



Durham E-Theses

CRIKEY! — It's co-ordination in temporal planning

Halsey, Keith

How to cite:

Halsey, Keith (2004) *CRIKEY! — It's co-ordination in temporal planning*, Durham theses, Durham University. Available at Durham E-Theses Online: <http://etheses.dur.ac.uk/2707/>

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

CRIKEY! — IT'S CO-ORDINATION IN TEMPORAL PLANNING

by

Keith Halsey

Minimising Essential Planner-Scheduler
Communication in Temporal Planning



**A copyright of this thesis rests
with the author. No quotation
from it should be published
without his prior written consent
and information derived from it
should be acknowledged.**

Submitted in conformity with the requirements
for the degree of Doctorate of Philosophy
Department of Computer Science
University of Durham

Copyright © 2004 by Keith Halsey



07 DEC 2005

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Durham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Copyright Notice

The copyright of this thesis rests with the author. No quotation from it should be published in any form, including electronic and on the internet, without their prior written consent and information derived from it should be acknowledged.

Acknowledgements

Firstly, I would like to thank Prof. Maria Fox and Dr. Derek Long, for their supervision, guidance and, above all, encouragement — too many times I have needed it. Also to all members of the Strathclyde Planning Group¹, past and present, for their friendship and for making the past three years so fulfilling.

I also extend my very grateful thanks to the University of Strathclyde and its Department of Computer and Information Sciences who have so generously lent me their facilities and resources in the final year of my PhD. Without their support I would have been sure not to complete this work.

Finally I would like to thank my family for their unerring emotional support, often in the face of complete bewilderment at my ever changing plans. Bexstar: thanks for being patient — put the kettle on; I'll be home soon.

¹Alex, Amanda, Andrew, John, Jonathan, Julie, Luke, Pete, Richard, and Stephen

Abstract

CRIKEY! — It's Co-ordination in Temporal Planning

Keith Halsey

Temporal planning contains aspects of both planning and scheduling. Many temporal planners assume a loose coupling between these two sub-problems in the form of “blackbox” durative actions, where the state of the world is not known during the action's execution. This reduces the size of the search space and so simplifies the temporal planning problem, restricting what can be modelled. In particular, the simplification makes it impossible to model co-ordination, where actions must be executed concurrently to achieve a desired effect.

Co-ordination results from logical and temporal constraints that must both be met, and for this reason, the planner and scheduler *must* communicate in order to find a valid temporal plan. This communication effectively increases the size of the search space, so must be done intelligently and as little as possible to limit this increase.

This thesis contributes a comprehensive analysis of where temporal constraints appear in temporal planning problems. It introduces the notions of minimum and maximum temporal constraints, and with these isolates where the planning and scheduling are coupled together tightly, in the form of co-ordination. It characterises this with the new concepts of envelopes and contents.

A new temporal planner written, called CRIKEY, uses this theory to solve temporal problems involving co-ordination that other planners are unable to solve. However, it does this intelligently, using this theory to minimise the communication between the sub-solvers, and so does not expand the search space unnecessarily. The novel search space that CRIKEY uses does not specify the timings of future events and this allows for the handling of duration inequalities, which again, few other temporal planners are able to solve.

Results presented show CRIKEY to be a competitive planner, whilst not making the same simplifying assumptions that other temporal planners make as to the nature of temporal planning problems.

Contents

1	Introduction	11
1.1	Preliminary Introduction and Overview	11
1.1.1	Classical Planning	11
1.1.2	Scheduling	12
1.1.3	Temporal Planning	13
1.2	Context	13
1.3	Scope, Aims and Motivation	15
1.4	Outline	16
2	Background	18
2.1	Models of Time	18
2.1.1	Views of Change	18
2.1.2	Classifications	20
2.1.3	Temporal Problems	22
2.1.4	Durative Actions	22
2.1.5	Reasoning About Time	26
2.2	Resources in Planning and Scheduling	26
2.3	Decomposition of Problems	27
2.3.1	HybridSTAN and TIM	28
2.3.2	Translation of the Planning Problem	29
2.3.3	Goal Orderings as Decomposition	29
2.3.4	Advantages of Decomposition	30
2.4	Integrating Planning and Scheduling Technologies	30
2.5	Planners	31
2.5.1	Graphplan-based Temporal Planners	31
2.5.2	Forward Heuristic Search	33
2.5.3	Decomposing Planners	35
2.5.4	State of the Art	36
2.6	Chapter Summary	37

3	Theory	38
3.1	An Initial Solution The LPGP/FF Hybrid	38
3.2	Coupling of Planning and Scheduling	47
3.2.1	Failure of the LPGP/FF Hybrid	49
3.3	Temporal Constraints in PDDL2.1	50
3.3.1	Translation of the Domain	54
3.4	Envelopes and Contents	56
3.5	Detecting Single Potential Envelopes	58
3.5.1	Reasons for Precedence	59
3.5.2	Defining Potential Envelopes	59
3.6	Chapter Summary	63
4	CRIKEY	64
4.1	Version 1	65
4.1.1	Envelope Analysis	66
4.1.2	Planning in Version 1	68
4.1.3	Scheduling in Version 1	74
4.2	Characteristics of Version 1	74
4.3	Version 2	75
4.3.1	Envelope Management	76
4.3.2	Scheduling	79
4.4	Comparison with Sapa	85
4.5	Chapter Summary	86
5	Results	87
5.1	Capabilities	88
5.2	IPC'04	90
5.2.1	Analysis Overview of IPC'04 Domains	109
5.3	Co-ordination	109
5.3.1	The Match Domain Revisited	110
5.4	DriverLog Shift	117
5.4.1	Mousetrap	120
5.4.2	Baseball	121
5.5	Using the Metric	121
5.6	Chapter Summary	122
6	Conclusions	124
6.1	Summary	124
6.2	Critique of CRIKEY	125
A	Example LPGP Translation	128

CONTENTS	7
B The Zeno Travel Domain	136
C The Match Domain	139
D Alternative Formalisation	141
E The Café Domain	143
F The Lift Match Domain	146
F.1 Partial Lift Match Numeric Domain	149
G DriverLog Shift Domain	150
H Mousetrap Domain	154
I Baseball Domain	157

List of Figures

1.1	A Generic View of Temporal Planning	15
2.1	Views of Change in Planning	19
2.2	Possible Concurrency Issues with Durative Actions	25
2.3	Different Types of Temporal Planning Graph	33
2.4	Communication in RealPlan	36
3.1	The Proposed Separation of Planning and Scheduling in the Hybrid Planner .	39
3.2	Architecture for Separating Planning and Scheduling	40
3.3	The LPGP Translation of Durative Actions	43
3.4	The Veloso Algorithm to Translate Totally Ordered Plans to Partially Or- dered Plans	44
3.5	Example of a Broken Invariant	46
3.6	Example of an End Action Deleting a Goal	46
3.7	Coupling Between Planning and Scheduling in Temporal Planning Domains .	47
3.8	A Valid Plan for the Match Problem	49
3.9	Expressing a Maximum Minimum Elapsed Time Between Actions in PDDL2.1	51
3.10	Possible Combinations of Representing the Same Constraint	52
3.11	Expressing both Minimum and Maximum Time Between Actions in PDDL2.1	54
3.12	Two Possible Equivalent Representation of the Breakfast Domain	54
3.13	Comparison of the Match Domain and Minimum and Maximum Delays in PDDL2.1	55
3.14	Envelopes and Contents	58
3.15	The Three Reasons to Order Actions	59
3.16	Potential Envelopes (with achieving contents)	62
3.17	A Hard Envelope modelling a time limited resource	63
4.1	Differences Between the LPGP/FF Hybrid and the Two Versions of CRIKEY	64
4.2	Architecture Overview of CRIKEY	65
4.3	Alternative Architecture Overview of the LPGP/FF Hybrid	66
4.4	Example Precedence Graph	82

4.5	A Partial Order for the Café Domain	84
4.6	Two Plans with Identical Goals but Different Metrics	84
5.1	Two Possible Complex Envelopes	89
5.2	Non-temporal Small PSR Domain	92
5.3	Non-temporal Dinning Philosophers Domain	94
5.4	Non-temporal Optical-Telegraph Domain	95
5.5	Non-temporal No Tankage Pipesworld Domain	97
5.6	Temporal No Tankage Pipesworld Domain	98
5.7	Non-temporal Tankage Pipesworld Domain	99
5.8	Temporal Tankage Pipesworld Domain	100
5.9	Temporal UMTS Domain	102
5.10	Temporal Flawed UMTS Domain	103
5.11	Temporal UMTS Domain with compiled Time Windows	104
5.12	Non-temporal Airport Domain	106
5.13	Temporal Airport Domain	107
5.14	Temporal Airport Domain with Time Windows	108
5.15	Standard Match Domain	111
5.16	Variable Time Match Domain	113
5.17	The Lift Match Domain	115
5.18	Performance of CRIKEY with and without matches encoded using fluents . .	116
5.19	Standard DriverLog domain as used in IPC'02	118
5.20	DriverLog Simple Time Domain Converted to use Shifts	119
5.21	Degradation of Performance when DriverLog Domain Converted to use Shifts	120
5.22	Plan Quality in the Café Domain with CRIKEY version 2	123

List of Tables

2.1	Basic Relations Between Intervals	20
2.2	State of the Art Temporal Planners	37
3.1	Nine Possible Combinations of Start End Pairs from the Three Ordering Reasons from the Veloso Algorithm	60
4.1	Possible Specifications of Durations and Resource Conditions and Operators .	83
5.1	Temporal Planner Concurrency Capabilities	88
5.2	Temporal Planner Temporal Capabilities	90
5.3	Percentage of Time Spent in Temporal Planning by CRIKEY in the Match Domain	112
5.4	Percentage of Time Spent in Temporal Planning by CRIKEY in the Driverlog Domain	121

Chapter 1

Introduction

1.1 Preliminary Introduction and Overview

1.1.1 Classical Planning

Classical Planning is a well defined construction problem where actions are chosen to reach a goal state from an initial state. In its simplest form (STRIPS [23]), states are defined by a set of logical propositions that describe currently true facts about the world. The effects of actions change the state by either adding propositions (“add effects”), or removing them (“delete effects”). Actions also have “conditions” — propositions that must be true in a state in order to apply the action.

The solution to a planning problem is an ordered sequence of actions called a **plan**, to be executed by an executive (or agent) that applies each action in turn from the initial state to reach a goal state.

It is well known that planning is a P-space hard problem [22], but furthermore, it is considered preferable to produce a good quality solution and at best, an optimal plan, that is, one with a minimum number of actions. Recently the notion of optimality has not been researched as much as satisfiability (simply finding *a* plan).

STRIPS has been extended to ADL [58] to increase the expressiveness of the problem definition language, including typing of objects in the domain, negative preconditions, and quantified conditions and effects. However, classical planning still makes a number of assumptions that are described in [70]. These include:

- **Static World** — The only cause of change in the world is from the actions performed by the executive.
- **Deterministic** — The effects of actions are completely known.
- **Fully Observable** — The state of the world is completely known.

- **Finite** — The world is finite in every aspect and objects cannot be created.
- **Atomic Time** — Time is composed of indivisible units with each action taking a single time unit.

These assumptions have all been weakened and the consequences explored. This thesis is interested in the relaxation the last of these, that of atomic time. As it holds, it has a significant consequence: The state of the world need not be considered whilst the execution of an action is in progress. Instead, the execution is an atomic transformation from one state to another. Importantly, on account of this, concurrent execution of actions is impossible. The addition of time into classical planning is called “Temporal Planning” and will be looked at closer in Section 1.1.3.

Another common extension to classical planning is “Metric Planning”, where there are not only propositional variables taking the values of true and false, but also metric fluent variables that can take numeric values. This allows the easy modelling of resources, but makes the problem more complex, as the state space is potentially infinite.

1.1.2 Scheduling

Within the research community there is less agreement, when compared to the planning problem, as to *exactly* what the scheduling problem is. However, there is agreement as to what the class of scheduling problems entail. Whereas planning is a construction problem, deciding which actions should be used to reach a goal without breaking any logical constraints, scheduling is often an optimisation problem deciding when actions (often called tasks or activities) should occur without breaking any temporal or resource constraints. Alternatively, scheduling could be defined as allocating resources to activities over time.

Classes of scheduling problem include job shop scheduling (allocating tasks to machines in a factory), multiprocessor scheduling and timetabling. Sometimes the tasks are pre-emptive and can be interrupted, other times not so. Scheduling problems can also be recursive, where the jobs are reoccurring and a repeating schedule must be found. It is common for the jobs to have deadlines.

Planning is commonly characterised as the problem of “what” activities should be performed and scheduling as the problem of “when” and “with what” should they be executed. Generally, in planning there may be fewer solutions to find, but in scheduling, finding a solution can be relatively easy, making it more important to find a good quality or optimal solution. What constitutes a good solution can change; it may be preferable to maximise the slack, or alternatively to minimise the number of late jobs, the quantity of resource used, the total time for the whole schedule or another, different, criteria.

1.1.3 Temporal Planning

Temporal Planning is classical planning with the assumption of atomic time removed. Metric time (where time takes a value, rather than simply being relative) is explicitly modelled in the planning problem. It is incorrect to assert that classical planning has no time as it is the building of a trajectory (a future course of actions), with a predicted outcome, and so is developed inherently with respect to time. However, in classical domains there is a very restrictive set of assumptions on the nature of time.

Time is an important element in many “real world” problems and adding a significantly less restrictive time model makes the problem more expressive. For example, it is impossible to form a good model of concurrency in the classical planning framework. It is also not suitable when modelling actions that preserve a value over time, goals that are situated in time or dynamic domains with predictable exogenous events outside the control of the executive. And of course, actions rarely all have the same duration in reality.

The modelling of time is examined closer in Section 2.1. It has an impact on the complexity of the planning problem (making it harder still) and also on what the planning problem is: temporal constraints must be met, as well as the logical constraints. Importantly, a solution is no longer simply an ordered sequence of actions, since these can now be executed concurrently, but a time-stamped plan, where actions are given metric time values for their proposed execution.

Temporal planning is the combination of classical planning and scheduling, since now the problem combines the “what”, “when” and “with what” elements (i.e. it must both plan the actions to use and schedule them in time against the resources). This is a natural combining of problems as they have similar building blocks (i.e. actions / activities).

All these problems lie on a spectrum. At one extreme are the pure planning problems, concerned only with logical reasoning, and at the other extreme sit the pure scheduling problems, with no choice of actions, just their position in time. Temporal planning problems lie somewhere on this spectrum in-between the two extremes.

1.2 Context

This thesis focuses on where problems lie on this spectrum — how much planning and how much scheduling is present in the problem and how they interact. The constraints between the two problems affects how coupled they are. Problems that are independent have no constraints between them. A weak set of constraints will result in one problem only affecting the quality of the other, and a strong set of constraints results in the solution to one problem affecting whether the other is satisfiable (that is, possible to solve).

Described here are three examples of temporal planning problems to put this work in context.

Building a House

When people decide they want to build a house, they must decide both what to do and when to do it. This is a temporal planning problem. It is logical to first decide on a plan of action which will contain tasks such as dig the foundations, build the walls, and put in the windows. Only once these activities have been planned would the human decide when to do the various activities. There will be some precedence constraints between some of these actions (for example, to build the walls before the roof is put on). However, there will also be some choices, for example the electrician could either come before or after the plumber, but not at the same time (as they would get in each others way). Having the electrician come first could mean that the plasterers can complete their job quicker and so the house is finished sooner. Other tasks could happen concurrently, for example the upstairs could be painted whilst the carpets are laid downstairs.

In this example the planning stage and the scheduling stage do not impact on each other. In this case it is logical to do the two phases completely separately, since only decisions made whilst scheduling affect how quickly the building is completed.

Evacuating an Island

Suppose there has been a volcanic eruption on an island, and it is necessary to evacuate it. This again is a temporal planning problem as there is an initial state (volcanic island with inhabitants), a goal state (all people evacuated) and actions to choose from (building a landing strip, evacuate by plane, evacuate by boat etc. . .). In this case the choice of action will affect the quality of the schedule. It may be quicker to build two landing strips and then operate twice as many planes. The choice of actions may also affect the satisfiability of the schedule. There may not be enough fuel to evacuate everyone by plane, so some must go by boat.

In this problem the planning and scheduling are more tightly coupled than in the previous example as the choice of action affects both the quality and the satisfiability of the schedule.

Air to Air Refuelling

In planning to perform air to air refuelling with aeroplanes, the planes must be co-ordinated to achieve the goal. They must both be at their respective locations at the same time and they must both be planned and scheduled simultaneously to achieve this. Some action will have to happen concurrently. Here the scheduling task cannot be separated from the planning task as in the first example.

