domains.

Suboptimal Classical Planning

We first present experimental results for PUP guided by the additive cost function and an inadmissible heuristic (h^{FF} or h^{sum}), to evaluate its qualities as a suboptimal planner.

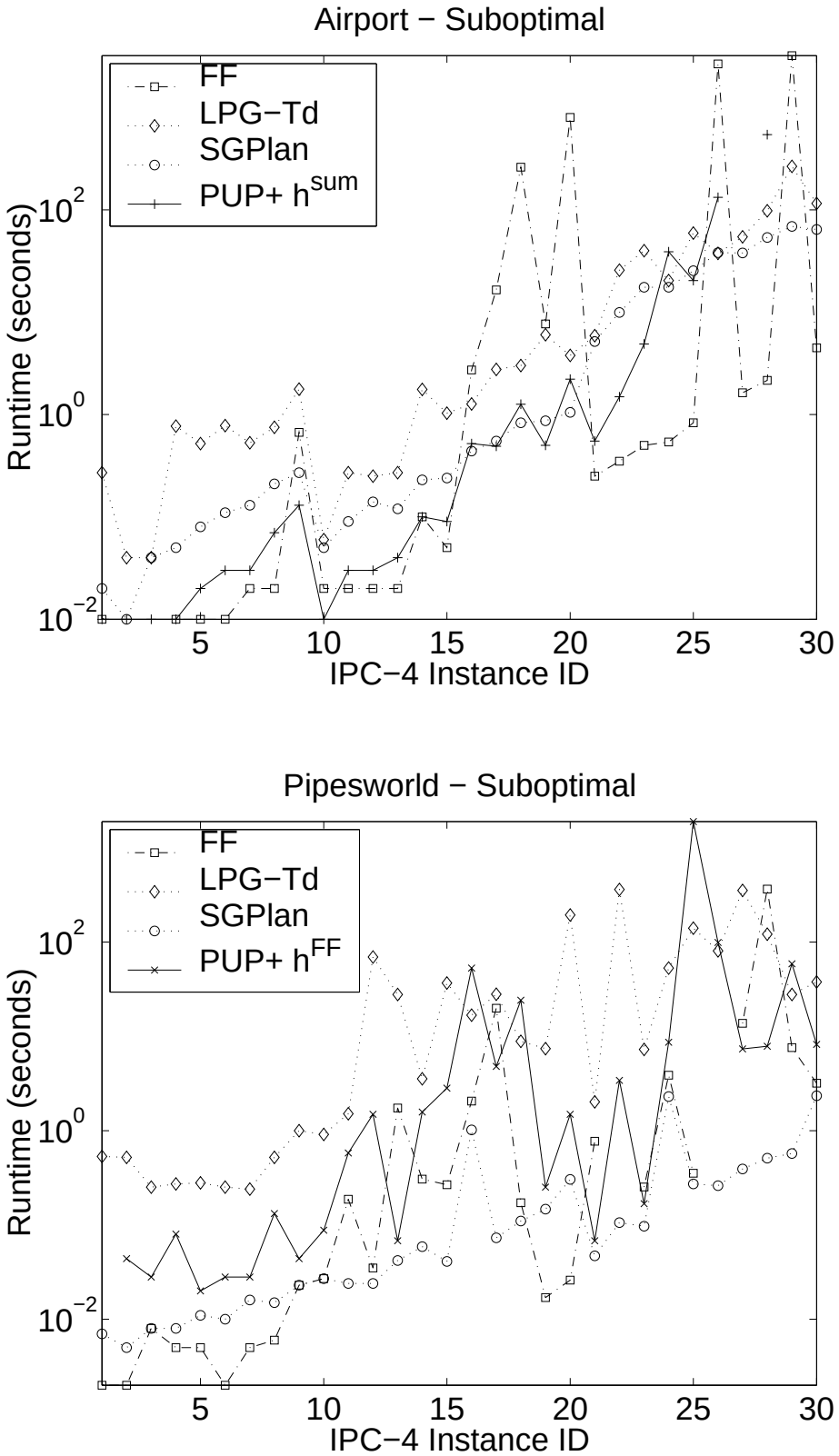


Figure 6.4: Results for AIRPORT and PIPESWORLD - Suboptimal Classical Planning

For comparison, we ran three of the best suboptimal classical planners today: FF[5] [86],

[5] http://members.deri.at/~joergh/ff.html

LPG-TD[6][66] and SGPLAN[7] [37]. FF is basically a variant of HSP, performing a forward state space search guided by the FF heuristic described in Chapter 4. FF was awarded for Outstanding Performance in IPC-2 and Top Performer in the STRIPS track of IPC-3. LPG-TD (Local Planning Graphs) can handle domains with arbitrary action costs and durative actions. Each point in its search space is an *action graph*, which is a sub-graph of the planning graph representing a partial plan. The search traverses from one point to another via graph modifications. LPG-TDś predecessor, LPG, was awarded Distinguished Performance of the first order in IPC-3. SGPLAN partitions the planning problem into sub-problems, each with their own sub-goal. Each sub-problem is hierarchically decomposed and irrelevant actions are eliminated. A modified version of FF is called to solve each bottom level sub-problem. If FF is deemed inappropriate then LPG-TD is called.

The results presented here are for the STRIPS formulations of AIRPORT *non temporal* and PIPESWORLD *notankage-nontemporal* domains. All planners were run on a Pentium M 1.7GHz with a 2GB memory limit and 1 hour time limit. LPG-TD was implemented with the *quality* option; this means it finds a plan then spends a certain amount of CPU time trying to improve it. It would likely be more efficient with the *speed* option, however we wish to also consider plan length.

In the AIRPORT domain, all tested planners synthesised plans of the same length, which also equal the best (i.e. shortest) plan length reported by any suboptimal planner in IPC-4. We conjecture that these problems have a single, obvious solution that all planners find. For clarity we only show the results for PUP+ h^{sum} which performed slightly better than PUP+ h^{FF} on many problem instances. For most problems, PUP is faster than LPG-TD and comparable with SGPLAN, but on larger problems SGPLAN is faster. FF is faster than all three other planners for approximately half the problems, and equally slower for the other half.

Results in PIPESWORLD are more interesting. Again, for clarity, we show the results for just one of the PUP planners. In this case, PUP+ h^{FF}, which slightly outperformed PUP+ h^{sum}. PUP's runtimes are very competitive with LPG-TD and fall short of SGPLAN and FF's (which are quite similar). However PUP finds plans of superior quality, not only to the other planners in this comparison but in many cases shorter than the best result reported by *any* planner in IPC-4. See Table 6.1.

[6] http://zeus.ing.unibs.it/lpg/
[7] http://www.cs.washington.edu/ai/sgp.html

id	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
pup	14	**13**	**18**	20	24	**16**	30	26	**42**	22	30	**26**	**32**	14	**31**	18	**24**	**35**	**39**	**28**	**35**	**33**	**40**
best	**10**	14	20	20	24	20	30	26	70	22	30	28	38	14	35	18	37	42	55	30	47	40	51

Table 6.1: Plan length reported by PUP with h^{FF} vs best length reported by any suboptimal IPC-4 planner (PIPESWORLD)

Optimal Classical Planning

We now consider the PUP SUITE's potential for optimal classical planning. Again we used the STRIPS formulations of AIRPORT *non temporal* and PIPESWORLD *notankage-nontemporal* domains, and all planners were run on a Pentium M 1.7GHz with a 2GB memory limit and 1 hour time limit.

With the exception of HSP$^*_0$ we are not aware of any IPC planners currently capable of optimising the sum of arbitrary action costs. Thus, for fair comparison, we compare the performance of PUP+ h^0 and PUP+ $h^{\max}$ with HSP$^*_0$.

The AIRPORT and PIPESWORLD domains do not differentiate between actions costs. So, we first look at synthesising an optimal plan with respect to length (i.e. the additive plan cost when every action has unit cost). Figure 6.5 shows the runtimes for AIRPORT and PIPESWORLD. In AIRPORT, PUP clearly outperforms HSP$^*_0$. This is not unexpected, since PUP outperformed the heuristic search planner FF in the suboptimal domain. Note that PUP+ $h^{\max}$ consistently outperforms PUP+ h^0 in this domain, so for clarity we do not include the later. In PIPESWORLD the results are more mixed, with PUP+ $h^{\max}$ and HSP$^*_0$ alternately outperforming each other. However both PUP+$h^{\max}$ and PUP+h^0 are able to solve some of the larger problem instances beyond the capability of HSP$^*_0$. These results confer with the suboptimal case, where suboptimal PUP was slightly slower than the heuristic state space planner FF in PIPESWORLD.

Since the PUP SUITE can consider actions with arbitrary costs, we decided to modify PIPESWORLD to differentiate between the cost of operators. We reasoned that it may be more expensive, difficult and/ or dangerous to handle one product compared to another. So, all oil derivative products in a particular problem instance were randomly assigned a rational number between 0.001 and 10. This number represents the cost of pushing a batch of this product into a pipeline. The goal is to transport products to the specified location with minimal additive cost. We ran PUP+ $h^{\max}$, PUP+ h^{sum}, LPG-TD and HSP$^*_0$. Figure 6.6 shows the runtimes (top graph) and additive costs (bottom graph) of the solution plans. We do not include HSP$^*_0$ in the later, since it generates plans with the same cost as those found by PUP+ h^{sum}.

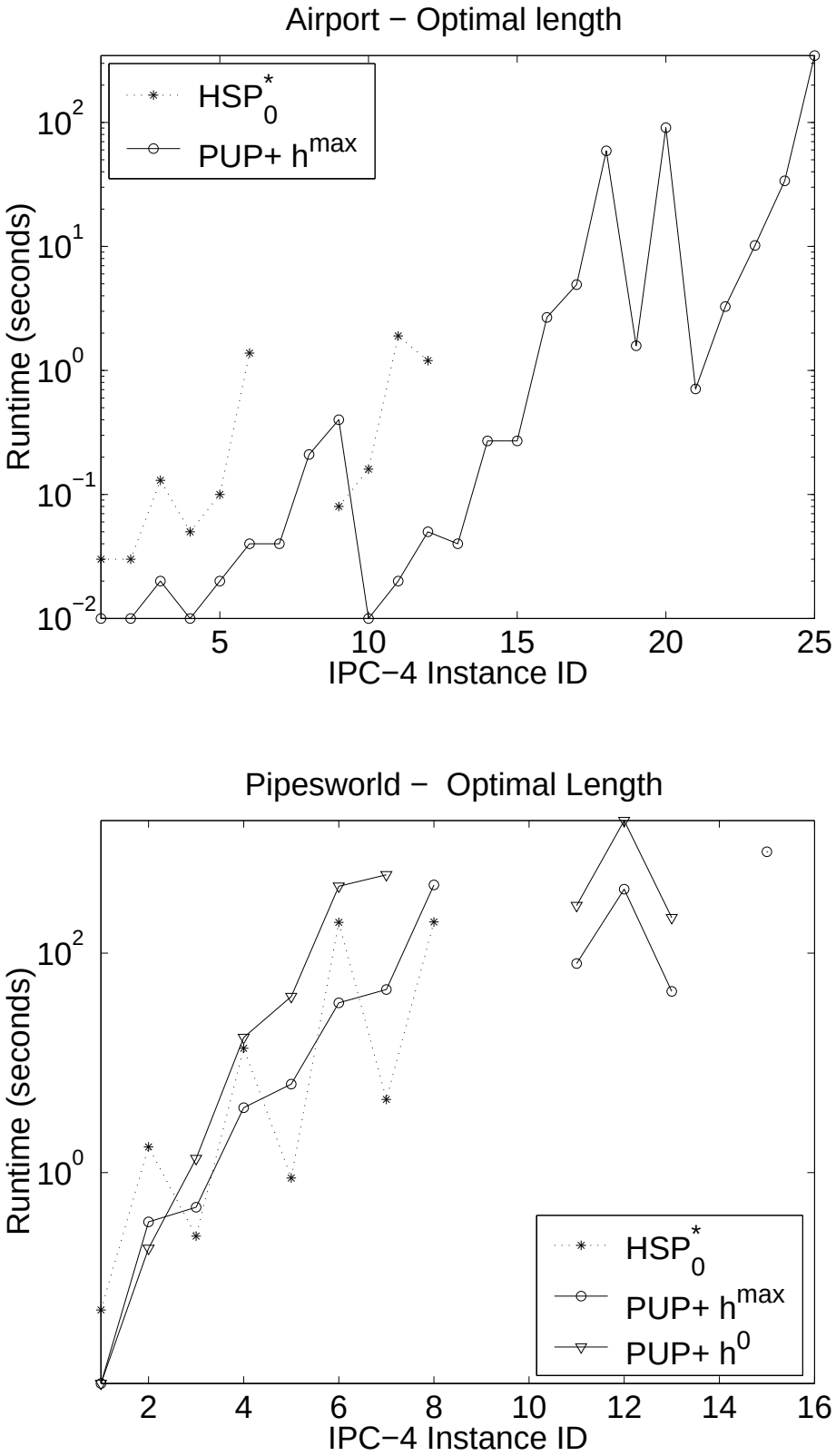


Figure 6.5: Results for AIRPORT and PIPESWORLD - Optimal Classical Planning with respect to Length

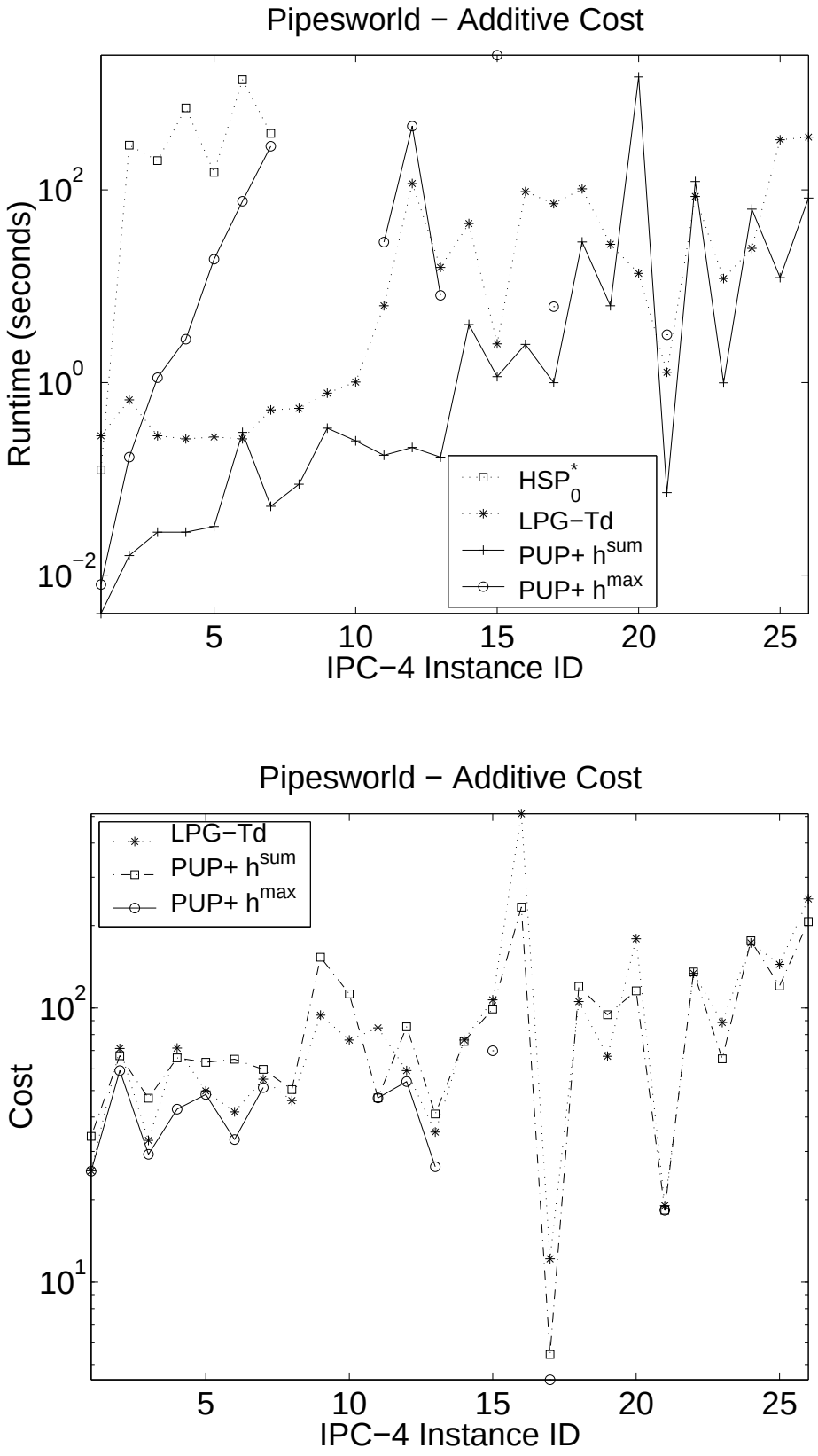


Figure 6.6: PIPESWORLD - Optimising Additive Cost

The two optimal planners, HSP$^*_0$ and PUP+ h^{max}, are able to solve approximately the same subset of problems as for the unit cost case. PUP+ h^{max} is faster than HSP$^*_0$ for most problems, and able to solve some larger problems beyond the reach of HSP$^*_0$. For the first half of the problems PUP+ h^{sum} is faster than LPG-TD however LPG-TD finds plans with lower cost; in a couple of cases LPG-TDś plan cost is optimal. Results for the second half of the problems are more mixed, but the slower planner generally finds the better solution.

Assigning arbitrary costs did not seem appropriate for AIRPORT since the suboptimal results indicate there is a single obvious solution to each of these problems.

We now look at synthesising an optimal plan with respect to its number of parallel plan steps (i.e. the parallel cost of a plan when every action has unit cost). As mentioned previously, by nature CSP based planning techniques are amenable to optimal parallel planning. Thus it is not surprising that the leading planner for such problems is SATPLAN06 [91]. SATPLAN06 ranked joint first for Optimal Planning (Propositional Domains) in IPC-5. Its predecessor SATPLAN04 ranked first in IPC-4. We also ran TP4, an optimal planner from the HSP$*$ family, to provide comparison with state space heuristic search.

The graphs in Figure 6.7 show the run-times of PUP|| h^{par}, SATPLAN06 and TP4 for AIRPORT and PIPESWORLD. As mentioned, in both domains, the concurrency in the net generated by our translation is guaranteed to coincide with that allowed in the original planning problem. Thus, for a given problem, PUP|| h^{par} synthesises a solution plan with the same number of time steps as the plans identified by SATPLAN06 and TP4.

PUP clearly outperforms both SATPLAN06 and TP4 on AIRPORT, solving more problems faster, but is equally clearly outperformed by SATPLAN06 on PIPESWORLD. These results are not surprising: SATPLAN06 is well known to excel on problems with short optimal parallel plans, as is the case for PIPESWORLD, whilst the PUP SUITE can handle longer optimal plans, provided there is a significant level of concurrency in the world model (as appears to be the case for AIRPORT).