1.3 Scope, Aims and Motivation

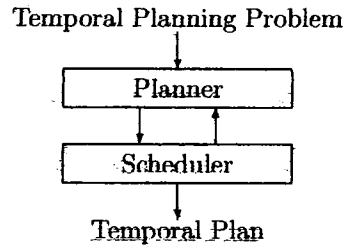


Figure 1.1: A Generic View of Temporal Planning

Figure 1.1 illustrates a general temporal planning architecture where the problem is decomposed into planning and scheduling. Partial solutions, satisfiability, constraints and cost estimates can all be passed between the two solvers. For example, the planner chooses some actions and passes them to the scheduler. The scheduler can then pass back the feasibility of finding such a schedule, a cost estimate of what a potential schedule might be in terms of time and resource consumption, or it could pass back a full or partial schedule to the planner. The planner can then use this information in its search.

Pivotal to the nature of the temporal planner is the amount of communication passing between the two solvers: both how often and how much information they communicate. This leads to a trade-off: the more they interact, potentially the higher the quality of the final solution. However, the communication is expensive and so it is preferable to minimise it, thus reducing execution time and resources.

In cases where there is no interaction between the planning and the scheduling, then no communication need take place. Where the choice of action affects the quality of the schedule, then the trade-off stands. Finally, where the planning and scheduling are tightly coupled and the choice of action affects whether a schedule can be found for the problem, the solvers *must* communicate in order to find a valid final solution.

Splitting the problem is not the only way to solve temporal planning, however to do so the interactions between the sub-problems must be understood. Once this is understood, the communication between the solvers can be controlled and the trade-off met with some intelligence.

The scope of this work is to examine and understand where, and to what extent, planning and scheduling interact in temporal planning domains. The focus is on where the problems are very tightly coupled, as in the case of co-ordination, as there are currently few temporal planners capable of solving such problems and it is largely ignored in the benchmark domains. The motivation is to understand the communication needed between the planner and the scheduler, with the aim to write a temporal planner to understand these interactions and

perform accordingly, communicating only where necessary. Whilst the nature of the planner and scheduler will, of course, greatly affect the performance of the overall system, this thesis is only concerned with the interaction between them.

Here is a summary of the objectives of the thesis as written above:

Scope of Theory To examine and understand where planning and scheduling interact in temporal planning.

Focus The thesis will focus on where the problems are tightly coupled.

Motivation To intelligently solve problems currently ignored by the community.

Aim To build a competitive planner to solve these problems using the understanding of logical and temporal constraint interactions.

The contribution to the community through this work is:

- Understanding of temporal constraints in temporal planning (specifically in PDDL2.1 and similar languages).
- A planner that uses this theory to minimise communication between the planner and scheduler and so solve problems that are not solved by other planners.
- A novel search state that does not specify the future timings allowing for duration inequalities.

1.4 Outline

The next chapter reviews different models of time and resources and how this affects the temporal planning problem (Sections 2.1 & 2.2). It looks at the decomposition of problems, and an abstract look at communication between sub-solvers in Section 2.3. Finally, it takes a closer look at temporal planning, the general principles behind integrating Planning and Scheduling (Section 2.4) and a survey of other research in the field that is tackling temporal problems (Section 2.5).

Chapter 3 develops a theory of co-ordination in temporal planning by looking at where planning and scheduling problems are tightly coupled. It examines where temporal constraints appear in temporal planning problems using durative actions. It introduces new concepts of envelopes and contents, and of minimum and maximum precedence relationships to classify the temporal constraints. The chapter starts with a planning system where the planning and scheduling are split, but very little communication takes place between the solvers. An example domain containing co-ordination is presented and it is shown how this system fails in this case.

Chapter 4 describes the implementation of a temporal planner called CRIKEY, written to solve the failures of the system set out in Chapter 3 and achieve the aims set out above. It uses the theory developed in the previous chapter to do this intelligently and efficiently. Two versions of the planner are described (Section 4.1 & 4.3), the second of which is built on the first and contains a novel search state. This new state results in complete search for domains where the planning and scheduling are tightly coupled. The planner is compared for similarities and differences with Sapa, a similarly expressive temporal planner (Section 4.4).

Chapter 5 presents some results of the implemented system, using temporal planning problems that lie on a range of positions on the spectrum between planning and scheduling, both where they interact heavily and where they do not. It is compared against systems described in Chapter 2 on both the quality of the plan produced and also of the speed of the planners. These results are analysed and explained. Finally, Chapter 6 summarises the work presented in the thesis and this is followed by a critique of it, including the strengths and weaknesses of the approach taken.

Chapter 2

Background

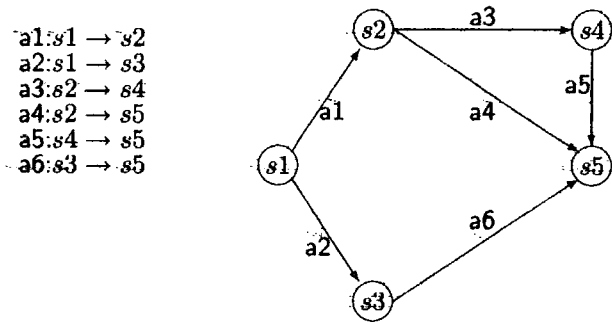
2.1 Models of Time

Modelling the flow of time is intrinsic to temporal planning and scheduling and requires a specific representation that is domain independent to allow general temporal reasoning. Its representation impacts on what is expressible, such as concurrency and also on the complexity of the problem. In many ways, time can be likened to resources but has an important property that differentiates it. Time flows independently and regardless of any actions: it is not produced and consumed like a traditional resource. Also, it orders causality, that is, causes must precede effects.

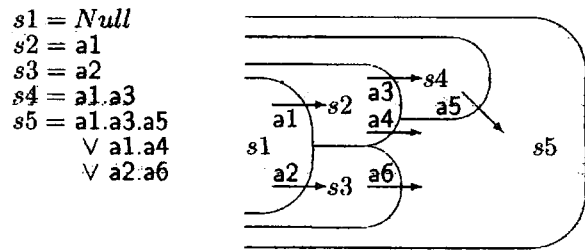
2.1.1 Views of Change

Change is fundamental in planning and is inextricably linked to time as time can only be observed through change. Classical planning can be seen as a state transition diagram, where change happens when transitioning from one state to another. Therefore, since the actions are sequenced, the flow of time in classical planning is represented by the transitions to get from the initial state to a goal state. However, concurrency is impossible (since it means taking two or more transitions at once).

Lansky [51] identifies a duality between actions and states. Actions are seen as state changing functions (taking a state orientated view), but simultaneously, states are seen as records of what actions have taken place (resulting in an action orientated view). The first of these is coherent with the state transition model, the second gives rise to a new description described in [29] as “the histories view of change”. In this, states are seen as evolving continuously, with different evolutions linked by instantaneous moments of change. States not only represent the current state of the world but also what has come before and is to happen in the future. Effects of actions, or exogenous events, can change this evolution and not necessarily at the time of the action. Fox and Long [29] produce two useful representations of these views (Figure 2.1).



(a) The Classical State Transition View of Planning



(b) The Histories View of Change: states are made up of histories of events

Figure 2.1: Views of Change in Planning

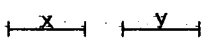

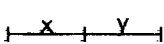
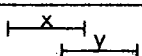
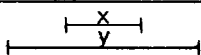
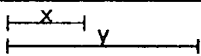
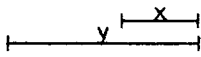
2.1.2 Classifications

Modelling Change in Time

Time is most naturally thought of as passing in intervals whereas (logical) change tends to happen at points in time. Thus time can either be modelled as interval-based or point-based [65].

One such interval-based framework is described in [1] and specifies thirteen basic relations that can hold between two intervals (see Table 2.1).

Table 2.1: Basic Relations Between Intervals

Relation	Predicate	Symbol	Inverse	Meaning
x before y	<i>BEFORE</i> (x, y)	<	>	
x equal y	<i>EQUAL</i> (x, y)	=	=	
x meets y	<i>MEETS</i> (x, y)	<i>m</i>	<i>m'</i>	
x overlaps y	<i>OVERLAPS</i> (x, y)	<i>o</i>	<i>o'</i>	
x during y	<i>DURING</i> (x, y)	<i>d</i>	<i>d'</i>	
x starts y	<i>STARTS</i> (x, y)	<i>s</i>	<i>s'</i>	
x finishes y	<i>FINISHES</i> (x, y)	<i>f</i>	<i>f'</i>	

Disjunctions are allowed between these relationships for greater expressivity (i.e. $\{<\} \cup \{=\} = \{\leq\}$) and so the predicate *IN* is defined as:

$$IN(t_1, t_2) \Leftrightarrow (DURING(t_1, t_2) \vee STARTS(t_1, t_2) \vee FINISHES(t_1, t_2))$$

Furthermore, there are axioms asserting that each relationship is mutually exclusive of the others and axioms to describe the transitivity of them, such as:

$$BEFORE(t_1, t_2) \wedge BEFORE(t_2, t_3) \Rightarrow BEFORE(t_1, t_3)$$

The predicate *HOLDS*(*p*, *t*) defines whether a proposition *p* is true during the interval *t*.

With these relationships it is possible to reason about other, more complex, relationships, and prove that certain facts must be true over certain intervals. “Occurrences” allow descriptions of action and are split into two categories: “processes” that refers to activity not culminating in a result (such as a fan being on) and “events” that produce an outcome (such as a moving across a room). Causality is expressed by *ECAUSE* (natural causes, such as a non supported object falling) and *ACAUSE* (where a deliberate action by the agent causes

an affect). The relationships allow reasoning about time and change in time. In particular this model can represent non-activity (such as waiting at the road side) and maintenance goals (satisfying a goal over a period of time).

Point-based frameworks include [55] and [69]. In the second of these there are only three relations that can hold between two points: $<$, $=$, and $>$. Again, disjunctions are allowed so the complete set of possible relations between two points are $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$.

It is proven in [69] that it is possible to translate between a subset of the interval-based framework and the point-based framework. We refer to the beginning and end of a point based action such that A^- is the start of A and the end A^+ . Using an example from [65], representing that “Fred read his paper, during which he started drinking his tea”:

$$\begin{aligned} & \text{paper}\{o, s, d\} \text{tea} \Rightarrow \\ & (\text{paper}^- < \text{paper}^+) \wedge (\text{tea}^- < \text{tea}^+) \wedge (\text{paper}^+ > \text{tea}^-) \wedge (\text{paper}^+ < \text{tea}^+) \end{aligned}$$

Discrete or Continuous

Metric time is classified as discrete or continuous. Where the time is continuous, time variables can take any real value. This means that time is always divisible, and if two timepoints are not exactly simultaneous, then it is always possible to order them. Conversely, discrete time proceeds in steps and it is impossible to reason about the state in between two time units. Whilst continuous time is more expressive, it leads to an infinitely larger search space.

Concurrency, Co-ordination and Synchronisation

Key to the concept of temporal reasoning is “Concurrency” — what can and what cannot happen simultaneously. For two actions to be able to happen concurrently they must not interfere with one another, for example, they cannot delete each others effects. Two actions that are concurrent have an interval relationship of $\{=, o, o', d, d', s, s', f, f'\}$ (Section 2.1.2).

“Co-ordination” is where the actions can (or even must) happen together (so are concurrent) and interact with one another. The classic example is of lifting a bowl [35]. To succeed without spilling the contents, both sides must be lifted at the same time. These two actions (lifting the left and lifting the right side) interact to keep the bowl level. This is opposed to two actions that are not co-ordinated (but can still happen in parallel) such as one truck being driven from Glasgow to Edinburgh and another being driven from London to Durham. Co-ordinated actions have an interval relationship of $\{o, o', d, d'\}$.

Finally, “Synchronisation” is a form of co-ordination where the precise timings are imperative to the effect of the actions. An example is the hitting of a ball with a bat. It is essential that the throwing of the ball and the swinging of the bat happen at *exactly* the right times to ensure the correct outcome. Synchronised actions have an interval relationship of $\{=, s, s', f, f'\}$.

Typically, benchmark temporal planning problems do not contain any co-ordination or synchronisation.

2.1.3 Temporal Problems

In relation to temporal planning, [29] identifies two classifications of temporal planning problems, “Temporally Extended Actions” (TEA) and “Temporally Extended Goals” (TEG). TEA is, “the classical planning problem extended with the notion of activities taking time to have their expected effects”, and so encompasses cases where actions have a duration. Plan quality can take a new metric: the length of time taken to complete the plan (since TEA allows actions to occur concurrently). Importantly in TEA, the representation of the goal and initial state are no different from classical planning. As will be seen in Section 2.5, these are the temporal problems that have been explored most extensively by researchers in temporal planning. This perhaps is not surprising since TEG is an extension of TEA and so more complex.

In TEG, the goal state is no longer associated with the final state, but with trajectories through the search space. As an example, a goal could require a proposition to be true over a specified time interval or achieved by a specified deadline which could force concurrency to occur in the plan. A further extension is Temporally Extended Initial States that allows predictable exogenous events to be expressed.

2.1.4 Durative Actions

The most common way to model time in temporal planning is to use durative actions (actions with an associated duration) where the effects of an action take time to change the world.

The casicst (and least expressive) method is to simply extend a classical action with the addition of a numeric duration. Preconditions must hold at the beginning and for the duration of the action. Effects are undefined during the action and only become true (or false) at the end. These are generally called “blackbox” actions since there is no knowing what is happening during their execution. Because of this, they only allow a very restrictive concurrency model; only actions that do not interfere in any way can be executed together. This does not allow for co-ordination and does not support actions that make a fact true only during their execution. Blackbox actions are used in TGP [62], TPSys [33] and TP4 [40] (see Section 2.5).

A more expressive form of durative action stipulates conditions to hold at the start or end of the action or for whole duration (these are called invariants). In addition, effects can become true at the start or end of an action, and so are defined during the execution of the action. Take as an example a “fly” action. It should be a precondition that the plane is at its start location. However, as soon as the action starts, it is no longer there, so this should be a start delete effect. For the duration of the flight, it should be an invariant that the engines remain on and it should be an end effect to assert that it is now at its destination.

This allows for a much greater degree of concurrency, namely co-ordination, since the state of the world is known during the execution of an action.

PDDL2.x

This durative action model is used by PDDL2.1 [28] and has been widely adopted, mainly due to its use in the International Planning Competitions (IPC). PDDL [37] in its original version was specified for the first two competitions (IPC'98 [56] and IPC'00 [2]). It covers STRIPS and ADL for classical planning. The problem is described in two parts: firstly the (abstract) domain, describing the operators (abstract actions) and predicates, and secondly the problem instance, specifying the initial and goal states.

For the third competition in 2002 [27], PDDL was extended to PDDL2.1 to include temporal and metric domains. The temporal aspects are introduced through durative actions where conditions and effects are specified to hold either at the start or at the end of the action. Conditions that must hold throughout the execution of the action are specified as invariants.

PDDL2.1 also introduces numeric variables (fluents) that become part of the state along with propositions. They can be used in both effects and conditions. The effects use operators (scale up, scale down, increase, decrease and assign) to change the value of a fluent by some function (+, −, ×, ÷) of fluents and real numbers. Conditions use comparators (\leq , $<$, $=$, $>$, \geq) between functions of fluents and real numbers.

For the purposes of the competition, PDDL2.1 was split into levels [26]. Levels 1 to 4 refer to the agreed specification and level 5 is the completed language with formal semantics (PDDL+) that allows the modelling of processes. The first four levels are as follows, with each level extending the previous level:

Level 1 As the original PDDL, corresponding to the propositional parts including ADL.

Level 2 Numeric variables and the ability to test and update their values instantaneously.

Level 3 Durative actions as described above.

Level 4 Effects happening during the execution of an action (much like the invariants for conditions). So called “continuous effects” can update a numeric variable by some function of time passed since the start of the action.

Level 4 allows an action to continuously update a numeric variable, the value of which is known throughout the execution of the action. For example, in an action representing the filling of a bath, the level of the bath is always known. Co-ordination is also expressible here as it is easy to model action interactions, such as where a jug of water is also poured concurrently into the bath, filling it up quicker. The numeric value representing the level of the water is updated correctly.

The language has been extended further once more to PDDL2.2 [21] for the last competition held in 2004 [44] to include two new features ; “Derived Predicates” and “Timed Initial Literals”(TIL). Derived predicates change the classical planning problem. Actions are still the sole cause of change, but not necessarily explicitly so. It is possible to specify when a proposition becomes true or false in relation to other propositions, based on “if then” rules. For example it is possible to express, “if the washing is on the line, and it is raining, then the washing is wet.”

Timed initial literals allow the specification of exogenous events in the initial state. So for example, it is possible to state that a shop will open at 0900 and close at 1730. These events are outside the control of the planner, although are predictable and known in advance.