We have combined the preceding PUP results for PIPESWORLD to compare its performance for different cost functions. The top graph in Figure 6.8 shows the runtimes for PUP+ h^{max} optimising length and arbitrary additive costs, and PUP|| h^{par} optimising the number of steps. Recall that h^{par} is a temporal adaption of the h^{max} heuristic, so this is a reasonable comparison. Interestingly, the problems solved and the runtimes are very similar. There are some discrepancies in the larger problems, but nothing consistent to indicate PUP optimises one cost function more easily than another. The bottom graph in Figure 6.8 shows the runtimes for PUP+ h^{sum} considering the length and arbitrary additive costs. Up to problem 23 the results are quite similar with just a few outliers. In the larger problems it appears

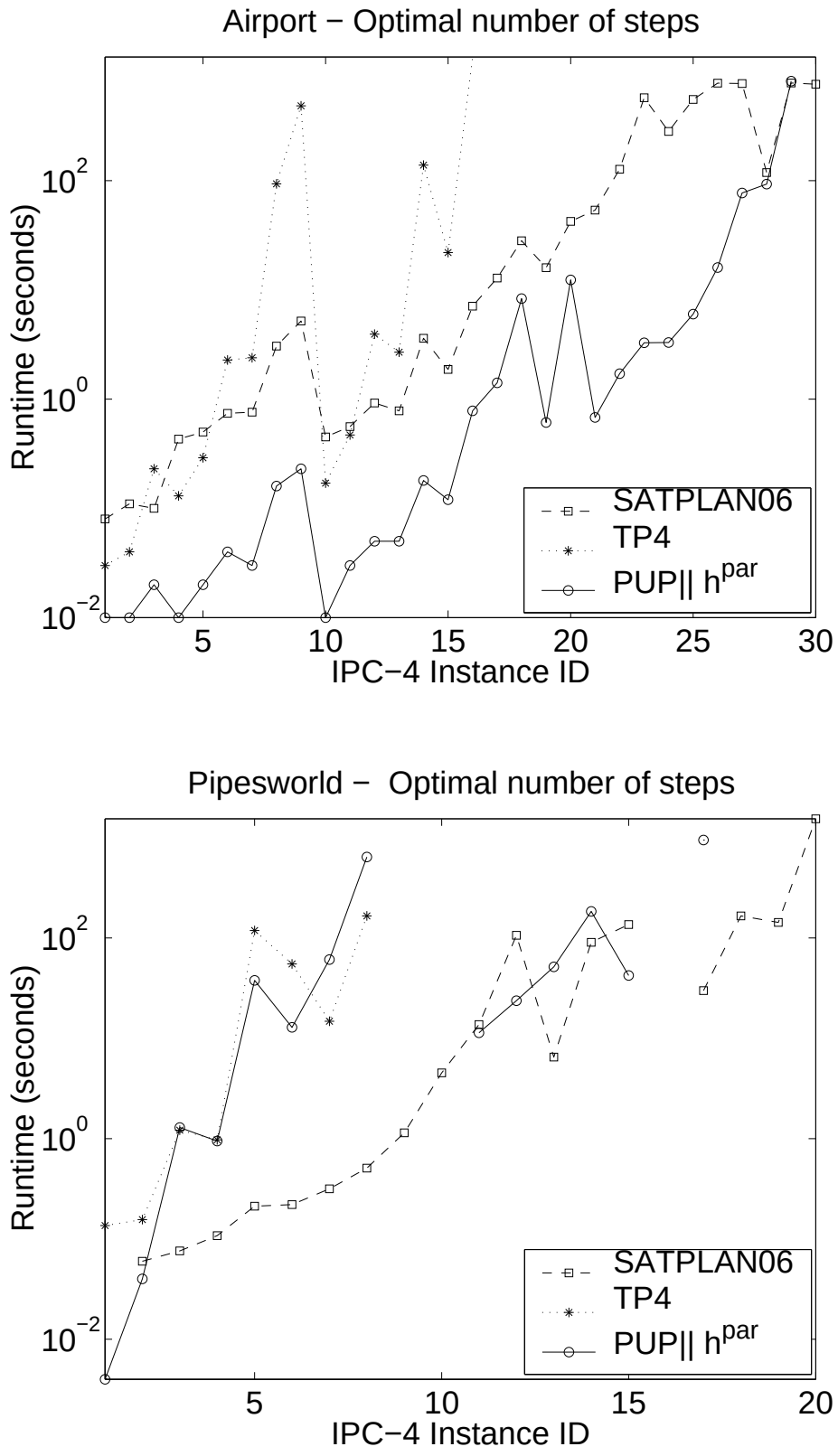


Figure 6.7: Results for AIRPORT and PIPESWORLD - Optimal Classical Planning with respect to number of time steps

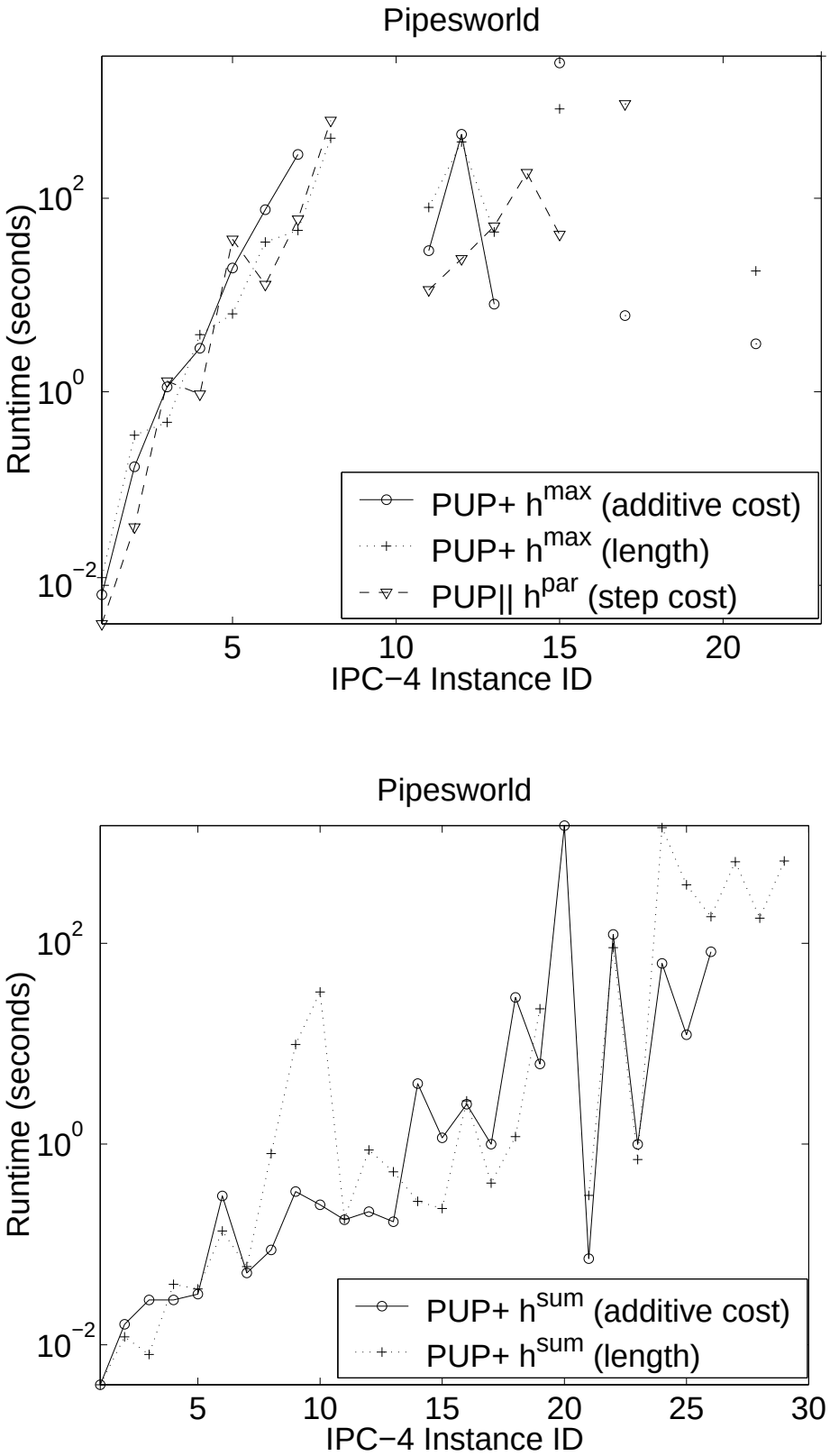


Figure 6.8: PIPESWORLD - Optimising Length vs Additive cost vs Number of steps

easier for PUP to consider plan length rather than arbitrary additive cost. Unfortunately these problems were beyond the reach of the optimal PUP planners so we do not know if a similar pattern would have followed.

Optimal Temporal Planning

We now look at optimal temporal planning, i.e. minimising makespan. We used the STRIPS formulations of AIRPORT *temporal* and PIPESWORLD *notankage-temporal* domains. Again all planners were run on a Pentium M 1.7GHz with a 2GB memory limit and 1 hour time limit.

We compared the temporal version of PUP, employing the h^{par} heuristic, with CPT2, the most recent version of CPT[165]. Like PUP|| h^{par} and PUP|| h^0, CPT2 is optimal with respect to makespan, and is likely the best optimal temporal planner, overall, at the moment. CPT2 was awarded Distinguished Performance in Temporal Domains in IPC-5. CPT2 uses POCL planning together with constraint propagation techniques for pruning in search. For comparison with heuristic guided state space search we again ran TP4, which is also able to consider arbitrary action durations.

The semantics of the plans synthesised by PUP is the same as that of CPT2 and TP4: interfering actions are not allowed to overlap in time (as described in [155]). Also, for the considered domains, the notion of concurrency in the PT-net generated by our translation is equivalent to that allowed in the original planning problem. Thus, PUP|| h^{par} and PUP|| h^0 synthesise plans with the same minimal makespan as those synthesised by CPT2 and TP4.

As shown in the graphs of Figure 6.9, PUP is generally faster than CPT2 and always faster than TP4 in the AIRPORT domain. However CPT2 is able to solve some of the larger problems where the available memory is insufficient for PUP. TP4 is only able to solve half the problem instances.

In PIPESWORLD the results of PUP and CPT2 are mixed, with both solving some problems not solved by the other, and both being sometimes faster. TP4 solves less than half the problems; in these cases the runtimes are comparable with PUP's.

Replanning

We ran REPUP on two benchmarks from the Probabilistic Track of IPC-5, TIREWORLD and ZENOTRAVEL, to see its potential for considering actions with probabilistic effects via

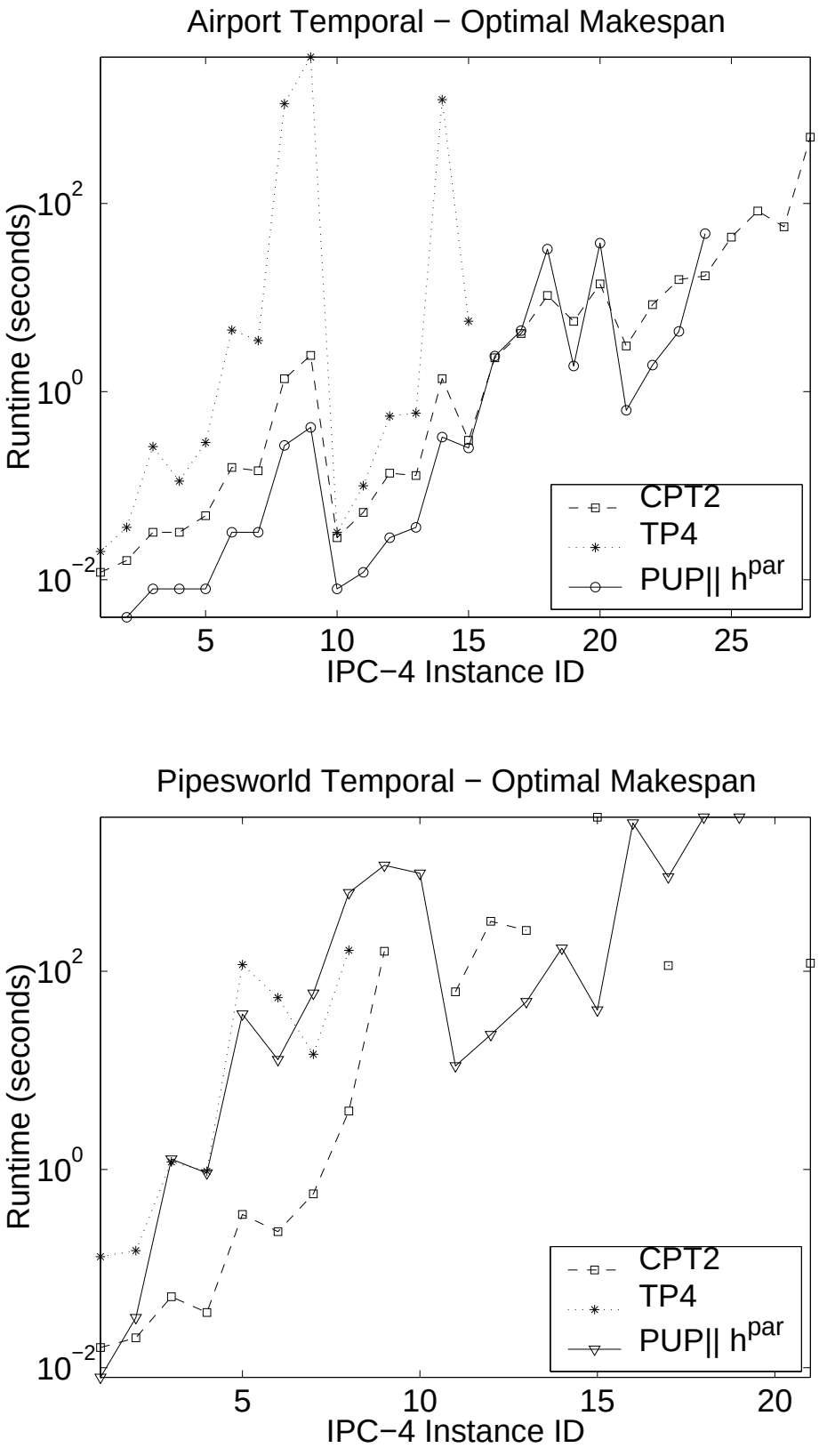


Figure 6.9: Results for AIRPORT and PIPESWORLD - Optimal Temporal Planning

replanning. We ran each problem 30 times, on a Satellite M70 with a 1GB memory limit, and 1 hour time limit (for all 30 runs).

Figure 6.10 shows the percentage of problems solved for TIREWORLD and ZENOTRAVEL. We ran REPUP with $h = h^{\max}, h^{\text{sum}}$ and h^{FF}, but for clarity have only shown the best result here. Also, in these graphs, we have included the published results[8] for FF-REPLAN [173] and FPG [27]. As mentioned previously, for each probabilistic action, FF-REPLAN creates one deterministic action per possible outcome and uses a version of FF to synthesise a plan. Conversely FPG is a "real" probabilistic planner that attempts to build a contingency plan with maximum probability of achieving the goal. FPG [27] (Factored Policy Gradient planner) performs a stochastic local search in the policy space using Policy Gradient reinforcement learning [8]. A policy is a state-action assignment, i.e. in state s execute action a. Rather than estimating state-action values, FPG learns the gradient of the long-term value of the initial state, based on parameters summarising the policy (i.e. the plan). Changing the parameters in the direction of the gradient increases the value of the initial state with respect to the plan. FPG ranked first in the Probabilistic Track of IPC-5.

In TIREWORLD the goal is to drive to a particular destination. This involves deciding what route to take, with the implication that a tire may go flat and some (longer) routes pass through locations where a spare tire can be obtained. TIREWORLD does not have any concurrent actions however the problems are sufficiently small for PUP to identify a plan, where possible. However PUP never chooses to pick up a spare tire, because the probability of not getting a flat tire is higher than getting one (0.6 vs 0.4). Consequently, there is a 40% possibility that the car becomes stranded, with no action possible. This is reflected by the fact the average success rate of REPUP over all 15 problems is 65%. FF-REPLAN appears to have a similar problem. In the case when the car gets a flat tire, PUP immediately identifies the goal propositions are not reachable and ends the problem. This is an example of a probabilistic problem where replanning is not appropriate, because achieving a higher success rate requires constructing a contingency plan. This said, FPG performs no better and, as the problem size increases, it may not be possible to use real probabilistic planning and replanning may be the only viable option.

ZENOTRAVEL involves flying people to specified cities. There are multiple planes that can be boarded, disembarked and flown concurrently. This suggests there may be benefit in planning via unfolding. However the level of concurrency gradually decreases across the problem instances because the number of locations, and the number of people to board and disembark, increases more rapidly than the number of aircraft. Consequently PUP does

[8] www.ldc.usb.ve/~bonet/ipc5/docs/results-probabilistic.pdf

not scale well, and we see an immediate drop in the success rate from 100%, in problems 1-10, to 0% thereafter, when the number of events in the unfolding causes the accumulative node processing time to meet the time limit. The perfect success rate in problems 1-10 can be accredited firstly to the fact there are no "dead ends" in this planning world (unlike TIREWORLD), i.e. it is possible to reach the goal from any state, and secondly to the sufficiently small size and/ or sufficiently high level of concurrency. REPUP's success rate for ZENOTRAVEL exceeds that of any of the probabilistic planners entered in IPC-5. However, we contend that this reflects the "probabilistic uninterestingness"[107] of ZENOTRAVEL, rather than the effectiveness of REPUP which is itself outperformed by FF-REPLAN.

None of the probabilistic planning benchmarks have a sufficient level of concurrency to exploit the benefits of unfolding. We conjecture that REPUP is appropriate for planning worlds with a high level of concurrency (as for the deterministic case), and where replanning is a satisfactory alternative to "real" probabilistic planning. With respect to the later, Thiébaux and Little [107] summarise the circumstances in which replanning is appropriate; the main result relevant to us here, is that either there exist no dead ends, or contingency planning can not reduce the probability of reaching a dead end.

Plan Flexibility

In [130] Nguyen and Kambhampati present the case of partial order planning versus Graphplan and state space planning. One of their main arguments in favour of partial order planning algorithms is the execution flexibility of partially ordered plans. Nguyen and Kambhampati define a plan's *flexibility* as the average number of actions in the plan that do not have any precedence relations among them. For each action a in plan π, $flex(a)$ is the number of actions in π that do not have any direct or indirect ordering constraint with a. $Flex(\pi)$ is the average value of $flex$ over all the actions in π. The higher this value, the higher the number of orders in which a plan can be executed. We observe that this measure is biased towards plan length. So, we define the *unbiased flexibility* as

$$\frac{flexibility}{plan\ length - 1}.$$

This value will be between 0 and 1. If a plan has an unbiased flexibility of 0 then it must necessarily be totally ordered. A value of 1 means every action is independent of every other action. If a plan has an unbiased flexibility of 0.5 then, on average, every action is independent of half the other actions in the plan.

So far, we have looked at how quickly PUP can solve a planning problem, and assessed