PDDL2.2 problems that contain either of these features can be compiled down to PDDL2.1 problems but whereas timed initial literals is a polynomial compilation, derived predicates can potentially lead to an exponential growth in the number of actions needed.

PDDL2.1 has a TEA outlook on the nature of temporal planning problems. However, just as it is possible to model timed initial literals easily in PDDL2.1, so it is also possible to model other TEG aspects, such as deadlines, maintenance goals and temporal constraints. This is discussed in [30]. Whilst they are not cleanly represented, it is still perfectly possible to express these features and the translation is polynomial in the size of the instance. Note however, that it is not possible to translate PDDL2.1 domains (with numerics and time) into the original PDDL domains polynomially in the size of the instance.

One problem encountered by the semantics of durative actions is illustrated in Figure 2.2. In part (a), action A achieves action B. The question is, can B start immediately as A finishes i.e. what is the truth value of p at this point? This is known as the “divided instance problem”. The solution adopted by PDDL2.1 is that intervals are half open on the right, represented as $[A)$. That is to say, the point of change takes the value of the interval on the right. e.g. in this case p is true at that point (since p is true after this point).

Figure 2.2(b) shows an instance where two actions finish at the same time, but have contradictory effects. The question here is what is the truth value of p after these two actions? One solution is to take them in some (arbitrary) predefined order but this does not seem satisfactory. PDDL2.1 adopts the “no moving targets” rule that declares that negative interactions between actions are mutually exclusive (mutex). In this case they must be separated by some small value, ϵ . A theoretical point is how small can this value be before they are again considered mutex? If time is continuous, as it is PDDL2.1, then this value could be infinitely small whilst never letting the two timepoints be equal. To overcome this, the user must set the minimum value of ϵ , where $\epsilon > 0$ or it defaults $\epsilon = 0.1$. This is known as the tolerance value, and specifies the minimum separation distance of two mutually exclusive actions. Through this, PDDL2.1 prohibits some synchronisation.

There is a more practical point to this; any executive carrying out the plan will only be accurate to a certain degree and so precise synchronisation of actions will be impossible to achieve. Therefore, no plan’s validity should rest upon it.

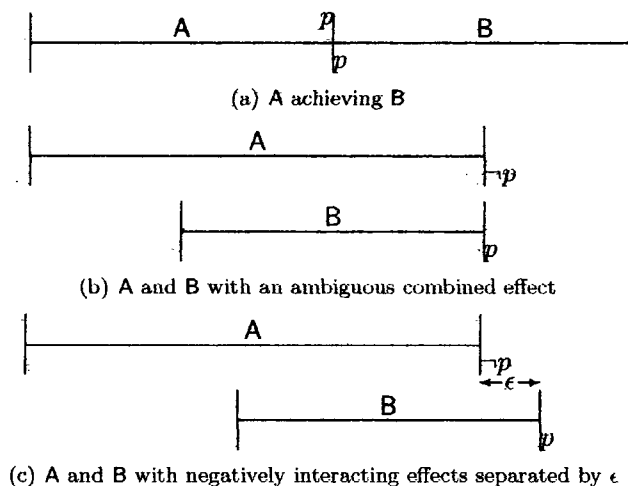


Figure 2.2: Possible Concurrency Issues with Durative Actions

Action Durations

The duration of actions can take one of four forms, increasing in complexity:

Fixed The duration of the operator is fixed and so the same for all instantiations. For example, all fly actions take the same length of time regardless of their start and destination cities (that does not change during planning).

Statically Computed The duration of the action is dependent on its parameters, but not the state of the world. For example, the length of a fly action will depend on the distance between the two cities.

State Dependent The duration of the action is dependent on the state of the world and so will change during the planning and scheduling. For example, completely filling up a tank will take longer the emptier the tank is.

Variable The duration of the action can be chosen by the planner subject to some constraint. For example it is possible to fill up a tank for as long as needed, so long as it is not over filled. The final level of the tank will depend on the length of time for which the action is executed¹.

PDDL2.1 durative actions can express all of these forms.

Other Languages

Some languages (notable those used by Sapa [17] and IxTeT [38]) allow effects to occur at any point during the action, not only at the two end points. It is simple to see these actions

¹In PDDL2.1 these are known as “duration inequalities”

as a form of abstraction, where they take the place of many shorter PDDL2.1 actions, and so can be translated into such actions using “clips” and “magnets” as described in [30]. It is harder to see the state transition approach to time and change with these actions where effects can happen at any time, as it leans further towards the histories view. Furthermore, having only interesting effects happen at the start and end makes it easier to understand the association with the temporal relations described in Section 2.1.2.

LTL (Linear Temporal Logic) is a logic that allows the modalities of \Box (always), \Diamond (eventually) and \bigcirc (next). This allows facts (including goals, conditions and effects) to be expressed in the form “always $p \vee q$ ” (for example). It is possible to convert LTL formulae into PDDL2.1 [14]. TLPlan [3] uses LTL to express domain dependent control rules.

2.1.5 Reasoning About Time

One of the most common ways to reason with time is with a Simple Temporal Network (STN) [15]. These take constraints of the form $b_1 \leq x - y \leq b_2$ where x and y are timepoints. This constraint semantically means that x must follow y by at most b_2 and at least b_1 , i.e. it describes a time interval over which one timepoint can lie in with respect to the other. A special timepoint, X_0 , fixing the beginning of time (the start of a plan) can place timepoints relative to absolute time. Simple precedence constraints can be expressed by setting $b_2 = \infty$ and $b_1 = 0$.

Temporal constraints can be put into a directed weighted graph where the nodes are the timepoints and the edges are the constraints. An inconsistent network (where not all the constraints can be met) is identified by finding negative cycles in the graph, and constraints are propagated by finding shortest paths between points. There are two well known algorithms that do this with negative edges: Bellman-Ford’s [32] which performs Single Source Shortest Path (SSSP) and is of order $O(ne)$, (where n = number of nodes, and e = number of edges), and Floyd-Warshall’s [32], which performs All Pairs Shortest Path (APSP) and is of order $O(e^3)$.

2.2 Resources in Planning and Scheduling

Resources form an integral part of both planning and scheduling but how they are normally modelled is very different. This is partly because they can take many forms. Resources can be qualitative (represented by the state of some object, such as the availability of a machine) or quantitative (often represented by a numeric variable, such as fuel). Quantitative resources are associated with their consumption and production, which can be discrete or continuous. They may be perishable (consumed by the passage of time) and exchangeable (one resource used to replenish another, such as buying fuel with money). Both quantitative and qualitative resources may or may not be renewable. A qualitative resource that can only handle one task is described as unary, and one that can take many, as multi-capacity.

Resources are seen as part of the scheduling problem and represented explicitly. They are then reasoned with directly as this is the scheduling problem - to allocate a known set of activities to available resource whilst respecting precedence, capacity and other constraints.

Planning, however, takes a different view of resources. They are not seen as part of the problem and are mostly represented implicitly. There is no distinction between an object acting as a resource or as part of the planning problem. For example, a truck may be seen as a resource when the goal is to get packages to destinations, but could also be part of the goal, where it itself must be at a particular location. Discrete resources can be modelled in STRIPS, whereas continuous resources need an extension (such as PDDL2.1). [54] examines the role of resources in planning noting that *“they place constraints on the shape and structure of a plan that will have to be met by the planner.”* A non-renewable resource will limit what can be achieved. Perhaps a more general view of what a resource is in planning is as a facilitator object whose actual identity is immaterial for the correctness of the plan [64].

The main disadvantage of representing resources implicitly is that it is difficult to do any specialised reasoning with them. It is also harder to realise the use of alternative resources, and this leads to excessive symmetry in the problem. However, by not representing them explicitly, the system must discover them for itself. When they can do this, they are able to find resources that perhaps the domain designer (incorrectly) did not realise were resources.

As with STNs, algorithms exist to perform consistency checking and propagate resource constraints. These include Timetabling [18], Edge-finding techniques [9] and Precedence Graphs [50] and can be used in conjunction with planning and “time” scheduling techniques as part of a temporal metric planner. They work well with STNs since they take and modify the earliest and latest start times. These can then be propagated back and forth with the STN. Resource-Envelopes [49] work out heuristics based on where resources are tightly constrained in planning problems.

2.3 Decomposition of Problems

Splitting problems into smaller components is a common strategy in computer science. Dividing the problem into smaller instances of the same problem and combining the solutions (“Divide and Conquer”) is intuitively a good idea. Indeed, this is a possible search strategy for planning problems. Rather than into smaller instances of the same problem, temporal planning can be divided into two different problems: classical planning and scheduling. This is not such a common technique since most academic computer science problems are “atomic”; whilst they may be big in the size of the instance, they are structurally small. Planning is “compound” as it can have many other sub-problems encoded within it. For example, a logistics planning problem will contain a route solving problem within it. This section looks at research into how problems integrate in planning and how they can be separated. This is then put into the context of separating planning and scheduling in temporal planning.

2.3.1 HybridSTAN and TIM

Many problems, for example travelling salesman, bin-packing and multi-processor scheduling problems, can be encoded as and within planning problems. HybridSTAN [25], with the aid of TIM [24][52], is able to find these sub-problems and abstract them out. It uses independent sub-solvers for the sub-problems and also solves the remaining parts of the problem. It then combines the solutions, to form a complete plan.

TIM identifies generic types, which are specific kinds of behaviours, examples of which appear in many different planning domains. For example, there is often a form of transportation in a domain, so TIM can identify “mobile” objects, the maps on which they move (static or dynamic) and the actions by which they move round the map. The types are identified even when they are not recognisable as such to a human. Using the information from TIM, HybridSTAN can recognise a sub-problem embedded in the planning problem. Take as an example the travelling salesman problem (TSP). This problem is then abstracted out by changing the actions to ignore parts of the problem that are in the TSP. HybridSTAN then uses a heuristic forward chain planner to start the planning problem. It uses the TSP-solver for two purposes. Firstly it can ask it for heuristic cost estimates, in this case, the cost of moving a mobile object round the map, and use this to contribute to the overall heuristic estimate of a state. Secondly, when it needs a mobile to be at a location, it can ask for a solution from the TSP-solver to move it to that location.

Most relevant to this thesis is the study of the possible interactions of the sub-problems, as identified in [31]:

1. Planning problem is itself a single problem

In this case the entire problem can be solved by the specialised sub-solver. There is a simple layer between the sub-solver and the overall planning system to present the sub-problem in the correct form to the sub-solver and to convert the solution back into a plan.

2. Sub-problem is an independent component of the planning problem

This is the case that has already been described. The sub-problem must be abstracted away from the overall problem, and the sub-solver used both to provide heuristic cost estimates and solutions to the sub-problem.

3. Multiple independent sub-problems

This is a generalisation of the previous case where there is more than one sub-problem.

4. Hierarchical sub-problem dependency

Here the sub-problems are in a strict hierarchy, where one sub-problem is encapsulated in another. Whilst this is slightly more complex, the theory behind it remains the same. In order for the higher sub-solver to provide either a cost or solution to the overall case, it calls on the sub-solvers below it for costs and solutions.

5. Sub-problem interdependency

This is the most difficult case, where the solution to one sub-problem is dependent on the other, and vice versa. This relationship can exist more generally as a cyclic dependency between a collection of sub-problems. This case is not solved by HybridSTAN.

How the sub-problems of planning and scheduling in temporal planning interact in relation to this classification is included in the next chapter.

Results presented in both [25] and [31] show that this is a successful approach to dividing up a problem, both in the quality of the solution and the performance of the planner. The key to this is that it uses specialised sub-solvers to tackle different parts of the problem. A single search strategy is not likely to always be appropriate either to find a solution or heuristic estimate of the cost of reaching the goal state from the current state. Fox and Long note that even in the fifth case, whilst it may be very hard to find a solution where there is an interdependency, it may still be better at producing a heuristic estimate with the interdependencies ignored than the overall solver is.

2.3.2 Translation of the Planning Problem

LPSAT [71] and BLACKBOX [47] both convert planning problems to Satisfiability Problems (SAT) problems, and GPCSP [16] and CPlan [66] both convert to Constraint Satisfaction Problems (CSP). After the translation, the compiled problem is solved and the solution to this is then translated back into a plan. Whilst in all cases the problem is *not* being decomposed in any way, it demonstrates the advantage of having a modular approach to planning. Just as with HybridSTAN, specialised solvers can be used with these planners. Of course the problem is no easier to solve in its new form, but the solvers could be much more advanced than planning technology. Should an improved SAT or CSP solver be written, it can replace the old one and the improvement can immediately be seen in the planning system. This is the same for HybridSTAN and any modular approach: if a better specialised solver is written, it can simply replace the old one and any improvement passed onto the planner.

2.3.3 Goal Orderings as Decomposition

“Goal Agendas” are precedence relationships between goals to determine the order they should be met in such that goals already met in the plan are not deleted when planning for goals later in the agenda. [48] describes a polynomial algorithm to find these. This is a kind of decomposition where each atomic goal is treated as its own sub-problem, thus splitting the overall planning problem into many smaller, potentially easier, planning problems. The algorithm is used in a relaxed form to estimate a goal agenda for the planner FF [45].

SGPlan [10] is a planner that also partitions large planning problems into sub-problems, each with its own sub-goal. Again a goal ordering is found, and the search constrained so

the sub-problems do not interfere and the sub-solutions can be fused into one plan. This again has the advantage of a modular approach since it chooses from a selection of planners (currently LPG [36] and MetricFF [42]) to use the best for each sub-problem. Once again, new planning technologies can be added to the choice as they become available.

2.3.4 Advantages of Decomposition

As discussed, there are two major advantages to problem decomposition, and with planning in particular. The first is that there is a smaller search space for each sub-problem. Backtracking in one of the search spaces does not necessarily mean having to backtrack over decisions made in the other. The second, only really relevant if the problem has been decomposed into different sub-problems, is that specialised solvers can be used on each sub-problem, rather than having one general solver for both. Specialised heuristics and search strategies can be tailored and used for each of the sub-problems. This in turn benefits from a modular approach. Here, once better sub-solvers have been written, they can be plugged in without changing any other part of the system and an improvement is gained by the whole system.

2.4 Integrating Planning and Scheduling Technologies

It has already been noted that realistic temporal planning problems lie somewhere in between the two problems of planning and scheduling. How these two *interact* is studied in the next chapter. Studied here are the problems associated with the *integration* of the two problems. i.e. how planning and scheduling problems can be combined to form temporal planning. This is not necessarily easy for a number of reasons:

- The scheduling problem is not uniquely defined, so it is necessary to decide what version to use, or to use some more generic version.
- The modelling of the two problems have differences, especially the manner in which resources are represented.
- The two problems must use a common model of time and this will affect which technologies can be re-used and integrated from the two areas.
- Scheduling tends to be an optimisation problem requiring the best solution, and is often an over-subscription problem. Little work has been done with planning as an over-subscription problem where there is a choice of goals. These two views need to be brought together for integration.

An example of where integration has succeeded is described in [5] which looks at a formal model for combining planning and production scheduling. [61] considers three different classes of integration.

Stratified Where planning is performed first to decide what actions are needed, and then these are scheduled.

Interleaved Where the two problems are separated, but decisions made in one solver are propagated through to the other.

Homogeneous One problem is turned into the other. Since [61] comes from a scheduling point of view, the examples given are for turning planning problems into scheduling problems where there is no distinction between action choices and ordering decisions.

In PDDL2.1 temporal planning problems, scheduling is introduced into classical planning through the use of durative actions. [11] performs an extensive review of how durative actions can be introduced into classical planning frameworks.

2.5 Planners

In this section, some non-temporal and temporal planners are described to demonstrate some of the aspects presented in this chapter so far, and also to introduce some other ideas common in planning which will become relevant later.

2.5.1 Graphplan-based Temporal Planners

Many successful planning techniques are based on GraphPlan [6] that works by building a planning graph. This is a compacted representation of the search space. It is a directed graph with the nodes in alternate layers of facts and actions. Edges between the nodes in each layer connect preconditions and effects with actions. Each fact layer contains all propositions that could possibly be true at that point and each action layer contains all actions that could be applicable at that point. Fact pairs and action pairs are marked **mutex** if they cannot both be true or applicable in the same layer. “No-ops”, special actions with a single precondition and effect, ensure the persistence of facts over time. There are two distinct phases of GraphPlan: the planning graph is built, and then a plan is extracted through regression search. The planning graph is built from the initial state until all goals appear non-mutex in a fact layer. If a plan cannot be found in the graph, then it is extended some further layers and extraction tried again. Graphplan is sound, complete and optimal (in its makespan).

This planner has been modified a number of times, both to improve its implementation and to extend it to make it more expressive, not least for temporal planning to produce “temporal planning graphs”. It lends itself well to this since non-mutex actions appearing in the same layer could happen concurrently.