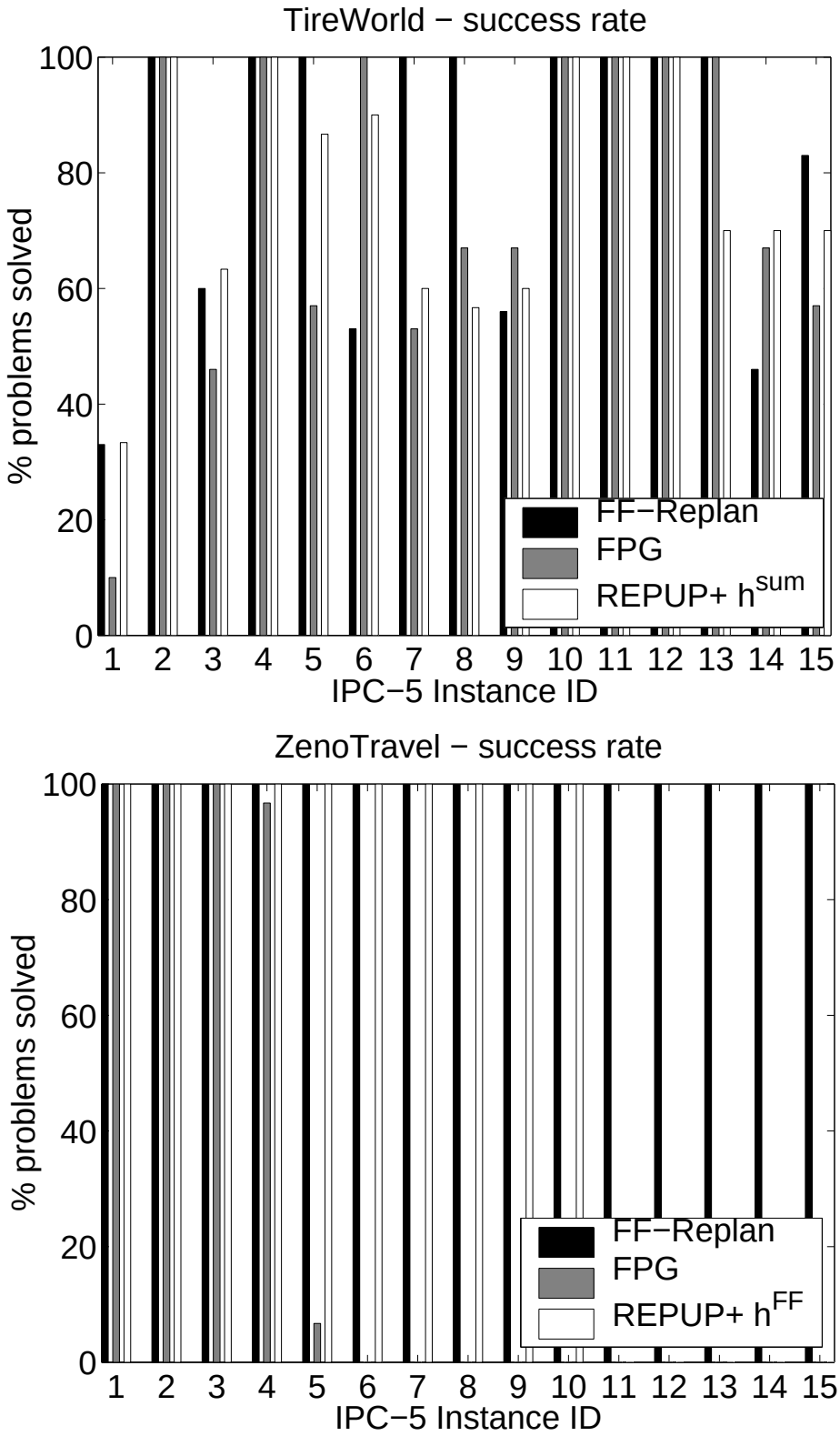


Figure 6.10: Results for BLOCKSWORLD, TIREWORLD and ZENOTRAVEL.

the quality of a solution plan by its length, parallel cost, additive cost and/or makespan. Obviously, each of these qualities depend on the heuristic function used to direct PUP. We now wish consider the unbiased execution flexibility of plans synthesised by PUP. Again, but perhaps less obviously, it is evident that this quality depends on the heuristic function employed. As mentioned previously, we applied PUP to all IPC benchmarks with a STRIPS formulation. We observed that for a given problem, PUP+ h^{sum} often generates a plan with less flexibility than the plan synthesised by PUP+ h^{FF}. This suggests that, for the IPC benchmarks at least, the h^{sum} heuristic tends to direct the unfolding in a depth-first manner, preferring to extend the same local configuration. Alternatively, we conjecture that the h^{FF} heuristic tends to direct the unfolding in a breadth-first manner, preferring to build independent local configurations and join them together.

Nguyen and Kambhampati [130] present empirical results for REPOP and Graphplan on a range of problems based on IPC benchmarks. As mentioned in the previous chapter, REPOP is a variant of UCPOP, developed by Nguyen and Kambhampati. In particular, the length, parallel cost and flexibility of solution plans are given, with aim to display the superior quality of partial order planning solutions. Since PUP also performs partial order planning, we decided to compare the flexibility of plans synthesised via unfolding, versus those built using REPOP and Graphplan. We ran PUP on a subset of the planning problems used in [130] [9]. We avoided those domains which fully disallow concurrency (referred to as serial domains in [130]), since any solution plan will necessarily have a flexibility of 0.

Table 6.2 presents the results of PUP+ h^{FF}, which had the best performance overall amongst planners in the PUP SUITE. The table includes results for REPOP and Graphplan, as published in [130]; note we have unbiased the flexibility value using the number of actions in a plan. These results are mixed and inconclusive. In the rocket domain PUP finds solutions of significantly better quality than REPOP, not only with respect to flexibility but also length and makespan, and similar quality to Graphplan. In the BlocksWorld (bw) domain PUP generates plans which are much less flexible than those of REPOP and Graphplan, but comparable with respect to length and makespan. In the gripper domain all three planners find plans of equal quality, suggesting there may be a single obvious solution to each of these problems. In the logistics domain, PUP and REPOP find plans with four times the flexibility, half the length, and approximately equal makespan, to those found by Graphplan.

With respect to PUP at least, more research is required to understand the relationship between properties of the planning world, the heuristic, and the unbiased execution flexibility of a plan. We can not say, from these results, whether planning via unfolding generates

[9] `http://rakaposhi.eas.asu.edu/repop.html`

Problem	RePop			Graphplan			PUP+ h^{FF}		
	Length	Makespan	Unbiased Flexibility	Length	Makespan	Unbiased Flexibility	Length	Makespan	Unbiased flexibility
gripper-8	21	15	0.03	23	15	0.03	21	15	0.03
gripper-10	27	19	0.03	29	19	0.03	27	19	0.03
gripper-12	33	23	0.02	-	-	-	33	23	0.02
rocket-a	35	16	0.07	40	7	0.18	27	9	0.15
rocket-b	34	15	0.22	30	7	0.17	30	7	0.57
logistics-a	52	13	0.4	80	11	0.08	51	13	0.32
logistics-b	42	13	0.49	79	13	0.07	42	13	0.49
logistics-c	50	15	0.35	-	-	-	51	13	0.49
bw-large-a(9)	8	5	0.39	11	4	0.2	6	6	0
bw-large-b(11)	11	8	0.33	18	5	0.16	10	9	0.04
bw-large-c(15)	17	10	0.32	-	-	-	4	13	0.10

Table 6.2: Plan length, makespan and unbiased execution flexibility, for REPOP, Graphplan and PUP+ h^{FF}.

more or less flexible plans than POCL planning. We conjecture that, in the same way we can not guarantee a short makespan when optimising length, there will probably be no clear answer without orienting each of these techniques to specifically create flexible plans.

6.5 Conclusion

This chapter cast the classical planning problem presented in Chapter 5 to the PT-net REACHABILITY$_\Sigma$ problem presented in Chapter 3. Considering a classical planning problem, we proposed a translation from the planning world to a 1-safe PT-net. This involved mapping the original planning operators to an equivalent set of 1-safe operators with no negative preconditions. The resulting net can then be extended to define a reachability problem that is equivalent to the planning problem. It is then possible to employ the *ERV-Fly* algorithm to solve the reachability problem and thus find a solution plan. Furthermore, we can use the theory of directed unfolding to:

- ◇ Consider the additive or parallel plan cost;
- ◇ Guide the search with an admissible or inadmissible state based heuristic function. If an admissible heuristic is used we can guarantee optimality with respect to the considered cost function.

In this way we can minimise the total additive cost of a plan, which may be its economic cost or simply its length. Alternatively we can minimise the makespan, or simply the number of steps, of a plan. We also outline one way of considering probabilistic actions effects by planning and replanning for the most likely sequence of outcomes. In all cases, a partially ordered plan is generated.

In this chapter we also compared planning via unfolding with the main approaches to classical planning. We recognised that planning via unfolding makes more commitments than traditional plan space planning approaches, but less than state space approaches. It thus offers a different compromise between the possibility of trivially serialisable plans, and the cost of plan refinement. Furthermore, like plan space planning it directly facilitates partial order planning, and can "easily" be extended to consider action duration. Unlike plan space planning, it can also utilise the informative state based heuristics which have significantly impacted the scalability of state space planning approaches. We also made some comparison between the unfolding and a planning graph, where the former performs reachability analysis exactly.

Finally, in this chapter, we presented empirical results for planning via directed unfolding, using benchmark problems from various International Planning Competitions. Additionally, results from a broad selection of distinguished classical planners were provided for comparison. What is clear from the results, is that given a domain with a sufficient level of concurrency, the PUP SUITE is consistently competitive with current state of the art planners with respect to suboptimal and optimal classical planning and optimal temporal planning.

6.5.1 Personal Contribution and Collaboration

The translation from a classical planning world to a PT-net is an original contribution, which resulted from collaboration with Thiébaux and Rintanen. The preliminary translation and systematic breakdown of the challenges of maintaining logical consistency and modelling negative preconditions and effects is my own work, and guided discussions which led to the final translation. In the case any proof presented in this chapter is not my own, it has been stated as such. Ideas for improvements to the translation come from discussions with Haslum. The comparative analysis between planning via unfolding, state space planning and plan space planning is my own work. Extensions to MOLE were made by Bonet, Haslum, Thiébaux and myself. At the ICAPS-06 Doctoral Consortium David Smith suggested applying directed unfolding to replanning, following poor experimental results when extended to optimal probabilistic planning (this later work is beyond the scope of this thesis). The design of REPUP is the result of discussions between Thiébaux and myself. The implementation of REPUP is my own work, but would not have been possible without debugging assistance from Owen Thomas and Olivier Buffet. A more detailed synopsis of the collaborative development of the theory of directed unfolding, and its application to automated planning, can be found at the conclusion of Chapter 4.

This page left blank.

Chapter 7

Conclusions

By focusing on the relationship between classical planning with concurrent actions, and the reachability problem for concurrent restricted DES, this thesis has exploited the connection between AI planning and Petri net analysis to the advantage of both fields. Firstly, inspired by the success of heuristic search in AI planning, we showed how problem specific information can be employed to guide the unfolding process, resulting in a Petri net analysis tool oriented specifically to solving the formal problem of reachability on-the-fly. Secondly, based on unfolding, this thesis presented a new forward search method for partial order planning which can optimise additive or parallel plan cost and utilise state based heuristic functions for guidance. In this final chapter, we recall the two primary contributions of this thesis separately, and consider the limitations of our work to date and possible directions for future work.

In Part I we presented the theory of directed unfolding. This involves controlling the unfolding process with informative strategies, for the purpose of optimality and increased efficiency. We observed that unfolding can be considered a search process when used for on-the-fly reachability analysis, and that its traditional implementation employs a breadth-first search strategy. We identified conditions that ensure the reachability problem is solved optimally with respect to a particular cost function, and crafted strategies to optimise additive and parallel cost functions. Then, motivated by heuristic state space search in AI planning, we showed that the cost function can incorporate a heuristic function to increase efficiency. Furthermore, optimality can be traded for efficiency by using an inadmissible heuristic function, rather than an admissible one. Finally, we demonstrated that a range of heuristic functions can be adapted from AI planning and extracted directly from the Petri net. As part of this work we also identified that the accepted requirements for a correct unfolding strategy are stronger than necessary, opening the door to a new family of strategies

for directing the unfolding.

In the context of PT-net unfolding, the theory of directed unfolding is a useful contribution for three distinct reasons. Firstly, it broadens the function and thus application of unfolding, due to its guarantee of optimality with respect to various cost functions. Secondly, experimental results indicate a significant increase in performance, with respect to reachability analysis, compared to the original "blind" unfolding approach. Finally, directed unfolding enhances our understanding of the nature of unfolding, not only by challenging the requirements on correct strategies (e.g. semi-adequate orders), but by providing a new perspective on unfolding. Our presentation of unfolding as search, drawing parallels and identifying critical differences with state space search, makes directed unfolding accessible to a wider range of researchers who are familiar with the notion of search in general, and heuristic state space search in particular.

There are, however, obvious limitations of the work to date. Whilst empirical results illustrate that directed unfolding provides a significant performance improvement over the original breadth-first implementation featured in MOLE, the actual difference depends on the particular problem and the heuristic function employed. For example, experimental results for the Petri net benchmark DARTES (see Chapter 4) suggest that the overhead in computing heuristic functions can outweigh their benefit for smaller problems. This is a problem inherent in *forward* search, see [19]. Possible solutions include switching to heuristics which only need to be computed once, such as pattern databases heuristics [48]. Alternatively, we could investigate whether an analogue of regression search would make sense in the unfolding space. In addition, looking at the translated classical planning benchmarks considered in the same chapter, we see that directing the unfolding with h^{FF} is more efficient than with h^{sum}, in AIRPORT, and vice versa for OPENSTACKS. Further investigation is needed to identify properties particular to a problem and search strategy (including the heuristic function employed), which are linked to the efficiency of on-the-fly reachability analysis via directed unfolding. Along the same lines, it would be useful to identify theoretical bounds on the size of the unfolding generated in the case when the reachability problem is negative. This will probably require consideration of what fraction of local configurations may have the same final marking but be incomparable, as this leads to multiple non cut-off events corresponding to the same state, and the pruning power of a heuristic function. Previous complexity results [59] are dependent on the order on configurations being total (which restricts the number of non cut-off events to the number of reachable markings), and only repeated parts of the search space being pruned (i.e. a complete prefix must be generated to conclude a negative solution).

Another drawback of this work is that it is so far restricted to PT-nets: most applications utilising Petri net models use more expressive nets such as coloured Petri nets. Future research may extend Khomenko and Koutny's work on defining branching processes for high order nets [96], and consider how the notion of directed unfolding could be applied accordingly. Well developed tools such as PUNF[1] could be adapted for experiments in this area.

In Part II of this thesis we showed how directed unfolding can be applied to classical planning. We are not aware of any work that explores the potential of Petri net analysis techniques for planning, to the depth achieved here. We translated a classical planning world to a PT-net, and subsequently cast the classical planning problem as a PT-net reachability problem. It is then possible to apply directed unfolding for plan synthesis, which we referred to as PUP (Petri net Unfolding Planning). We considered how PUP may fit within the larger picture of automated planning algorithms, and observed that this new approach lies between current state space and plan space planning methods, with respect to the level of commitment it makes during plan refinement. In fact, PUP performs a forward search through the space of partially ordered plans, but unlike traditional partial order planing methods, it makes enough commitment to reason about the state of the world and thus employ state based heuristic functions to make more informed search choices. We also presented the PUP SUITE, a collection of planners that perform optimal and suboptimal classical planning with respect to the additive cost of actions, optimal temporal planning with respect to the plan execution time, and address probabilistic planning problems via replanning.

Planning via directed unfolding contributes to both the practical and theoretical aspects of AI planning. As discussed, it provides a compromise between the ideas of state space and plan space planning. The practical benefit of this is clear from experimental results, which reveal PUP is competitive with current state of the art planning systems on a range of criteria, for problems with a high level of concurrency. Planning via directed unfolding also has theoretical benefit as it provides another perspective on the notion of planning, by combining the state based and action based perspectives of a situation. Also, its graphical representation of both the planning problem, and the process of plan synthesis, is arguably more pragmatic and insightful than any other planning approach as is reveals the correlations between variables in the sensed world and actions in the world of process.

Unfortunately however this thesis lacks conclusive results regarding the complexity of PUP and its applicability to particular planning problems. As discussed in Chapter 3 the worst

[1] `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/tools.html`

case complexity of unfolding is worse than the theoretical complexity of the reachability problem for PT-nets, i.e. REACHABILITY$_\Sigma$. At the same time, we have shown that PUP can be exponentially more efficient than state space planning. To really utilise this new approach to planning we need to better understand what factors contribute to the complexity of PUP. As mentioned earlier, it is desirable to identify bounds on the size of a complete and finite prefix, which are dependent on particular properties of the planning problem and the search strategy employed. Such an investigation may also consider the connections between unfolding and structural and complexity analysis for planning. This includes for example determining whether a precise relationship holds between the size of the unfolding and the width of the causal graph. This will assist in the identification of appropriate domains for PUP. Along the same lines, we need to better understand the trade offs in different approaches to planning. A study to identify and analyse problems which are tractable for traditional partial order planning but intractable for PUP, would be useful for recognising when and how the particular level of commitment made by PUP is beneficial. This should consider the work of Barrett and Weld [7], which identifies a range of problems which vary in tractability with respect to total and partial order planning algorithms.

Another issue of concern is the limited expressivity of the current representation. For example, in the current model, an action's effects are deemed true upon completion of the action; if trying to minimise the execution time of a plan then ideally we want to consider the possibility of an effect becoming true as soon as an action begins. The difference in concurrency semantics is also a concern, but not a major one since we believe this can be remedied by including additional places in the PT-net, as explained in [166]. Also, limiting our expression to propositional logic severely restricts the way in which we can perceive a situation. As discussed in Chapter 6, the translation of planning operators into Petri nets is another area where improvements are likely. Indeed, the problems of limited expressivity and translation size could be addressed concurrently with more ambitious developments. For instance, future research should consider the translation of a planning problem into a higher level Petri net, making use of first-order and multi-valued variables for example. This coincides with our earlier comment that future research should consider how the notion of directed unfolding could be applied to higher level nets. Even if the level of expression remains the same, the current bottleneck in memory usage suggests a more complex reasoning process is a reasonable trade for a more compact representation. Another possibility, since the Petri net and planning world are closely related, is to look at unfolding the PDDL description of a planning problem directly, without translating it to a Petri net representation first. Since an increase in expressivity will mean dealing with new semantics, whatever approach is taken the reasoning process involved in unfolding will need to be extended accordingly.

Before concluding this thesis, let us now outline some more questions for future research:

- ◇ Directed unfolding employs state based heuristic functions, however there is more information contained in the unfolding structure than an enumerated state space: what additional information could be utilised by heuristic functions? Do particular heuristic functions lead to more flexible plans?

- ◇ We have observed the connection between Petri net analysis and planning, but applied just one AI planning technique to PT-net analysis. Can ideas from plan space planning, for example heuristic functions for threat selection, be applied to unfolding? Considering Suzuki and Murata's step-by-step refinement of the nodes in a Petri net [158], how could directed unfolding be used for HTN planning? Would it be useful to use the planning graph to reduce the actions considered when generating new possible events? For example, by identifying a correspondence between an event in the unfolding and an action at a particular level in the planning graph, we can conclude that only the actions in the following level of the planning graph can descend from this event in the unfolding.

- ◇ The complete finite prefix will contain all partially ordered plans from the initial state. Is this useful in some way, perhaps combined with reachability methods designed specifically for the complete prefix? Could the complete prefix be used to synthesise a plan that *avoids* undesirable states? Can parts of the unfolding be reused, perhaps for replanning but more interestingly for planning in the loop?

- ◇ The requirements on a correct strategy for directed unfolding, i.e semi-adequacy, and the conditions for optimality presented in this thesis, are proved to be sufficient but may not actually be necessary. Identifying the necessary conditions in both these respects may open the door to even more possibilities for directing the unfolding.

This thesis has laid solid foundations for the theory of directed unfolding and its application to AI planning, illustrated the value and potential of both these contributions, and in the process formed new connections between the fields of Petri net analysis and AI planning. This is just the beginning however: a deeper understanding of the relationship between the complexity of on-the-fly reachability analysis via directed unfolding and the structural properties of a particular problem, including a measure of the level of concurrency, is required to fully utilise directed unfolding. In addition, it is necessary to move beyond place transition Petri nets and classical planning for this work to be of real practical use in our complex world.

This page left blank.

Bibliography

[1] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, and C. Jard. A Petri net approach to fault detection and diagnosis in distributed systems. Technical Report Publication Interne 1117, IRISA, France, 1997.

[2] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state-space exploration. *Formal Methods in System Design*, 18(2):97–116, 2001.

[3] F Bacchus. Subset of PDDL for the AIPS 2000 planning competition, 2000. http://www.cs.toronto.edu/ aips2000/pddl-subset.ps.

[4] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. *New directions in AI planning*, pages 141–153, 1996.

[5] Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100, 1994.

[6] Christer Backstrom and Bernhard Nebel. Complexity results for SAS+ planning. Technical Report LiTH-IDA-R-93-34, Linkoping, Sweden, 1993.

[7] Anthony Barrett and Daniel S. Weld. Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[8] J. Baxter, P. Bartlett, and L. Weaver. Experiments with infinite-horizon policy-gradient estimation. *Journal of AI Research*, 15, 2001.

[9] Ilan Beer, Shoham Ben-David, and Avner Landver. On-the-fly model checking of RCTL formulas. In *Computer Aided Verification*, pages 184–194, 1998.

[10] A. Benveniste, E. Fabre, and S. Haar. Markov Nets: probabilistic models for distributed and concurrent systems. *Rapport de Recherche INRIA RR-4253*, September 2001.