TGP

TGP [62] (Temporal Graphplan) is an optimal planner that extends GraphPlan to handle metric time. It uses a restrictive blackbox model of durative actions. A new type of mutex that exists between layers is introduced between actions and preconditions. These are facts that cannot be true whilst an action is being executed (i.e. the invariants of the action). Time is associated with actions and so each action layer represents the same amount of time. This leads to a difficult question; how long should the action layer represent? It really only makes sense to set this to the smallest interval in which “something interesting” will not happen. Any smaller and the graph becomes too big, consuming more memory and taking longer to search. If the interval is any bigger, a possible action that could happen will be missed or the planner is no longer optimal (in terms of the duration of the plan). The solution is to set it to the GCD (Greatest Common Divisor) of the all actions’ durations. If this number is low in comparison to the majority of the actions’ durations the graph built becomes large. For example, in Figure 2.3(a) there is an action A that takes 1000 time units and another, B, that takes 999 time unit, so the GCD is 1. This produces a graph where every 1 time unit is examined to see if anything new could happen. If state dependent durations were allowed (which in TGP they are not), calculating the GCD could be hard to do and be very small since all the actions can be of very different lengths. LPG [36] performs local search on a planning graph of this type.

LPGP

LPGP [53] (Linear Programming Graph Plan) is another planner that extends GraphPlan and uses the richer semantics of PDDL2.1. However, the temporal planning graph associates time with state (i.e. the fact layers), rather than associating time with actions. The action layers are only present when something interesting happens (i.e. the state changes). The plangraph no longer represents the flow of time, but the logical structure of the plan. It does this by splitting up durative actions into two instantaneous actions, one for the start conditions and effects and another for the end conditions and effects. Invariants are kept through invariant actions for which no no-ops are constructed, forcing the planner to put in the invariant action at every layer necessary. This translation converts between an interval-based framework of time and a point-based framework. The duration for each fact layer is not fixed, but solved through constraint satisfaction. This results in it not being dependent on the GCD of the actions’ duration (see Figure 2.3(b)). However, this does mean that it is not optimal in this form without extending the graph further.

TP4

TP4 [40] does *not* directly extract its plan from a temporal planning graph, (it is an extension of HSP — see below) but uses a temporal planning graph similar to LPG where time is associated with action layers. In [39] it describes a novel method for solving problems

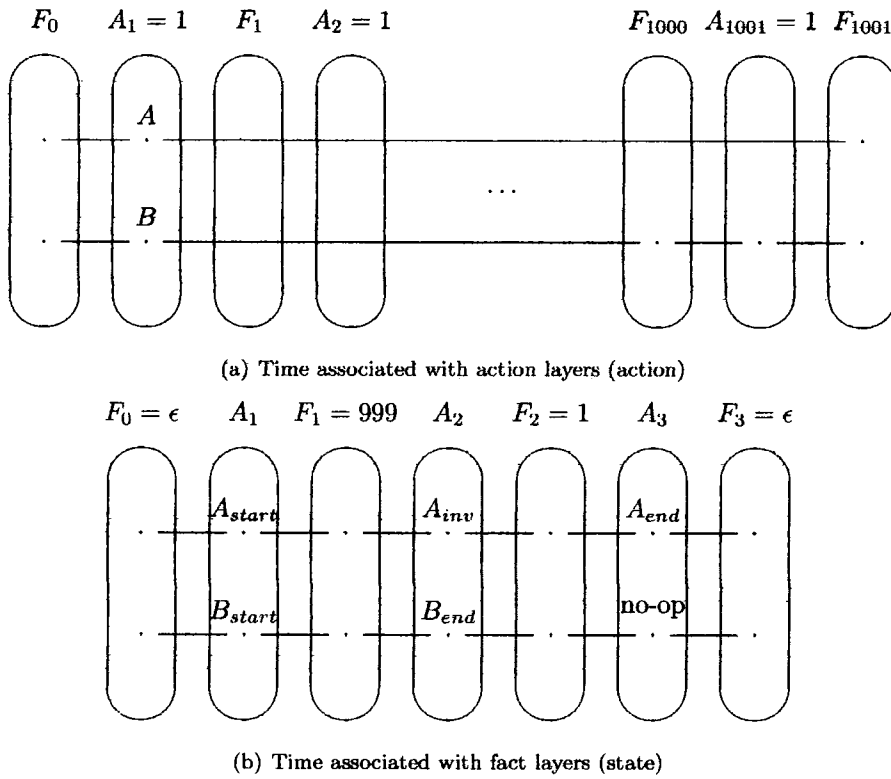


Figure 2.3: Different Types of Temporal Planning Graph

with a low GCD of action durations. Firstly, all action durations are rounded up to the nearest integer. Then the resulting problem is solved using the standard TP4 method (Section 2.5.2). The cost of this solution is an upper bound on the optimal solution cost of the original problem. Finally, action durations are restored to their original values, and a branch-and-bound search, starting from the known upper bound, is used to find the optimal solution.

2.5.2 Forward Heuristic Search

Whereas GraphPlan-based planners perform full systematic search, Heuristic Search planners will often not, but instead rely on good heuristics to guide the search. The consequence of this is usually a trade-off of quality (and especially optimality) for the performance of the planner. In fact, since all planners involve search, all will use a heuristic guide to decide which branches to explore first. Calculating the heuristic functions can be computationally expensive, often in proportion to the accuracy of the guess, so a trade-off is made. Planners described as heuristic search planners do little work with other reasoning functions, so the heuristic function can be relatively complex. If the heuristic is admissible, it is possible

to use a search algorithm such as A* which will still guarantee optimality, but a general admissible heuristic that is also informative is hard to find. Heuristics can be also be used to prune dead end states. In temporal planning, the search space tends to be significantly larger than those in classical planning, implying that the heuristics have to be better and take temporal aspects into account.

HSP and FF

HSP [7] (Heuristic Search Planner) and FF [45] (Fast Forward) both perform heuristically guided forward search from the initial state to the goal state. They both base their heuristics on a relaxed planning graph. This is identical to a regular planning graph but the delete effects of actions are ignored. This has a number of consequences (all proved in [45]):

1. There are no mutexes in the graph, since there are no delete effects.
2. The graph takes polynomial time to build.
3. A (relaxed) plan can be extracted without the need for backtracking (so can be done in “one shot”). This makes this phase also polynomial. Search can be performed to find an optimal relaxed plan, to produce an admissible heuristic, but this is NP-hard to compute.
4. The graph need never be extended.

HSP’s estimates are based on computing weight values for all facts (and so also goals) based on how difficult they are to achieve — assuming all facts are achieved independently — whilst FF find a relaxed plan that can take account of positive interactions between goals (and sub-goals). The number of actions in the plan forms the heuristic cost estimate. The differences between the two heuristics are explored in [46]. Heuristics based on relaxed planning graphs have varying success (as investigated in [43]).

Since FF exploits positive interactions between goals, it is generally considered to be the more successful heuristic approach. Many have incorporated it into their planners and modified it or extended it, including MetricFF [42] (that extends the heuristic to handle metric variables according to PDDL2.1 level 2), MacroFF [8], Marvin [12] (both of which extend the search to include macro operators), Fast Downward [41] (extends the planning graph to deal with causal dependencies) and YAHSP [68] (a search strategy based on the extracted relaxed plan). None of these can yet handle temporal domains.

More TP4 and HSP*_a

TP4 [40] and HSP*_a are both temporal planners that perform regression search using the HSP heuristic. They are both optimal although they assume different semantics to PDDL2.1 in the form of blackbox actions.

Sapa

Sapa [17] searches a set of time stamped states, represented by a tuple $S = (P, M, \Pi, Q, t)$ Where P is the set of propositions true at time t , M is the set of values of metric resources, Π , the set of invariants that must currently remain true and Q , the set of updates scheduled to happen in the future at some point. These states do not just describe the state of the world now, but also the state of planners search; It takes a step towards the histories view of change. An action can be applied if its preconditions are satisfied by P and M , the effects do not interfere with anything in Π or Q and there are no future events that will interfere with the invariants of the action. A special “advance-clock” action is added that can advance the state to the next timepoint in Q .

There are several heuristics that it can be configured to use, all based on a relaxed temporal planning graph. The search is A^* , and some of the heuristics are admissible, making Sapa optimal when they are used.

Sapa does not decompose temporal planning problems into scheduling and planning, but solves both the problems at once. As shall be seen later, this leads to a larger search space. Sapa is unable to handle end conditions and contains bugs not allowing it to correctly solve problems where the scheduling and planning are tightly coupled (although in theory this should not be the case).

2.5.3 Decomposing Planners

In this section, two planners that decompose temporal planning problems are described.

MIPS

MIPS [19][20] is based on model checking methods by compactly representing planning states in binary decision diagrams and then searching the underlying space through A^* search, with the heuristic once again based on relaxed plans. These are then scheduled to improve the heuristic. It is also helped by a pattern database to serve as a domain-independent, admissible heuristic estimate that is computed off-line.

It splits the temporal planning problems into classical planning and scheduling, as suggested could be done earlier in Section 1.1.3. Again, it assumes a loose coupling between the two problems through the use of blackbox actions. It performs two lots of scheduling, firstly on the relaxed plan as part of the heuristic (this allows it to minimise the total duration of the plan) and secondly, after the final plan has been found. For this it performs Critical Path Analysis.

RealPlan

RealPlan [63] does not perform any temporal reasoning (i.e. it cannot solve temporal planning problems) but is interesting as it separates the causal reasoning from the resource

reasoning (resource scheduling). There are two versions, RealPlan-MS (Master-Slave) and RealPlan-PP (Peer-to-Peer) (Figure 2.4). The difference is in how the scheduler of the resources and the planner interact. In both cases, the resources are abstracted out of the domain and translated into a CSP (Constraint Satisfaction Problem), which is then solved by a specialised CSP solver. In the Master-Slave scenario, should the scheduler fail to find a solution to the current context, then that partial plan is not pursued any further. In this version, where all the allocation policies lead to failure, it implies that the causal reasoning and the resource reasoning were, in fact, tightly coupled. In this case, the planner resorts to traditional planning methods where the resource reasoning is not abstracted out. However in the peer-to-peer relationship, the causal reasoning is also translated into a CSP (from a planning graph). Both CSP problems can be solved simultaneously and so should the scheduler not find a solution it can tell the planner why not (i.e. what constraints are broken) and the planner can act accordingly. RealPlan does not use PDDL2.1 as its problem description language.

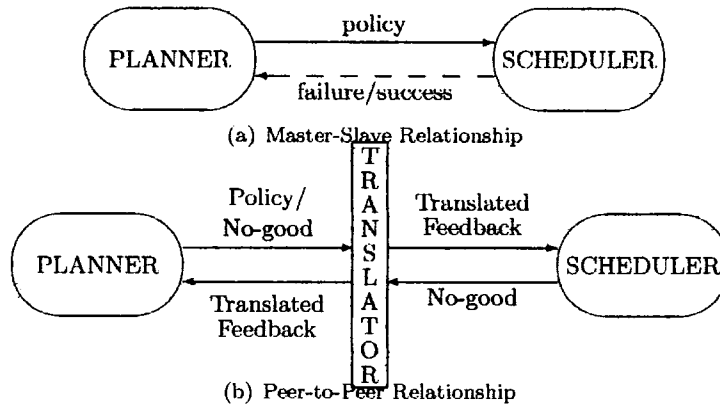


Figure 2.4: Communication in RealPlan

2.5.4 State of the Art

The International Planning Competition has been held 4 times (1998 [56], 2000 [2], 2002 [27], and 2004 [44]) with the aim of comparing current planning technologies. Many problems, differing in size and difficulty, are run for a selection of domains and the performance and quality of solution compared for each planner. There is a time limit and memory restriction on finding a plan. PDDL2.1 (and so, temporal planning) was introduced in 2002, and PDDL2.2 in 2004. Over these two competitions there have been 11 domain independent planners (excluding CRIKEY, the subject of this thesis) that have competed in the temporal domains. These are listed in Table 2.2.

Chapter 5 takes a closer look at the capabilities of these planners, however, none of these temporal planners split the problem into its component parts of planning and scheduling

Table 2.2: State of the Art Temporal Planners

Planner	Description	IPC'02	IPC'04
CPT	a constraint programming based planner.	✗	✓
HSP and TP4	see Section 2.5.1	✗	✓
LPG	local search of “action graphs”, particular sub-graphs of the planning graph representing partial plans. It is non-deterministic, so can be run multiple times and the best solution taken. This results in anytime behaviour.	✓	✓
MIPS	see Section 2.5.3	✓	✗
Optop	an optimal planner performing regression search	✗	✓
P-MEP	an expressive planner that performs A* search, using a relaxed planning graph.	✗	✓
Sapa	see Section 2.5.2	✓	✗
SGPlan	see Section 2.3.3	✗	✓
tilSapa	extension of Sapa to deal with timed initial literals and derived predicates.	✗	✓
TPSys 1&2	builds and repairs plans around a relaxed plan.	✓	✗
VHPOP	a partial order temporal planner.	✓	✗

and can handle the full temporal expressive power of PDDL2.1 (i.e. those that do split the problem, assume a blackbox model of action). There is a good reason for this as blackbox actions assume a loose coupling between the two components of planning and scheduling. Therefore, when the problem is split, the two components are relatively independent of each other and the planner and scheduler need not communicate. If the full temporal semantics of PDDL2.1 are used and the problem split, then the planner and scheduler must communicate and this can be expensive and complex. This is explored further in the next chapter.

As set out in Section 1.3, the aim of this work is to fill this gap. That is, to write a temporal planner that splits planning and scheduling, whilst not assuming a loose coupling between the problems. To avoid the problems of expensive communication between the solvers, a theory is developed as to how the problems are coupled, so as to minimise this communication.

2.6 Chapter Summary

This chapter has looked at the current knowledge in the field of temporal problem solving, in particular in temporal planning. There are various models of time, which differ in what they can and cannot represent. The most common way to integrate planning and time is through the use of durative actions. Many planners described in this chapter use these, and in particular, durative actions defined by PDDL2.1 semantics. All planners are searching some search space, however, through assumptions to this search space they simplify the problems by making the search space smaller and so easier to solve.

Chapter 3

Theory

This chapter examines where planning and scheduling interact in temporal planning problems, and in particular where they are tightly coupled. Through examining where temporal constraints arise in problems (through durative actions and their precedence relationships), new concepts of envelopes and contents, and of minimum and maximum precedence relationships are developed. These are then used to minimise the communication between a planner and scheduler in a new planner described in Chapter 4 that does not assume a loose coupling between the problems, but still solve the problems separately.

3.1 An Initial Solution — The LPGP/FF Hybrid

This first section describes a temporal planning system that separates the planning from the scheduling in PDDL2.1 domains. The communication between the planner and scheduler is one way (Figure 3.1) as there is no feedback from the scheduler to the planner. This is a specific case of the more general case presented back in Figure 1.1. In this system, the two sub-solvers work in a strict sequential order. First the planner solves the planning problem, ignoring all temporal information, selecting actions purely for logical reasons, and then passes this (classical) plan onto the scheduler¹.

Whilst this system has been implemented, it is not proposed as a good solution due to the lack of communication between the solvers. It is presented here to show how the planning can be separated from the scheduling in PDDL2.1 domains, to simplify the explanations, and to help understand where the sub-problems interact.

The architecture for this system is presented in Figure 3.2. Firstly, a temporal planning domain and problem are passed through a translator which takes out the temporal aspects, converting it to an equivalent STRIPS-like domain that preserves all the key temporal relationships. Durative actions are split into three instantaneous actions, representing the start of the action, the end of the action and the invariant. It stores the duration of the

¹This is stratified integration as classified in Section 2.4

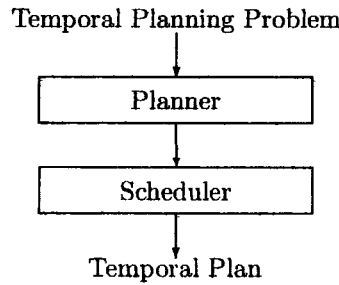


Figure 3.1: The Proposed Separation of Planning and Scheduling in the Hybrid Planner

actions in a separate file. The translated problem is passed through a classical planner, in this case FF. This is where the ‘hard’ work is done. The resulting totally ordered plan is passed through a program that lifts a partially ordered plan, allowing actions that can be executed together to happen concurrently. The partial ordering, along with the duration file created by the translator, are put as constraints into a Simple Temporal Network (STN). This schedules the plan by calculating the relative and actual timings of the actions to produce a valid temporal plan.

Each box is now taken in turn and explained in more detail.

LPGP Translator

The translator is taken from the LPGP planner (as described in Section 2.5.1). It takes in domain files and separates the durative actions into three separate instantaneous STRIPS actions: a start action, an end action and an invariant action. The start action takes the start conditions and start effects of the durative action, and the end action takes the end conditions and end effects. The invariant action has the durative action’s invariants as preconditions. This translation takes the model of time from interval-based to point-based, as described in Section 2.1.2. The interval between the end points is represented by the invariant action.

STRIPS and durative actions are defined followed by the translation between them.

Definition 3.1 — STRIPS action

An instantaneous STRIPS action operator o is a triple

$$o = (cond, add, del)$$

where each element is a set of propositions. *cond* is the set of logical preconditions, *add* is the set of add effects (propositions that become true after execution of the action), and *del* is the set of delete effects (propositions that become false after the execution of the action).

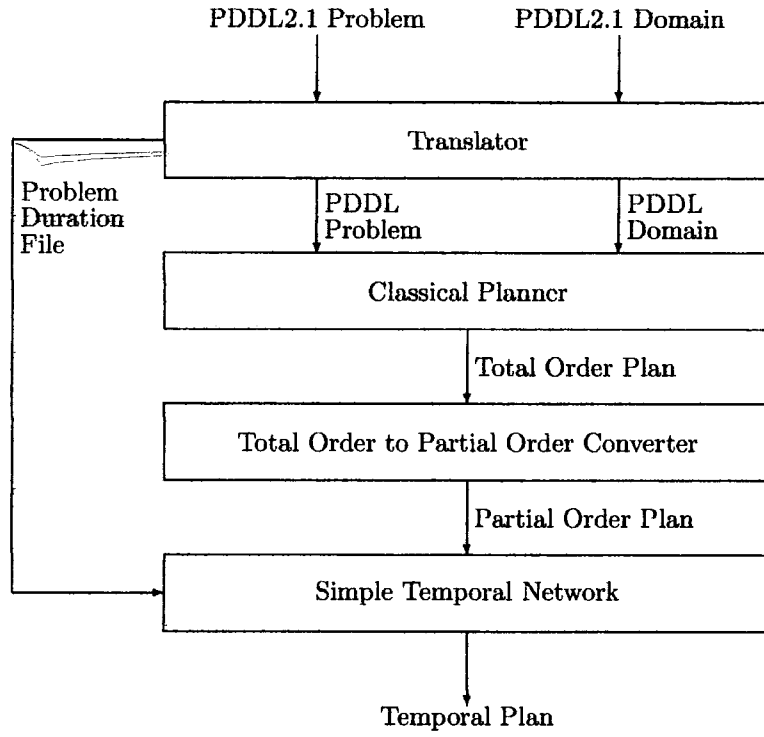


Figure 3.2: Architecture for Separating Planning and Scheduling

Definition 3.2 — Durative Action

A Durative Action operator da is a quad-tuple

$$da = (ta_cond, ta_add, ta_del, dur)$$

where the first three elements are a many-to-many mapping from propositions to time annotations

ta_cond : proposition $\leftrightarrow \{at\ start, at\ end, over\ all\}$

ta_add : proposition $\leftrightarrow \{at\ start, at\ end\}$

ta_del : proposition $\leftrightarrow \{at\ start, at\ end\}$

ta_cond are the time annotated conditions of the actions, ta_add are the time annotated add effects and ta_del are the time annotated delete effects.

For each mapping ta_map , we define

$$\begin{aligned} ta_map_{-} &= \{x \cdot \text{proposition} \mid ta_map(x) = \text{at start}\} \\ ta_map_{\leftrightarrow} &= \{x \cdot \text{proposition} \mid ta_map(x) = \text{over all}\} \\ ta_map_{-} &= \{x \cdot \text{proposition} \mid ta_map(x) = \text{at end}\} \end{aligned}$$

dur is the duration of the action where

$$dur \in \mathbb{R}^{+}$$

Definition 3.3 — LPGP Action Translation

A Durative Action da is split into 3 STRIPS actions, da_{-} , da_{\leftrightarrow} and da_{-} .

$da_{-} = (cond_{-}, add_{-}, del_{-})$ is

$$\begin{aligned} cond_{-} &= ta_cond_{-} \\ add_{-} &= ta_add_{-} \\ del_{-} &= ta_del_{-} \cup \{\text{Action_Name_inv}\} \end{aligned}$$

$da_{\leftrightarrow} = (cond_{\leftrightarrow}, add_{\leftrightarrow}, del_{\leftrightarrow})$ is

$$\begin{aligned} cond_{\leftrightarrow} &= ta_cond_{\leftrightarrow} \cup \{\text{Action_Name_inv}\} \\ add_{\leftrightarrow} &= \{\text{Action_Name_inv}, i_Action_Name_inv\} \\ del_{\leftrightarrow} &= \emptyset \end{aligned}$$

$da_{-} = (cond_{-}, add_{-}, del_{-})$ is

$$\begin{aligned} cond_{-} &= ta_cond_{-} \cup \{\text{Action_Name_inv}, i_Action_Name_inv\} \\ add_{-} &= ta_add_{-} \\ del_{-} &= ta_del_{-} \cup \{\text{Action_Name_inv}, i_Action_Name_inv\} \end{aligned}$$

The durations file is a function df that contains an entry for each durative action and its corresponding duration

$$df : da \rightarrow \mathbb{R}^{+}$$

a_+ , a_+ and a_- are the corresponding split actions for a durative action a whose duration is a_{dur} .

Durations can be computed durations (based on a function of a static fluent) but not state dependant.

An example durative action is that of loading a truck. This has a duration, as it takes time to complete this action. To carry out this action successfully, it is necessary that the object that is to be loaded into the truck must be at the location where the loading is to take place at the start of the action. For the duration of the action, the truck must also remain at this location and this is modelled as an invariant. Immediately after the start it can be considered that the object is no longer at the location; this stops it being loaded into two trucks at once. However, only at the end of the action will the object be in the truck.

Figure 3.3 illustrates the split of the LOAD_TRUCK durative action from the driver-log domain. The full example translation of the domain is given in Appendix A. In the LOAD_TRUCK start action, it is a precondition that the object is at the location where the loading is to take place. After the start, the object is no longer considered to be at this location (as it is being loaded). For the duration of the action (as represented by the invariant action), it is a condition that the truck stays at the location where the loading is taking place. At the end of the durative action (as represented by the end action) it is an effect that the object is now inside the truck.

There are two extra dummy propositions added during the conversion process. The first, Action_Name-inv, is an effect of the start and invariant action, and a condition of the invariant and end action. The second, iAction_Name-inv, is an effect of the invariant action and a condition of the end action. These ensure that if an end action is chosen, then so is the corresponding invariant, and similarly, if an invariant is chosen, then so is the corresponding start action. This works as follows:

The dummy propositions take the parameters of the durative actions and are unique to the three split actions. Action_Name-inv is precondition of the invariant action and end action. For these actions to be applicable, Action_Name-inv must be true and this can only be achieved by the start action. Therefore an invariant and end action can *only* be present in the plan if the start action has already been applied and achieved this condition. If not, the preconditions for these actions are not met. The same logic applies for iAction_Name-inv and the invariant and end action.

Both dummy propositions are deleted by the end action. This is required so multiple identical invariant and end actions are not put in the plan without another corresponding start action. Deleting the dummy propositions ensures that a new start action is needed to achieve new invariant and end actions (as before).

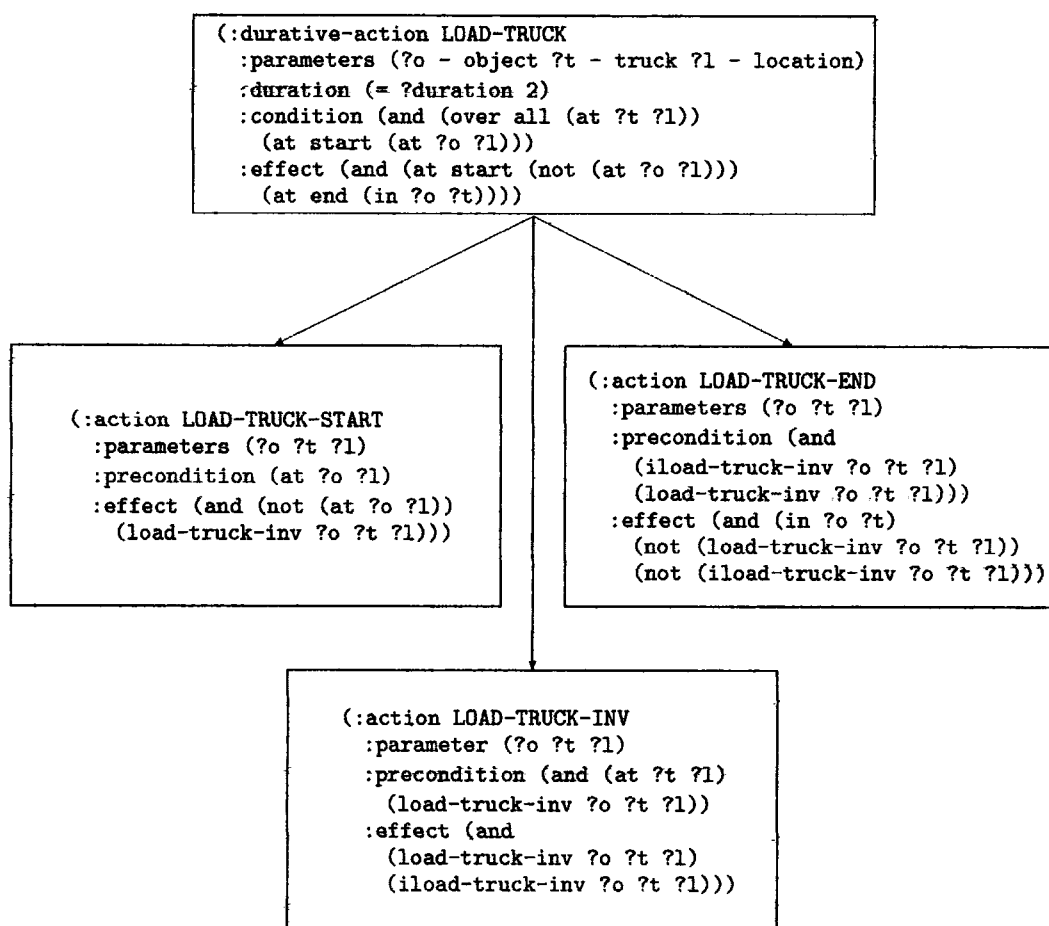


Figure 3.3: The LPGP Translation of Durative Actions

The Planner

This translated domain with the problem file is then passed to the classical planner, in this case FF, the working of which is described in Section 2.5.2. Any classical planner would suffice here.

The Partial Order Lifter

The Partial Order Lifter takes the totally ordered plan produced by FF and converts it into a partially ordered plan. This is an implementation of the Veloso algorithm [67] (sketched out in Figure 3.4) that takes advantage of the given total ordering of the plan by visiting only earlier actions in the plan on each iteration of the algorithm, and then removes unnecessary precedence orderings from the total order to produce a partial order. The total order plan is a valid partial order plan, and so in the worst case no precedence orderings will be removed

and it is this total order that will be returned. This algorithm finds concurrency where possible.

Input: TO-Plan: A list of actions $\langle a_1, \dots, a_n \rangle$
Output: PO-Plan: A set of orderings between actions $\{a_i \prec a_j\}$

```

for  $i = n$  down-to 1 do
  (a) for each  $p \in \text{precond}(a_i)$  do
    Find an action  $a_j$  where  $p \in \text{add}(a_j)$ 
    Add an ordering  $a_j \prec a_i$ 
  (b) for each  $d \in \text{del}(a_i)$  do
    Find all actions  $a_j$  where  $d \in \text{precond}(a_j)$ 
    Add an ordering from all actions  $a_j \prec a_i$ 
  (c) for each  $p \in \text{primary\_add}(a_i)$  (in the goal or sub-goal chain) do
    Find all actions  $a_j$  where  $p \in \text{del}(a_j)$ 
    Add an ordering from all actions  $a_j \prec a_i$ 

```

Figure 3.4: The Veloso Algorithm to Translate Totally Ordered Plans to Partially Ordered Plans

Note that this is still operating on the translated split durative actions, and so will find the correct orderings between the start, invariant and end action triplets, due to the dummy propositions.

The Veloso algorithm is a greedy polynomial algorithm that does not necessarily find the best (temporally shortest) partial order. However, step (a) is a choice point as there could be a number of achievers for an action, of which only the latest in the plan is used. Search could be performed here to find a better ordering using different achievers.

The STN

The start and end actions represent instantaneous moments of time but the invariant action represents an interval of time (between the two end points). STNs, however, only reason with instantaneous timepoints. Before the partial order can be converted into an STN, precedence relations involving invariant actions must be converted to use their corresponding end points:

$$\begin{aligned}
 a_{i\leftrightarrow} \prec a_j &\Rightarrow a_{i-} \preceq a_j \\
 a_i \prec a_{j\leftrightarrow} &\Rightarrow a_i \preceq a_{j-} \\
 a_{i\leftrightarrow} \prec a_{j\leftrightarrow} &\Rightarrow a_{i-} \preceq a_{j-}
 \end{aligned}$$

In a precedence relationship $(a_i \prec a_j)$, should one action (a_j) follow an invariant action (a_i) then this action a_j should now follow the whole interval that a_i represents. The interval's latest point is just before the end of the durative action, and so the precedence relationship is changed to follow this end.

Conversely, should an action (a_i) precede an invariant action (a_j), it must precede the whole interval that a_j represents. The reason that the precedence relation turns from a strict ordering (\prec) to a simple or equal to ordering (\preceq) is the PDDL2.1 semantics state that invariants hold from just after the start to just before the end. Thus the orderings are not between the end points, but rather either side of them.

Definition 3.4 — Conversion of Partial Order to STN

A Partial Order $pop = (ia, pr)$ where ia is a set of instantaneous STRIPS Actions and pr is a set of precedence relations between the members of ia , is converted into a set of temporal constraints tc such that

- (a) $\forall a_i \prec a_j \in pr \cdot \{\varepsilon \leq a_j - a_i \leq \infty\} \in tc$
- (b) $\forall a_i \preceq a_j \in pr \cdot \{0 \leq a_j - a_i \leq \infty\} \in tc$
- (c) $\forall a_i \in ia \cdot \{\varepsilon \leq a_j - X_0 \leq \infty\} \in tc$
- (d) $\forall a_{\vdash} \in ia \cdot \{a_{dur} \leq a_{\vdash} - a_{\vdash} \leq a_{dur}\} \in tc$

where $X_0 = 0$ and represents the start of the plan.

Part (a) of Definition 3.4 ensures that timepoints that are in strict precedence must be separated by at least ε (the tolerance value), reasons for which are described in Section 2.1.4. Timepoints that are not in strict precedence can happen simultaneously (part (b)). Part (c) constrains each action to start after the start of the plan (X_0). Each corresponding start and end action must have a constraint, made by part (d), for their duration, which is read from the duration file produced by the LPGP translator. These constraints take the model of time from a point-based, back to an interval-based model.

To calculate the earliest possible start time for each action, the shortest distance must be found between the action's start timepoint and X_0 in the network. Floyd-Warshall's All Pairs Shortest Path algorithm is used once and is of complexity $O(n^3)$ (where n is the number of timepoints.) Bellman-Ford's Single Source Shortest Path would have to be used repeatedly — once for each action ($\frac{n}{2}$) — making the complexity $O(\frac{n}{2}) \times O(ne) = O(n^2e)$ (where e is the number of constraints). Since there are *at least* n constraints (one for each timepoint to make it follow the start - see part (d) of Definition 3.4) this makes the complexity at least $O(n^2n) = O(n^3)$. As there will be in fact more constraints from precedence relations and duration constraints, the complexity will be greater than this, and so it is less complex to use Floyd-Warshall's.

It should be noted that this system could potentially not respect invariants correctly. In Figure 3.5, a start (A_{\vdash}) and invariant action (A_{\vdash}) are put in the plan, followed by an action (B) that breaks the invariant condition, s , before the end action (A_{\vdash}). Even if the invariants become conditions of the end action, it would still be possible for the invariants to

be broken and then re-achieved. This is because the translation of the durative action treats the invariant as a single point of time, when it should actually be an interval. Therefore, in Figure 3.5, FF produces a “valid” total order classical plan for the translated domain, however, when this is passed through the partial order lifter and scheduled, it produces an invalid temporal plan with respect to the original temporal domain, since the invariant s of action A has been broken.