[11] A. Benveniste, E. Fabre, C. Jard, and S. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[12] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. *Lecture Notes in Computer Science*, 1536:81–102, 1998.

[13] G. Berthelot. Checking properties of nets using transformations. In *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, Berlin, 1985.

[14] E. Best and B. Grahlmann. PEP tool documentation and user guide. Technical report, 1995.

[15] A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science (ENTCS'02)*, 6(2), 2002.

[16] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of Design Automation Conference (DAC'99)*, 1999.

[17] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *International Joint Conference on Aritificial Intelligence IJCAI*, pages 1636–1642, 1995.

[18] Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In *Proceedings of the Ninth International Conference on Automated Planning and Scheduling (ICAPS/ECP-99)*, pages 360–372, 1999.

[19] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[20] Blai Bonet, Patrik Haslum, Sarah Hickmott, and Sylvie Thiébaux. Directed unfolding of Petri nets. Presented at the Workshop on Unfolding and Partial-Order Techniques (UFO-07), 2007.

[21] R. Boubour, C. Jard, E. Fabre, A. Aghasaryan, and A. Benveniste. A Petri net approach to fault detection and diagnosis in distributed systems. Part I: application to telecommunication networks, motivations, and modeling. In *Proceedings of the annual IEEE Control and Decision Conference (CDC'97)*, December 1997.

[22] Renee Boubour and Claude Jard. Fault detection in telecommunication networks based on a Petri net representation of alarm propagation. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets (ICATPN), Toulouse*, volume 1248 of *Lecture Notes in Computer Science*, pages 367–386, 1997.

[23] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy constructions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.

[24] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[25] Wilifried Brauer and Wolfgang Reisig. Zur person Carl Adam Petri und den "Petrinetzen" (Carl Adam Petri and Petri nets). *Informatik-Spektrum*, 29(5):369–374, 2006.

[26] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[27] O. Buffet and D. Aberdeen. The factored policy gradient planner. In *Proceedings of the 5th International Planning Competition (IPC-5)*, 2006.

[28] Alan Bundy, Fausto Giunchiglia, Roberto Sebastiani, and Toby Walsh. Computing abstraction hierarchies by numerical simulation. In *AAAI/IAAI*, volume 1, pages 523–529, 1996.

[29] Alan Bundy, Fausto Giunchiglia, and Toby Walsh. Building abstractions, 1990. DAI Research Paper no. 506, University of Edinburgh. Also IRST-Technical Report 9007-02.

[30] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[31] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[32] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. Prodigy: an integrated architecture for planning and learning. *SIGART Bulletin*, 2(4):51–55, 1991.

[33] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer Science and Business Media, Inc., 1999.

[34] D. Chapman. Planning or conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[35] T. Chatain and V. Khomenko. A note on the well-foundedness of adequate orders used for truncating unfoldings. Technical Report 998, Newcastle University, School of Computing Science, Jan 2007.

[36] Y. Chen, W. T. Tsai, and D. Chao. Dependency analysis-a Petri-net-based technique for synthesizing large concurrent systems. *IEEE Transactions on Parallel Distributed Systems*, 4(4):414–426, 1993.

[37] Y. Chen, B.W. Wah, and C. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of AI Research*, 26:323–369, 2006.

[38] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *Proceedings of the Thirteenth Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS-93)*, pages 326–337, 1993. LNCS 761.

[39] E. M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *Software Tools for Technology Transfer*, 4(3):279–287, 1998.

[40] Edmund M. Clarke, Orna Grumbers, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[41] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional Model Checking. In *Proceedings of Fourth Annual Symposium on Logic in Computer Science*, pages 353–361, Washington D.C., 1989. IEEE Computer Society Press.

[42] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22(3), 1996.

[43] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computer Intelligence*, 5(3):142–150, 1990.

[44] M. B. Do and S. Kambhampati. Solving planning graph by compiling it into CSP. In *Proceedings of the European Conference on Planning (AIPS)*, pages 82–91, 2000.

[45] Mark Drummond. Situated control rules. In *Knowledge Representation*, pages 103–113, 1989.

[46] E.A. Emerson, S. Jha, and D. Peled. Combining partial order and symmetry reductions. In E. Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 19–34, Enschede, The Netherlands, 1997. Springer Verlag, LNCS 1217.

[47] S. Edelkamp, S. Leue, and A. Lafuente. Partial-order reduction and trail improvement in directed model checking. *International Journal on Software Tools for Technology Transfer*, 6(4):277–301, 2004.

[48] Stefan Edelkamp. Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning Systems (AIPS)*, pages 274–283, 2002.

[49] Stefan Edelkamp and Shahid Jabbar. Action planning for directed model checking of Petri nets. *Electronic Notes Theoretical Computer Science*, 149(2):3–18, 2006.

[50] Stefan Edelkamp, Alberto Lluch-Lafuente, and Stefan Leue. Directed explicit model checking with hsf-spin. In *Proceedings of the Eighth International SPIN Workshop*, pages 57–79, 2001.

[51] Stefan Edelkamp and Frank Reffel. OBDDs in heuristic search. In *KI - Kunstliche Intelligenz*, pages 81–92, 1998.

[52] F. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design: An International Journal*, 9(1/2):105–131, August 1996.

[53] Joost Engelfriet. Branching processes of Petri nets. *Acta Inf.*, 28(6):575–591, 1991.

[54] Esra Erdem and Elisabeth R. M. Tillier. Genome rearrangement and planning. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 1139–1144. AAAI Press / The MIT Press, 2005.

[55] G. Ernst, A. Newell, and H. Simon. GPS: A case study in generality and problem solving. *ACM Monograph Series*, 1969.

[56] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-06)*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer-Verlag, 1996.

[57] Javier Esparza. Model checking using net unfoldings. *Science of Compututer Programming*, 23(2-3):151–195, 1994.

[58] Javier Esparza, Pradeep Kanade, and Stefan Schwoon. A negative result on depth first unfolding. *Software Tools for Technology Transfer*, To appear.

[59] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[60] Javier Esparza and Claus Schröter. Unfolding based algorithms for the reachability problem. *Fundamentia Informatica*, 46:1–17, 2001.

[61] Patrick Fabiani and Yannick Meiller. Planning with tokens: an approach between satisfaction and optimisation. In *PuK*, 2000.

[62] E. Fabre, A. Aghasaryan, A. Benveniste, R. Boubour, and C. Jard. Petri net approach to fault detection and diagnosis in distributed systems. Part II: extending Viterbi algorithm and HMM techniques to Petri nets. In *Proceedings of the Annual IEEE Control and Decision Conference (CDC'97), San-Diego*, December 1997.

[63] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[64] Maria Fox and Derek Long. A note on Chapman's modal truth criterion. In *EPIA '93: Proceedings of the 6th Portuguese Conference on Artificial Intelligence*, pages 307–310, London, UK, 1993. Springer-Verlag.

[65] B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of graphplan. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97*, New York, 1997. Springer-Verlag.

[66] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. LPG-Td: a fully automated planner for PDDL2.2 domains. In *Proceedings of the 3rd International Planning Competition (IPC-3)*, 2004.

[67] M. Ghallab, D. Nau, and P.Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[68] Fausto Giunchiglia. Using abstrips abstractions – where do we stand? *Artificial Intelligence Review*, 13(3):201–213, 1999.

[69] Patrice Godefroid and Froduald Kabanza. An efficient reactive planner for synthesizing reactive plans. In *AAAI*, pages 640–645, 1991.

[70] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *CAV '91: Proceedings of the 3rd International Workshop on Computer Aided Verification*, pages 332–342, London, UK, 1992. Springer-Verlag.

[71] Ursula Goltz and Wolfgang Reisig. The non-sequential behavior of Petri nets. *Information and Control*, 57(2/3):125–147, 1983.

[72] C. Green. Application of theorem-proving to problem solving. In D. E. Walker and L. M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, page 219239. William Kaufmann, 1969.

[73] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Transactions of Programning Languages and Systems*, 16(3):843–871, 1994.

[74] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[75] Stefan Haar. Probabilistic cluster unfoldings. *Fundamenta Informaticae*, 53(3,4):281–314, 2002.

[76] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Artificial Intelligence Planning Systems (AIPS)*, pages 140–149, 2000.

[77] Patrik Haslum and Hector Geffner. Heuristic planning with time and resources. In *Proceedings of 6th European Conf. on Planning*, pages 121–132, 2001.

[78] K. Heljanko. Deadlock and reachability checking with finite complete prefixes. Research Report A56, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland. Licentiate's Thesis. December 1999.

[79] Keijo Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. *Fundamenta Informaticae*, 37:247–268, 1999.

[80] Malte Helmert. New complexity results for classical planning benchmarks. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 52–61, 2006.

[81] James Hendler, Austin Tate, and Mark Drummond. AI planning: systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[82] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via Petri net unfolding. Presented at the Workshop on Model Checking and Artificial Intelligence (MoChArt) in European Conf. on Artificial Intelligence (ECAI-06), 2006.

[83] Sarah Hickmott. Concurrent planning using Petri net unfoldings. In *Proc. of 16th International Conference on Automated Planning and Scheduling (ICAPS-06) Doctoral Consortium*, pages 54–57, 2006.

[84] Sarah Hickmott, Jussi Rintanen, Sylvie Thiébaux, and Langford White. Planning via Petri net unfolding. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1904–1911, 2007.

[85] Jöerg Hoffmann, Stefan Edelkamp, Roman Englert, Frederico Liporace, Sylvie Thiébaux, and Sebastian Trüg. Towards realistic benchmarks for planning: the domains used in the classical part of IPC-4. In *Proceedings of the 4th International Planning Competition (IPC-4)*, 2004.

[86] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[87] Bernhard Josko. Verifying the correctness of AADL modules using model checking. In *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop*, pages 386–400, London, UK, 1990. Springer-Verlag.

[88] S. Kambhampati and B. Srivastava. Unifying classical planning approaches. Technical report, Arizona State University, Temple, AZ, 1996.

[89] Subbarao Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 329–340. Morgan Kaufmann, San Francisco, California, 1994.

[90] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18:67–97, 1997.