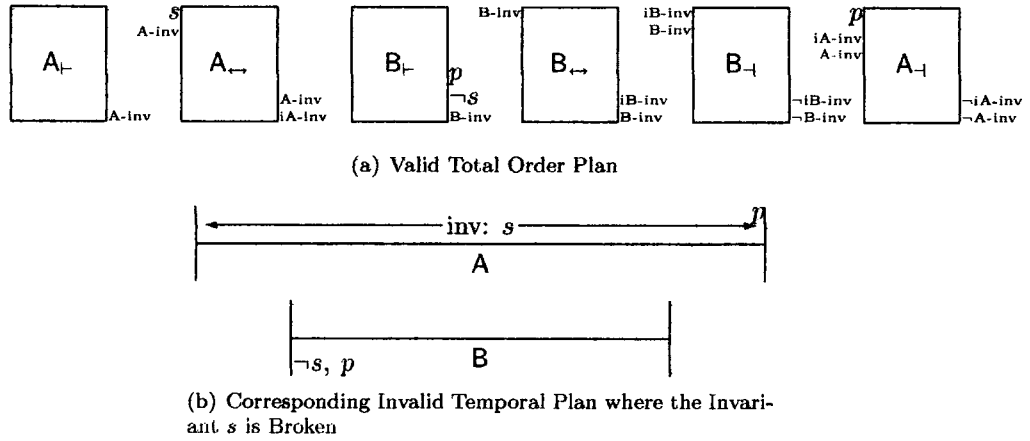


Figure 3.5: Example of a Broken Invariant

A further problem with this hybrid is in the way in which the dummy propositions operate in the translation. Whilst there cannot be an end action without a start, there could be a start in the plan without its end. This is against PDDL2.1 semantics as all actions must complete, so a post processing step is needed to ensure that an invariant and end action are put in the plan for each start action if necessary. This is how LPGP handles these cases. However, this is not suitable if the end action then deletes a goal (as shown in Figure 3.6).

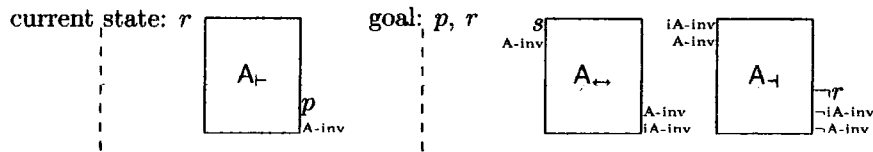


Figure 3.6: Example of an End Action Deleting a Goal

This is an initial solution to the temporal planning problem which demonstrates how to separate the planning from the scheduling in PDDL2.1 domains by taking a point based representation and planning using only logical reasoning, and then re-introducing temporal aspects. The main advantage is demonstrated by its modular approach; FF could be replaced

with any classical planner, and the partial order lifter and STN also could be replaced with equivalent scheduling algorithms. By decomposing the problem, it is searching smaller state spaces than if it were to combine the problems. The planner and scheduler do not communicate with one another and the weaknesses of this is analysed later. First the structure of temporal planning domains is investigated.

3.2 Coupling of Planning and Scheduling

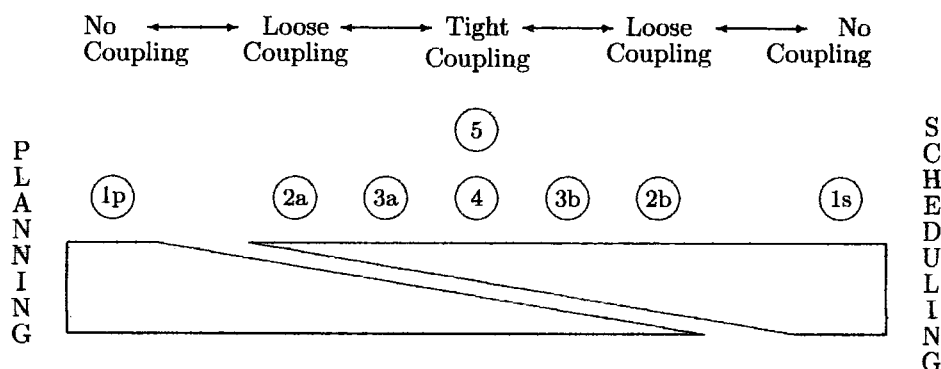


Figure 3.7: Coupling Between Planning and Scheduling in Temporal Planning Domains

Figure 3.7² illustrates the spectrum of coupling between planning and scheduling in temporal planning domains. The coupling increases with the number. This spectrum is compared with the sub-problem interactions classified for TIM, as described in Section 2.3.1. In this case, the main problem is the temporal planning problem and the sub-problems are planning and scheduling.

On the left (1p) are pure planning problems that contain no scheduling. These include classical planning benchmark domains. On the far right (1s) there are the domains that are completely scheduling problems that have been encoded as planning problems, where there is no choice of actions. Each goal is achievable by *one* action that the planner *must* choose. These problems (1p and 1s) refer to interactions of type 1 on TIM's classification, where either the planning or the scheduling is a complete sub-problem of temporal planning.

Domains in 2 represent problems where there is a component from the other sub-problem but this is easily solved and has no consequence either on satisfiability or on the quality of the other the problem. For example, a domain in 2b will be predominantly a scheduling task with a planning component where the choice of actions is easy and has no effect on

²2a and 2b are equivalent, as are 3a and 3b since the planner must always choose the actions before they can be scheduled. However, they are separated here, partly for symmetry reasons and partly to demonstrate how a problem may be more planning centric or more scheduling centric.

the schedule. Here there is no coupling between the problems and an example would be the problem of building a house as set out in Section 1.2. These domains are associated with interactions of type 2 and 3 on TIM's classification, where the scheduling is an independent component of planning.

Domains of type 3 have a loose coupling where the solution to one problem only affects the quality of the solution to the other, and not the satisfiability. All the problems in the IPC'02 [27] were of this type, where the choice of action affected only the quality of the schedule produced. An example is the ZenoTravel Time domain (see Appendix B) that has two fly actions: one for flying fast, and the other for flying slowly that uses less fuel, but is a longer action. The choice of action (i.e. whether to fly fast or slow) affects the quality of the schedule that is produced, but a schedule can always be found for the plan. Concurrency in these domains *may* occur in order to produce a better schedule. A consequence of this is that all plans to problems in the first three levels of this spectrum can be sequentialised so that there is a complete ordering between the actions with no concurrency. Domains of this type are classified as type 4 (hierarchical sub-problem dependency) on TIM's classification, where the scheduling is an dependent component of planning.

The tightest coupling happens in domains in the centre of the spectrum (4 and 5) where concurrency *must* happen and is of type 5 on the classification TIM uses, as the sub-problems are interdependent. 4 refers to TEG domains where the concurrency *must* happen only in order to achieve the goals by their deadlines. This is a similar coupling to 3, but the quality of the solution is now a hard constraint that must be met. In domains of type 5, concurrency *must* happen, not to achieve a goal by a deadline, but to achieve a goal at all. The coupling between planning and scheduling is stronger in domains of type 5 since the concurrent actions interact, whereas they do not in domains of type 4; This interaction is co-ordination.

Definition 3.5 — Co-ordination

“Co-ordination” occurs where actions which, when executed concurrently, interact to produce an interesting effect.

An example of co-ordination as present in a domain of type 5 occurs in the match domain (Appendix C, a variant of which was first presented in [53]) where the goal is to mend fuses. To mend a fuse (with the MEND_FUSE action), there must be light for the duration of the action. This is achieved by lighting a match (with the LIGHT_MATCH action) which provides light only whilst it burns (i.e. for the duration of the action). To mend a fuse you must also have a hand free, the effect of which is that you can only fix one fuse at a time.

Where the LIGHT_MATCH action is 8 time units long and the MEND_FUSE action is 5 time units long, it should be obvious that two matches will be needed, since both fuses cannot be fixed by the light of one match before it burns out. However, if the fuses take less time to fix, the matches burn for longer, or fuses can be fixed concurrently, then a different number of matches may be required. Importantly, the MEND_FUSE actions *must*

be executed (and completed) during the execution of the `LIGHT_MATCH` action. These actions must be co-ordinated (i.e. happen concurrently and in the correct order) so that the goal of fixing the fuse is reached. Figure 3.8 is a valid plan for the problem.³

```
0.01: (LIGHT_MATCH match1) [8.0]
0.02: (MEND_FUSE fuse1 match1) [5.0]
2.04: (LIGHT_MATCH match2) [8.0]
5.03: (MEND_FUSE fuse2 match2) [5.0]
```

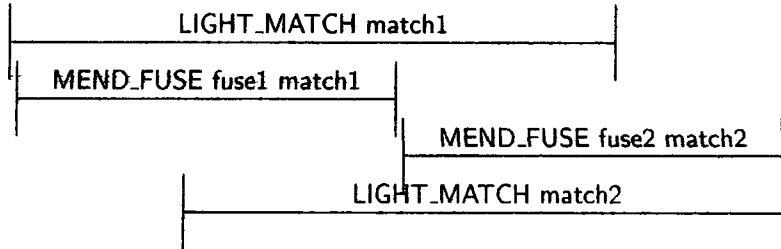


Figure 3.8: A Valid Plan for the Match Problem

Temporal planning domains in IPC'04 [44] where the new features of PDDL2.2 were not used were of type 3. Variants where the new feature of timed initial literals were used, were of type 4. Once compiled down to PDDL2.1 they become of type 5 as the dummy actions required co-ordination.

An alternative view of the spectrum in Figure 3.7 is in terms of “constrainedness”. Generally, the more constrained a problem, the harder it is. In respect to temporal planning problems, the more constraints there are between the planning and scheduling, the tighter they are coupled and the harder the problems become. Domains of types 1 and 2 have no constraints between the problems whereas domains of types 4 and 5 have many.

3.2.1 Failure of the LPGP/FF Hybrid

The system described at the beginning of this chapter is capable of planning for all domains of type 1, 2, and 3, where there is no forced concurrency. The LPGP/FF Hybrid cannot produce valid plans where the two problems are tightly coupled (i.e. types 4 and 5). In this section the failure of the system and the reason for this un-soundness is examined to gain an understanding of where co-ordination arises in domains of this type and how to handle it best. In the match domain, FF produces the plan:

³It is valid according to the problem specification, even if semantically it is odd.

```

(LIGHT_MATCH.start match1)
(LIGHT_MATCH.inv match1)
(MEND_FUSE.start fuse1 match1)
(MEND_FUSE.inv fuse1 match1)
(MEND_FUSE.end fuse1 match1)
(MEND_FUSE.start fuse2 match1)
(MEND_FUSE.inv fuse2 match1)
(MEND_FUSE.end fuse2 match1)
(LIGHT_MATCH.end match1)

```

The partial order lifter produces the constraints:

$$\begin{aligned}
\varepsilon &\leq (\text{MEND_FUSE.start fuse2}) - (\text{MEND_FUSE.end fuse1}) \leq \infty \\
\varepsilon &\leq (\text{MEND_FUSE.start fuse1}) - (\text{LIGHT_MATCH.start match1}) \leq \infty \\
\varepsilon &\leq (\text{LIGHT_MATCH.end match1}) - (\text{MEND_FUSE.end fuse1}) \leq \infty \\
\varepsilon &\leq (\text{MEND_FUSE.start fuse2}) - (\text{LIGHT_MATCH.start match1}) \leq \infty \\
\varepsilon &\leq (\text{LIGHT_MATCH.end match1}) - (\text{MEND_FUSE.end fuse2}) \leq \infty \\
\varepsilon &\leq (\text{LIGHT_MATCH.end match1}) - (\text{LIGHT_MATCH.start match1}) \leq 8 \\
\varepsilon &\leq (\text{MEND_FUSE.end fuse1}) - (\text{MEND_FUSE.start fuse1}) \leq 5 \\
\varepsilon &\leq (\text{MEND_FUSE.end fuse2}) - (\text{MEND_FUSE.start fuse2}) \leq 5
\end{aligned}$$

Finally, the STN finds this set of constraints to be inconsistent and, as there is no feedback to the planner, the system fails. The reason for the inconsistency is that two matches are needed in order to have enough time to fix both fuses, but since all temporal information is ignored whilst planning, it failed to realise this, trying instead to fix both fuses by the light of one match. Communication is needed between the planner and the scheduler at this point.

The rest of this chapter looks at where co-ordination occurs in temporal planning domains and this is then used to minimise communication between the planner and the scheduler.

3.3 Temporal Constraints in PDDL2.1

The reasoning in this section is restricted to PDDL2.1 where logical change can only happen at the start of a durative action, or on its completion. Unscheduleable plans come from temporal constraints in the problem that cannot be met. In PDDL2.1, temporal constraints are not represented explicitly, but rather implicitly, using other, potentially dummy, durative actions, as these are the only way to represent temporal information in the problem. What follows is a review of possible constraints that can be represented in PDDL2.1 and the different ways that these constraints can be expressed.

Temporal constraints take the form (or can be rearranged to) $x - y \{ \leq, <, \geq, > \} b$, where x and y are the actual times of the start or end points of actions, and so their difference $(x - y)$ is how far apart in time they are relatively. b gives the maximum or minimum (depending on whether it is greater than or less than) that this difference can be.

All other constraints that do not use disjunctions are specialised cases of these. For example, to represent an exogenous event e that occurs at a particular time, t , y is simply set to zero and the constraint becomes

$$t \leq e - 0 \leq t$$

To represent a deadline, d , that some end point, e , must happen by, y is again set to zero and constraint is

$$e - 0 \leq d$$

Figure 3.9 (a) and (b) show diagrammatically how the constraint $B - A \leq b$, which semantically represents the maximum time by which B follows A, can be enforced using a dummy action⁴. Figure 3.9 (c) and (d) shows how the constraint $B - A \geq b$ can be encoded, and represents the minimum time by which B follows A.

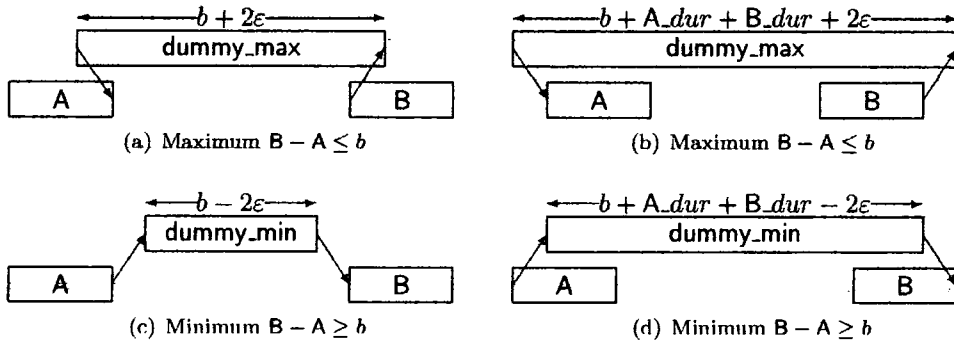


Figure 3.9: Expressing a Maximum Minimum Elapsed Time Between Actions in PDDL2.1

In each case the duration of the dummy action has been extended or reduced by 2ϵ . This is because the PDDL2.1 semantics dictate they must be separated by a small amount (as discussed in Section 2.1.4). When calculating the duration of the dummy action, two gaps between the dummy action, and A and B must be compensated for. For the rest of the explanation ϵ is omitted from the reasoning to ease the complexity, but can easily be reintroduced.

For both figures (b) and (d) the dummy action's duration must also have the duration of A and B summed on to it. This is because the dummy action now encapsulates both of these actions.

These figures have been arranged in a fashion that should make obvious the similarities both within the different representations of the same constraint, and also the similarities

⁴Here $x - y \leq b$ can be rearranged as $y - x \geq -b$, which of course means exactly the same. Other constraints can be equally rearranged, however, since an action cannot have a negative duration, all constraints are kept in the form that keeps b non-negative.

in representing the different constraints. The different representation can be “mixed and matched” within themselves, as shown in Figure 3.10.

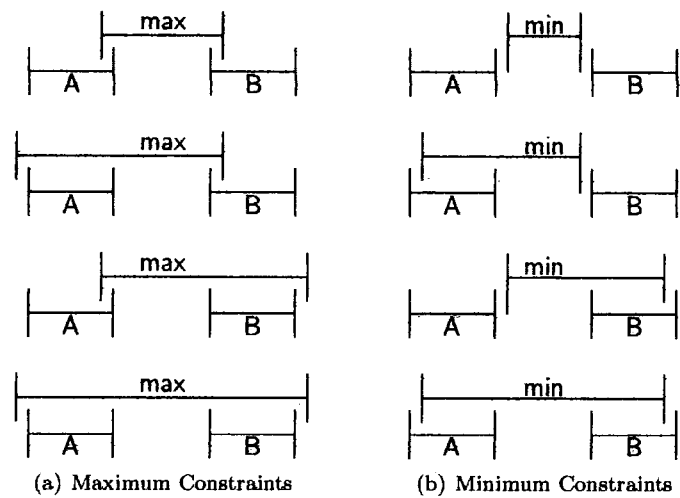


Figure 3.10: Possible Combinations of Representing the Same Constraint

Regardless of the form used, when expressing a maximum time between actions, the ordering is from the start of the dummy action to A, and from B to the end of the dummy action, whereas when expressing a minimum time, regardless of the form used, the ordering is from A to the start of the dummy action and then from the end of the dummy action to B.

if $B - A \leq b$
then $\text{dummy_max}_+ \prec A$
 $B \prec \text{dummy_max}_+$

if $B - A \geq b$
then $A \prec \text{dummy_max}_+$
 $\text{dummy_max}_+ \prec B$

Notably, in maximum constraint orderings, there are *no* precedence relations where an end action precedes a start action and in minimum constraint orderings, there are *no* precedence relations where a start action precedes an end action. In both cases, ends precede ends and starts precede starts.