[91] H. Kautz, B. Selman, and J. Hoffmann. SATPLAN: Planning as satisfiability. In *Proceedings of the 5th International Planning Competition*, 2006. Available at `http://zeus.ing.unibs.it/ipc-5/`.

[92] Henry A. Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[93] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *AAAI*, pages 1194–1201, 1996.

[94] Henry A. Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS-98 Workshop on Planning as Combinatorial Search*, pages 58–60, 1998.

[95] Pauline N. Kawamoto, Yasushi Fuwa, and Yatsuka Nakamura. Basic concepts for Petri nets with boolean markings. *Formalized Mathematics*, 4:87–90, 1993.

[96] Victor Khomenko and Maciej Koutny. Branching processes of high-level Petri nets. In *Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-03)*, pages 458–472, 2003.

[97] Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical prefixes of Petri net unfoldings. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 582–595, London, UK, 2002. Springer-Verlag.

[98] H. C. M. Kleijn and Maciej Koutny. Causality semantics of Petri nets with weighted inhibitor arcs. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR '02)*, pages 531–546, London, UK, 2002. Springer-Verlag.

[99] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[100] I. Koh and F. Dicesare. Modular transformation methods for generalized Petri nets and their application to automated manufacturing systems. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 21, 1991.

[101] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün. Verifying hardware in its software context. In *ICCAD '97: Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design*, pages 742–749, Washington, DC, USA, 1997. IEEE Computer Society.

[102] Nicholas Kushmerik, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):238–286, 1995.

[103] A.L. Lansky. A representation of parallel activity based on events, structure, and causality. In *Reasonning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, 1987.

[104] H. Lee-Kwang, J. Favrel, and P. Baptiste. Generalized Petri net reduction method. *IEEE Transactions on Systems, Man and Cybernetics*, 17(2):297–303, 1987.

[105] Melissa Liew, Lang White, and Sarah Hickmott. Mapping a concurrent temporal plan to a 1-safe timed Petri net. Distribution restricted to DPOLP members, October 2006.

[106] A. Linhares and H.H. Yanasse. Connection between cutting-pattern sequencing, VLSI design and flexible machines. *Computers & Operations Research*, 29:1759 – 1772, 2002.

[107] Iain Little and Sylvie Thiébaux. Probabilistic planning vs replanning. Presented at the Workshop on the International Planning Competition: Past, Present and Future, at the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007).

[108] Iain Little and Sylvie Thiébaux. Concurrent probabilistic planning in the graphplan framework. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 263–273, 2006.

[109] Alberto Lluch-Lafuente, Leue Edelkamp, and Stefan Leue. *Partial Order Reduction in Directed Model Checking*. 2001.

[110] D.E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, 1993.

[111] A.A. Desrochers M.C. Zhou, F. Dicesare. A top-down modular approach to synthesis of Petri net models for manufacturing systems. In *Proceedings of the IEEE Robotics and Automation Conference (Scottsdale, Ariz., May)*, pages 534–539. IEEE, 1989.

[112] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 634–639, Anaheim, California, USA, 1991. AAAI Press/MIT Press.

[113] J. McCarthy, M.L. Minsky, N. Rochester, and C.E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. dartmouth college.

[114] D. McDermott. AIPS-98 planning competition results, 1998.

[115] D. McDermott, A. Ghallab, M.and Howe, C. A. Knoblock, A. Ram, D. M., V.and Weld, and D. Wikins. PDDL the planning domain denition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control., 1999. ftp://ftp.cs.yale.edu/pub/mcdermott/software/ pddl.tar.gz.

[116] Drew V. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[117] K. L. McMillan. *Symbolic Model Checking*. PhD thesis, 1993.

[118] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6:45–65, 1995.

[119] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV*, pages 164–177, 1992.

[120] Kenneth L. McMillan. Circular compositional reasoning about liveness. In *CHARME '99: Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 342–345, London, UK, 1999. Springer-Verlag.

[121] Stephan Melzer. *Verifikation verteilter Systeme mittels linearer- und Constraint-Programmierung*. PhD thesis, Technische Universität München, 1998.

[122] Glenn Miller. Planning and scheduling for the hubble space telescope. In J. A. Eaton G. W. Henry, editor, *ASP Conference Series*, volume 79, pages 173–183, 1995.

[123] Steven Minton, John L. Bresina, and Mark Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[124] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions of Software Engineering*, 7:417–426, 1981.

[125] M.Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Journal of Theoretical Computer Science*, 13(1):85–108, January 1980.

[126] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[127] Nicola Muscettola, Ben Smith, Charles Fry, Steve Chien, Kanna Rajan, Gregg Rabideau, and David Yan. On-board planning for new millennium deep space one

autonomy. In *Proceedings of the IEEE Aerospace Conference*, volume 1, pages 303–318, 1997.

[128] D. S. Nau, J. Meyer, M. Ball, J. Baras, A. Chowdhury, R. Rajamani E. Lin, and V. Trichur. Integrating ai planning and integer programming for use in integrated product and process design. In *AAAI-2000 Workshop on Integration of AI and OR Techniques for Combinatorial Optimization.* AAAI Press, 2000.

[129] G. L. Nemhauser. *Introduction to Dynamic Programming*. New York: Wiley, 1966.

[130] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *International Joint Conference on Aritificial Intelligence (IJCAI)*, pages 459–466, 2001.

[131] William T. Overman. *Verification of concurrent systems: function and timing*. PhD thesis, 1981.

[132] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[133] Doron Peled. Combining partial order reduction with on-the-fly model-checking. In *CAV '94: Proceedings of the 6th International Conference on Computer Aided Verification*, pages 377–390. Springer-Verlag, 1994.

[134] Doron Peled. Ten years of partial order reduction. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 17–28, London, UK, 1998. Springer-Verlag.

[135] Doron A. Peled, Vaughan R. Pratt, and Gerard J. Holzmann, editors. *POMIV '96: Proceedings of the DIMACS workshop on Partial order methods in verification*. AMS Press, Inc., New York, NY, USA, 1997.

[136] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992.

[137] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.

[138] A. Pnueli. In transition from global to modular temporal reasoning about programs. *Logics and models of concurrent systems*, pages 123–144, 1985.

[139] Lucia Pomello. Some equivalence notions for concurrent systems. an overview. In *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets-selected papers*, pages 381–400, London, UK, 1986. Springer-Verlag.

[140] L. Portinale. Petri net reachability analysis meets model-based diagnosticproblem solving. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, 1995*, volume 3, pages 2712–2717, 1995.

[141] M. J. Rattermann, L. Spector, J. Grafman, H. Levin, and H. Harward. Partial and total-order planning: evidence from normal and prefrontally damaged populations. *Cognitive Science*, 25:941–975, November 2001.

[142] Eberhardt Rechtin. *Systems Architecting: Creating and Building Complex Systems*. Prentice Hall PTR, 1991.

[143] Frank Reffel and Stefan Edelkamp. Error detection with directed symbolic model checking. In *World Congress on Formal Methods*, pages 195–211, 1999.

[144] W. Reisig. *Petri Nets An Introduction*. Springer-Verlag, 1985.

[145] Jussi Rintanen. Translation. Personal Correspondence, February 2006.

[146] W. Ruml, M.B. Do, and M.P.J. Fromherz. Online planning and scheduling for high speed manufacturing. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 2005.

[147] Stuart Russel and Peter Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 2003.

[148] Earl D. Sacerdoti. The nonlinear nature of plans. In *4th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 206–214, 1975.

[149] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, sep 1995.

[150] Karsten Schmidt. How to calculate symmetries of Petri nets. *Acta Informatica*, 36(7):545–590, 2000.

[151] John R. Searle. Minds, brains, and programs. *Mind design*, pages 282–307, 1985.

[152] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[153] Fabiano Silva, Marcos A. Castilho, and Luis Allan Künzle. Petriplan: A new algorithm for plan generation (preliminary report). In *IBERAMIA-SBIA*, pages 86–95, 2000.

[154] David E. Smith, Jeremy Frank, and Ari K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[155] D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *16th International Joint Conference on Artificial Intelligence IJCAI'99*, pages 326–333, 1999.

[156] Stephen J. J. Smith, Dana S. Nau, and Thomas A. Throop. Computer bridge - a big win for AI planning. *AI Magazine*, 19(2):93–106, 1998.

[157] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Systems Analysis - Modeling - Simulation*, 8(4-5):293–303, 1991.

[158] I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer Systems Science*, 27:51–76, 1986.

[159] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence IJCAI*, pages 888–900, Cambridge, MA, 1977.

[160] F. Brioschi U. Bertelè. *Nonserial Dynamic Programming*. New York: Academic, 1972.

[161] Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.

[162] Antti Valmari. On-the-fly verification with stubborn sets. In *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, pages 397–408, London, UK, 1993. Springer-Verlag.

[163] P. van Beek and X. Chen. Cplan: A constraint programming approach to planning. In *AAAI/ IAAI Proceedings*, 1999.

[164] Allen VanGelder and Yumi K. Tsuji. Satisfiability testing with more reasoning and less guessing. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1995.

[165] Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335, 2006.

[166] Walter Vogler, Alexei L. Semenov, and Alexandre Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of the ninth International Conference on Concurrency Theory (CONCUR-98)*, pages 501–516, 1998.

[167] Richard Waldinger. Achieving several goals simultaneously. *Machine Intelligence*, 8:94–136, 1977.

[168] D. H. D. Warren. Warplan: A system for generating plans. Technical report, University of Edinburgh. Edinburgh, United Kingdom, 1974.

[169] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[170] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[171] Patrick Henry Winston. *Artificial Intelligence*. Addison Wesley, 1992.

[172] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 233–246, London, UK, 1993. Springer-Verlag.

[173] S. Yoon, A. Fern, and B. Givan. FF-Replan: a baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 52–61, 2007.

[174] H. L. S. Younes and M. L. Littman. PDDL1.1: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Melon University, 2004.

[175] L. Zhang, L.M. Kristensen, C. Janczura, G. Gallasch, and J. Billington. A coloured Petri net based tool for course of action development and analysis. In *Conferences in Research and Practice in Information Technology*, volume 12. Australian Computer Society, 2002.