From this observation, two new precedence relationships (\prec^{max} and \prec^{min}) are defined.

Definition 3.6 — Maximum Precedence Relationship

Maximum Precedence Relationship between two action end points i and j where $i \prec j$ is defined as:

$$i \prec^{max} j = \begin{array}{l} i_+ \prec j_- \\ \vee \quad i_+ \prec j_+ \\ \vee \quad i_- \prec j_- \end{array}$$

Definition 3.7 — Minimum Precedence Relationship

Minimum Precedence Relationship between two action end points i and j where $i \prec j$ is defined as:

$$i \prec^{min} j = \begin{array}{l} i_- \prec j_- \\ \vee \quad i_+ \prec j_+ \\ \vee \quad i_- \prec j_+ \end{array}$$

The maximum precedence relationships occur in maximum temporal constraints (Figure 3.10(a)) and the minimum precedence relationships occur in minimum temporal constraints (Figure 3.10(b)). These two definitions have an intersections of types (an end can precede an end, and a start can precede a start), however a maximum temporal constraint does not have any ends forced to precede a start (as in a minimum temporal constraint) and a minimum temporal constraint does not have any starts that must precede another action's end (except, in both cases, through transitive relationships).

Where there are maximum constraints with no minimum, B could happen before A, and of course with minimum constraints, B could happen infinitely after A without breaking the constraint. The more interesting cases occur when both a maximum and minimum time occur, i.e. where the constraints are combined to the form $b_1 \leq x - y \leq b_2$. To form these constraints, the minimum and maximum constraints are simply combined in any combination. Two possibilities are shown in Figure 3.11.

Of course, for it to be possible for these constraints with both maximum and minimum time differences to be met, the duration of min must be less than or equal to the duration of max.

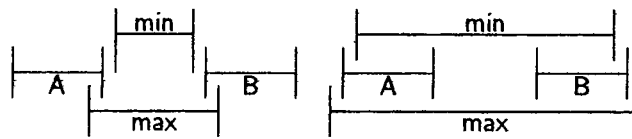


Figure 3.11: Expressing both Minimum and Maximum Time Between Actions in PDDL2.1

3.3.1 Translation of the Domain

Consider another domain which involves making a cup of tea. There must be a maximum time between boiling the water and pouring it into the mug (or else the water cools, and tea cannot be made with cold water). There is also a minimum time set between the boiling and pouring of the water (to avoid steam burns). This could be expressed in two ways; either by the first *clipping* method, or the second *enveloping* method (as seen in Figure 3.12).

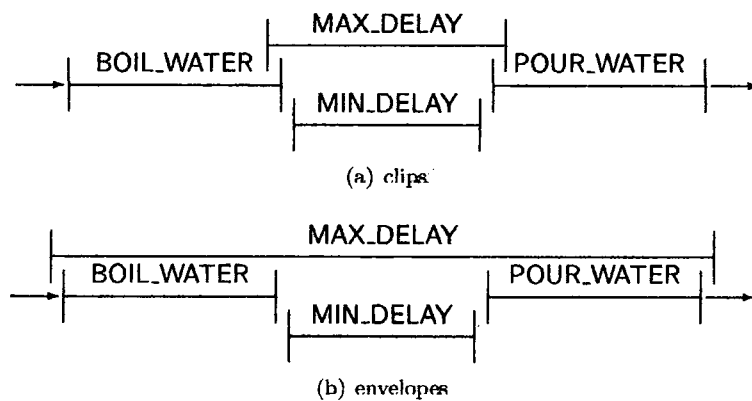


Figure 3.12: Two Possible Equivalent Representation of the Breakfast Domain

As shown in the previous section, these two representations are equivalent. If the BOIL_WATER action in fact has a greater duration (for example if more water is put in the kettle), the LPGP/FF Hybrid would not produce an un-schedulable plan with the clip method, whereas if envelopes are used there is a danger that this could happen. This would occur where the MAX_DELAY action is not long enough to include the BOIL_WATER, the MIN_DELAY and the POUR_WATER actions. The dummy action's duration relies on the duration of the other actions where an envelope is used (see previous Figure 3.9), whereas it does not where clips are used (with the exception of the MIN_DELAY action. It is fair to assume that the domain designer ensures that $\text{MIN_DELAY}_{dur} < \text{MAX_DELAY}_{dur}$.) However, if the duration of the BOIL_WATER or POUR_WATER actions were to change then either the duration of the envelope would have to change to keep the constraint the same, or the constraint would change meaning accordingly.

The clipping method is therefore better for encoding maximum constraints as it does not

rely on the duration of the actions it is trying to constrain. The LPGP/FF Hybrid would be able to produce a valid temporal plan for the clipping method, but not necessarily for the envelope method, if the `MAX_DELAY` action were not sufficiently long enough to contain all actions. Could it be possible to detect such cases with envelopes and translate the problem to use the easier clipping method? In this particular case it would seem so.

Returning to the match domain, it is possible to change it such that there is a delay between fixing the two fuses (in order to get the fuse out of its packet) and also so that it is possible to fix two fuses with one match (i.e. the `LIGHT_MATCH` action is longer). When this is done, it looks identical in structure to the domain where tea is made (see Figure 3.13 (a) and (b)). Would it then be possible to translate this domain, as was done previously, into a form in using clips? This would mean changing the duration and structure of the `LIGHT_MATCH` action, which would result in a plan as in Figure 3.13 (c). This semantically does not make sense, but this plan could be translated back again in a post processing step to the original form. Whilst this guarantees that there is enough time to fix the fuses, problems arise if a third action relies on the duration or structure of the `LIGHT_MATCH` action. Clearly, if the `LIGHT_MATCH` action changed duration, there would be difficulties with anything else requiring that light. Also it is unclear how the translation would work in the case where there are three or more fuses to fix. Bearing all this evidence in mind, this is not a viable solution to co-ordination.

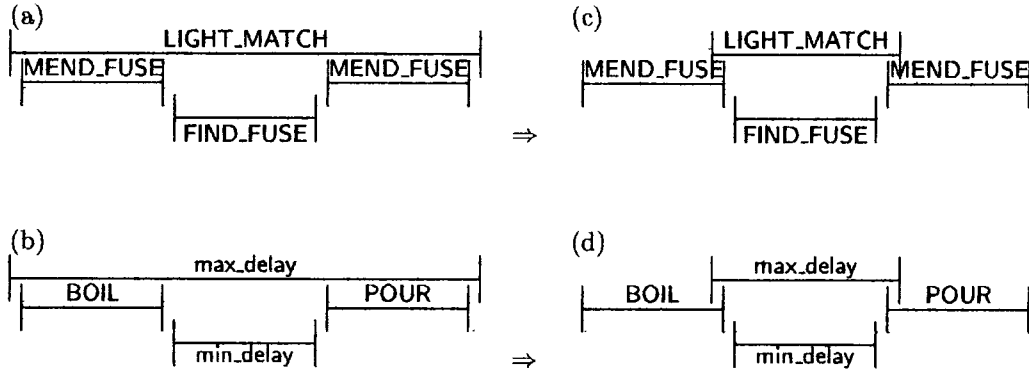


Figure 3.13: Comparison of the Match Domain and Minimum and Maximum Delays in PDDL2.1

Whilst the representations are syntactically equivalent, they are not always semantically equivalent. The actions that need translating, those which are the equivalent of the dummy maximum and minimum actions, may not actually be dummy actions and have other conditions and effects that are important to the domain. It is reasonable to assume that if a domain writer wished to explicitly express such a constraint, then they would make sure that the minimum value was less than the maximum. However, this assumption cannot be relied upon where the constraints arise naturally or in a disguised form. e.g. you cannot assume that $\text{FIND_FUSE}_{dur} \leq \text{LIGHT_MATCH}_{dur}$.

For these reasons, it would seem that in the general case you cannot always loosen the coupling between the planning and scheduling sub-problems through a simple translation of the domain.

3.4 Envelopes and Contents

Strong coupling between planning and scheduling occurs in co-ordination (Definition 3.5) where actions must happen concurrently. Envelopes and contents are sequences of actions that are logically constrained to be executed concurrently with one another.

Definition 3.8 — Envelope and Contents

An Envelope E and Contents C are both triples

$$E = (A_e, P_e, T_e)$$

$$C = (A_c, P_c, T_c)$$

where A_e and A_c are sets of action end points (either a start or end), P_e and P_c are sets of precedence constraints between those end points, and T_e and T_c are sets of temporal constraints relating to the duration of corresponding end point pairs in A_e and A_c respectively.

In the envelope:

$$\forall i \prec j \in P_e \cdot i \prec^{max} j$$

and in the contents:

$$\forall i \prec j \in P_c \cdot i \prec^{min} j$$

In co-ordination concurrent actions are logically, as well as temporally, constrained. One set of actions, called the “content” actions, must be executed whilst another set of actions, called “envelope” actions, executes. The contents must fit in the envelope, that is to say, the contents must start after the envelope has started and finish before the envelope finishes.

Definition 3.9 — End Points

The first end point in each is defined as

$$First_e = x \in A_e \cdot (\forall y \in A_e \cdot y \prec x \notin P_e \wedge x \neq y)$$

$$First_c = x \in A_c \cdot (\forall y \in A_c \cdot y \prec x \notin P_c \wedge x \neq y)$$

And the last end point in each is defined as

$$Last_e = x \in A_e \cdot (\forall y \in A_e \cdot x \prec y \notin P_e \wedge x \neq y)$$

$$Last_c = x \in A_c \cdot (\forall y \in A_c \cdot x \prec y \notin P_c \wedge x \neq y)$$

To ensure that they happen concurrently

$$First_e \prec First_c$$

$$Last_c \prec Last_e$$

For it to be schedulable it is necessary to know whether the minimum amount of time that the content actions can be executed in is less than the maximum amount of time that the envelope actions could take to execute. It stands to reason that if the envelope has an infinitely large maximum time or the content actions have a minimum time of zero, then there will be no problems scheduling since the content actions will always “fit in” the envelope. The problem occurs where the inverse is true. An envelope will have a finite maximum total execution time where *all* the temporal constraints between the actions are of a maximum type and the content will have a greater than zero minimum time where *all* the temporal constraints between the content actions are of a minimum type. This is regardless of whether these temporal constraints are explicitly encoded with dummy actions or whether they arise naturally with normal durative actions.

If there is even one precedence relationship in the content actions which is of the maximum type and not the minimum type (i.e. where $i_+ \prec j_+$), then the content actions can have a minimum time of zero and so definitely fit inside the envelope. Conversely, if there is even one precedence relationship in the envelope actions which is of the minimum type and not the maximum type (i.e. where $i_- \prec j_-$), then the envelope can be infinitely large and so can encompass any contents. This is the reason why:

$$\forall i \prec j \in P_e \cdot i \prec^{max} j$$

$$\forall i \prec j \in P_c \cdot i \prec^{min} j$$

Importantly, content actions can be envelope actions themselves (with other actions being the contents) and so similarly, envelope actions can also be content actions for other envelope actions. Content and envelope actions cannot be sequentialised with respect to one another and *must* be executed in parallel. In the case of the match domain, the `LIGHT.MATCH` action is the envelope action, and the `MEND.FUSE` actions are the content actions. See Figure 3.14 for examples of envelopes and content actions.

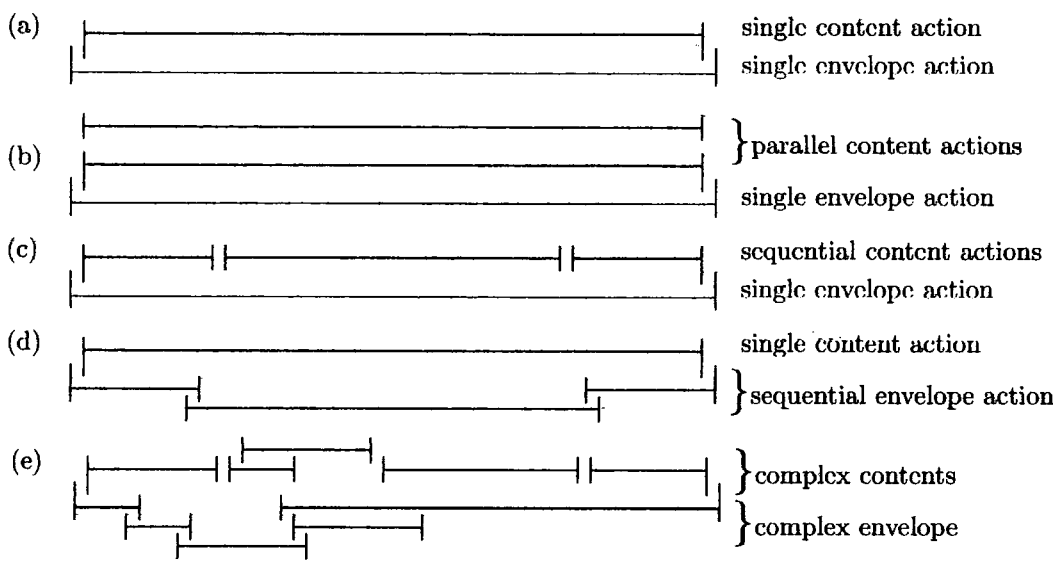


Figure 3.14: Envelopes and Contents

3.5 Detecting Single Potential Envelopes

The vast majority of action interactions in a domain are of the minimum precedence type; an end add effect of one action simply achieves a start condition of another, so must precede it. It is much more rare to find examples of envelopes (such as the `LIGHT_MATCH` action) involving start effects and end conditions (as already noted — none appear in benchmark domains), and so it is these envelopes that are focused on. In this chapter, only envelopes that are one durative action long (and so two instantaneous actions, one for the start and one for the end) will be looked at, but later in the next chapter, longer envelopes are investigated.

Definition 3.10 — Single Envelope _____

An envelope $E = (A_e, P_e, T_e)$ is a Single Envelope iff

$$|A_e| = 2$$

This definition means that in fact a clip *is* an envelope, as the two action extremity points contained in the contents come from different durative actions.

The following reasoning shows where these potential envelopes can occur. It should be noted that this only holds for the STRIPS and durative-action subsets of PDDL2.1; In particular, negative conditions are not permitted.

3.5.1 Reasons for Precedence

The precedence of the actions is forced through logical constraints. The Veloso algorithm from Figure 3.4 identifies three reasons why it may be necessary to order actions (summarised below in Figure 3.15).

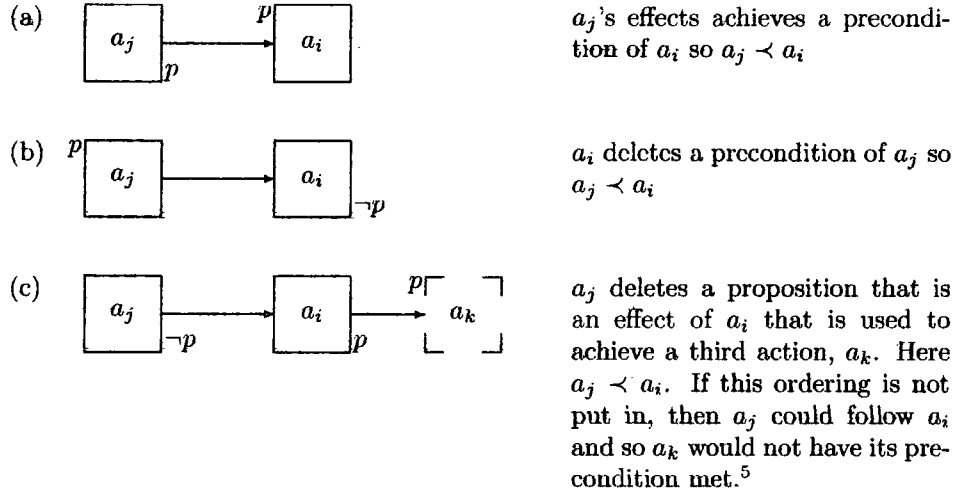


Figure 3.15: The Three Reasons to Order Actions

3.5.2 Defining Potential Envelopes

Presented here is a case analysis of where single envelopes could occur

For a single envelope and content action, two orderings are needed to ensure that the contents must fit inside the envelope: $First(E) \prec First(C) \wedge Last(C) \prec Last(E)$	2
There are three possible reasons to order two timepoints, as detailed in Figure 3.15.	3
There are then a possible 3^2 combinations which are shown in Table 3.1.	$3^2 = 9$
There are a further six possibilities if the precondition involved in the (a) start orderings from Figure 3.15 or in the (b) end orderings is instead an invariant.	$6 + 9 = 15$

⁵This is a standard declobbering technique used in partial order planners. Another is to order $a_k \prec a_j$, however, the Veloso algorithm does not allow for this as it can only remove orderings from the total order. i.e. if $a_n \prec a_{n+1}$ is in the total order, it is not possible to make $a_{n+1} \prec a_n$.

Each possibility is complicated further if the same proposition is used for both the start ordering and the end ordering (i.e. if $p = q$ in Table 3.1). This doubles the number of possibilities.	$15 \times 2 = 30$
The total number of possible envelopes is therefore	30

In fact any action with either a condition or effect (add or delete) at both the start and end of the action could be an envelope since all three of these propositions are involved in a potential ordering.

Compounded with these possibilities, it is assumed that the propositions involved in the envelope and content actions are not achieved by other actions. If they were, the following reasoning would be complicated even further. This is assumed since deciding whether a particular action (that could achieve this proposition) appears in a plan can be as hard as planning itself.

These single envelopes can be catagorised further. In some envelopes, such as the a-b pairing or the b-c pairing in Table 3.1, the content actions *could* appear outside (in these cases, after) the envelope action. Others cases (such as the a-a pairing), there is no possibility of this. These cases are referred to as “Hard Envelopes” and cases where potentially the content action could appear outside the envelope, are called “Soft Envelopes”.

Definition 3.11 — Hard Envelopes _____

In a Hard Envelope the content actions *must* go in the envelope such that

$$First_e \prec First_c \wedge Last_c \prec Last_e$$

Table 3.1: Nine Possible Combinations of Start End Pairs from the Three Ordering Reasons from the Veloso Algorithm

		End Orderings		
		a	b	c
Ordering Start	a	a-a 	a-b 	a-c
		b-a 	b-b 	b-c
		c-a 	c-b 	c-c
	b			
	c			

Definition 3.12 — Soft Envelopes

In a Soft Envelope the content actions *could* occur somewhere outside (either before or after) the envelope action such that

$$\begin{aligned} & First_e \prec First_c \wedge Last_c \prec Last_e \\ \vee & Last_c \prec First_e \\ \vee & Last_e \prec First_c \end{aligned}$$

With co-ordinated actions the actions must occur in parallel to produce the desired effect. However, with Hard Envelopes, the actions can only be executed concurrently, but with Soft Envelopes they could also be executed sequentially but with a different effect.

It is important to note that in the case of soft envelopes, the contents cannot simply “slip” out of the envelope. There must be an ordering between the end points of the envelope and content action. However, in the case of soft envelopes, there will be two similar states in the search space, one with the content inside the envelope and one with the content outside the envelope. In the case of hard envelopes, there will only be one state relevant in the search space, that where the content action is in the envelope.

Soft envelopes are distinguished between being “relevant soft” and “irrelevant soft”. In relevant soft envelopes (such as the b-c pairing), moving the content action outside the envelope does not result in any more true facts, so there would be no reason to do so, *unless* the content action did not fit in the envelope action. Conversely, in the case of irrelevant soft envelopes (such as the a-b pairing), keeping the content action inside the envelope results in fewer true facts, so there would be no reason to keep it in.

Figure 3.16 gives examples of these different envelopes. Figure 3.16(d) shows a possible envelope-content pair that is impossible (since the envelope action deletes its own invariant) so cannot be in the search space.

So, whilst there are 30 possible situations where one content action may be forced to be placed within one envelope action (assuming these are the only actions involving the propositions), some of these simply cannot arise with content actions (as in the case of deleting an invariant of the envelope action) and some do not compromise completeness (as in the case of irrelevant soft envelopes). Further still, some of these cases are obscure and are unlikely to arise in realistic domains. For these reasons, only one envelope will be analysed here: the case that occurs in the match domain.

Definition 3.13 — Single Hard Envelope

A Durative Action, *da*, is a Single Hard Envelope where:

$$add_{\vdash} \neq \emptyset \wedge del_{\vdash} \neq \emptyset$$

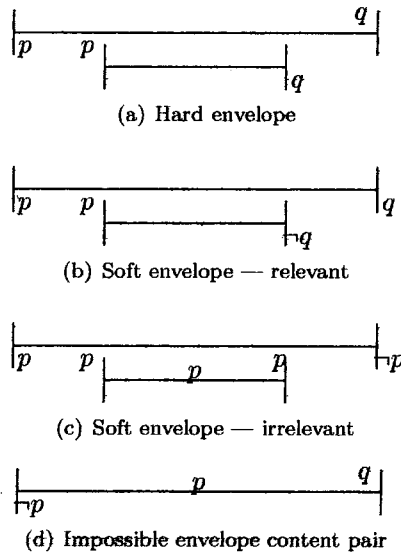


Figure 3.16: Potential Envelopes (with achieving contents)

There is a good reason to select this particular potential envelope. This is because it models a unary resource that is *only* available over a time window. It is common to want to model this. In the case of the match domain, the resource is light which is *only* available during the LIGHT_MATCH action. The handfree proposition also models a unary resource, however, the difference here is that this resource is always available, *except* during the MEND_FUSE action.

This potential envelope has another unique property. It is the only hard envelope (i.e. the contents cannot appear outside it) that is capable of existing on its own (i.e. the contents are not compulsory). This is proved below.

We shall name three states, s_1 , the state immediately before the start of the envelope action, s_2 , the state immediately after the start of the action, and s_3 the state immediately after the end of the action (see Figure 3.17). An action applicable in s_2 and not in s_1 must have been achieved by the start add effects (since there are no negative conditions, it could not have been achieved by a start delete effect). Taking it further, there are no actions that could be applied in s_2 and not in s_3 which could not have been applied in s_1 , apart from those achieved by the start add effects and then deleted by the end delete effects.

Any action conforming to this could be one of these envelopes, and so a simple domain analysis step can detect these in a problem.

The next chapter describes this and a temporal planner based on the LPGP/FF Hybrid system that can use this analysis to ensure that a valid plan is found, and so solve the match domain problem and other cases where co-ordination is present in the problem.

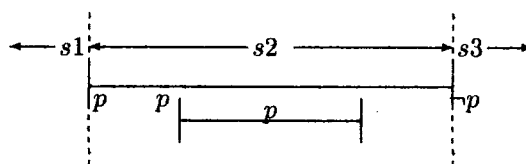


Figure 3.17: A Hard Envelope modelling a time limited resource

3.6 Chapter Summary

The FF/LPGP Hybrid splits planning and scheduling by decomposing the durative action into three separate instantaneous actions, planning with these, and then scheduling the resulting plan. This planner fails where the planning and scheduling sub-problems are tightly coupled on the spectrum of “tightness” (i.e. in the case of co-ordination).

Temporal constraints arise in PDDL2.1 problems through the arrangement of durative actions. Some arrangements lead to maximum precedence relationships, and others to minimum precedence relationships. An envelope is made of just maximum precedence relationships between the actions which will have a maximum execution time which cannot be exceeded, whilst a set of content actions will all be arranged with minimum precedence relationships, and so have a minimum execution time which cannot be reduced. Where these two sets of actions are logically constrained to happen concurrently, the execution time of the contents must be less than the execution time of the envelope. To simplify matters, envelope actions that are only one action long are examined, of which one case is singled out, the single hard envelope. This models a unary resource that is only available during the execution of the action.

Chapter 4

CRIKEY

This chapter describes a temporal planner named CRIKEY that splits the planning and scheduling components of temporal planning in a similar fashion to the LPGP/FF Hybrid (described at the beginning of Chapter 3). CRIKEY solves problems involving co-ordination where the hybrid system, and indeed all other planners, fails. In these cases the components are tightly coupled, which requires some communication between the planner and scheduler. CRIKEY minimises this communication using the theory presented in the previous chapter.

Whilst CRIKEY is based on the LPGP/FF Hybrid system, all components have been re-implemented in Java1.4 to form a complete unified system. Two versions of CRIKEY are described in this chapter. Figure 4.1 illustrates the differences between each of them including the LPGP/FF Hybrid. CRIKEY version 1 performs envelope analysis to detect single hard envelopes (Definition 3.13), whilst version 2 can reason with all envelopes (even those of many actions in length). Version 2 also performs more complex scheduling to handle duration inequalities.

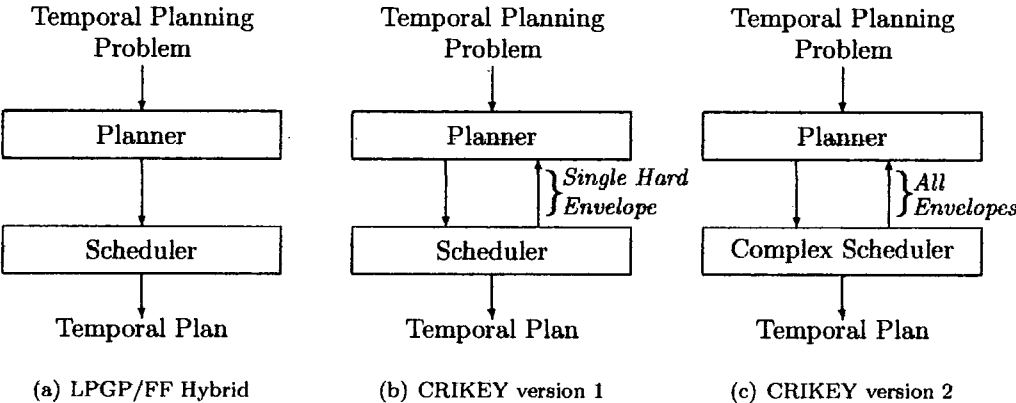


Figure 4.1: Differences Between the LPGP/FF Hybrid and the Two Versions of CRIKEY

4.1 Version 1

This version can only handle co-ordination where there are single hard envelopes and was the version used in the International Planning Competition 2004 (IPC'04). The architecture is outlined in Figure 4.2 and can be compared against a similar diagram for the LPGP/FF Hybrid in Figure 4.3.

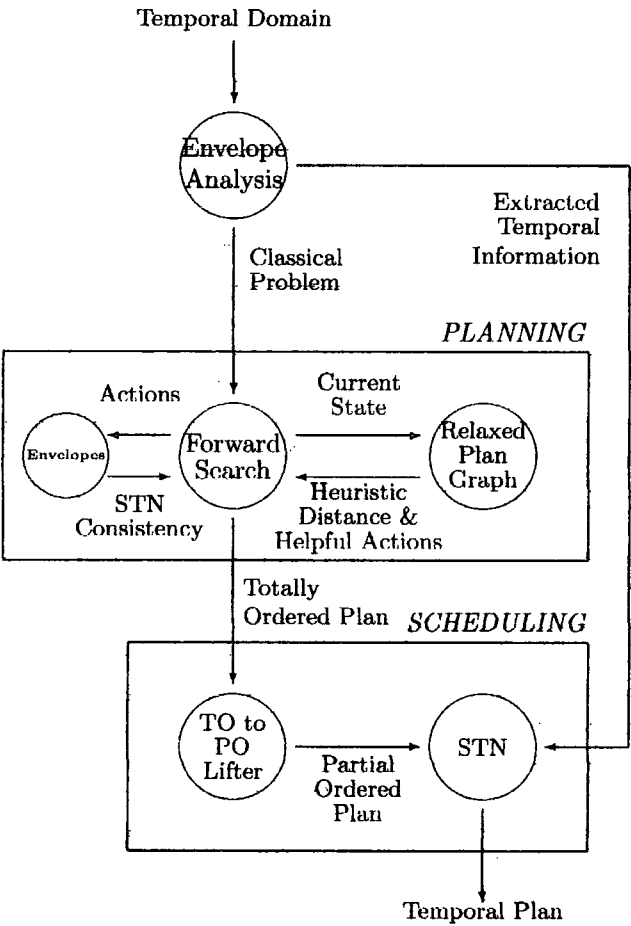


Figure 4.2: Architecture Overview of CRIKEY

In Figure 4.2, there is still no arrow back to the planner from the scheduler (as there is in Figure 4.1(b)). However, they are conceptually the same, since the envelopes in Figure 4.2 perform scheduling and it is here that the communication between the planner and scheduler takes place.

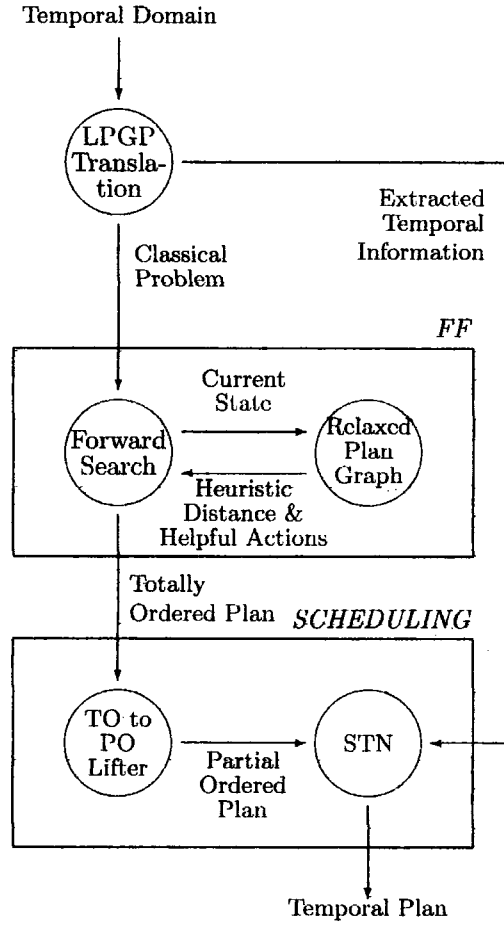


Figure 4.3: Alternative Architecture Overview of the LPGP/FF Hybrid

4.1.1 Envelope Analysis

After parsing the domain, durative actions (Definition 3.2) that are not single hard envelopes (Definition 3.13) are compressed into single, instantaneous STRIPS action (Definition 3.1)

Definition 4.1 — Compressed Action

A compressed action, $ca = (cond, add, del)$, is an STRIPS action that has been formed from a durative action, $da = (ta_cond, ta_add, ta_del, dur)$, where

$$\begin{aligned}
 cond &= ta_cond_+ \cup ((ta_cond_- \cup ta_cond_{\rightarrow}) \setminus ta_add_-) \\
 add &= (ta_add_+ \setminus ta_del_-) \cup ta_add_- \\
 del &= (ta_del_+ \setminus ta_add_-) \cup ta_del_-
 \end{aligned}$$

A compressed action has the effect of applying the whole action at once, i.e. applying the start effects first followed by the end effects, while still respecting the conditions. The preconditions to the compressed action are the start conditions of the durative action and all end conditions and invariants not achieved by the start effects. The add effects of the compressed action are the end add effects of the durative action and all start add effects that are not deleted by the end effects. Finally the delete effects of the compressed action are the end delete effects of the durative action and all start delete effects that are not re-achieved by the end add effects.

Single hard envelopes are split into two actions, one each for the start and end points, and not three as in LPGP translation (Definition 3.3) used in the LPGP/FF Hybrid. The rationale for this is that invariants are not dealt with correctly in the LPGP/FF Hybrid (as explained in Section 3.1) and so to rectify this, invariants are now handled separately, not requiring their own action. This is described below in Section 4.1.2.

Definition 4.2 — CRIKEY Action Translation

CRIKEY splits a single hard durative action $da = (ta_cond, ta_add, ta_del, dur)$ into two instantaneous STRIPS actions, da_+ and da_-
 $da_+ = (cond_+, add_+, del_+)$ is

$$\begin{aligned} cond_+ &= ta_cond_+ \cup (ta_cond_{\leftrightarrow} \setminus ta_add_-) \\ add_+ &= ta_add_+ \cup \{iAction_Name\} \\ del_+ &= ta_del_+ \cup \{gAction_Name\} \end{aligned}$$

$da_- = (cond_-, add_-, del_-)$ is

$$\begin{aligned} cond_- &= ta_cond_- \cup ta_cond_{\leftrightarrow} \cup \{iAction_Name\} \\ add_- &= ta_add_- \cup \{gAction_Name\} \\ del_- &= ta_del_- \cup \{iAction_Name\} \end{aligned}$$

As in the LPGP translation, a dummy proposition ($iAction_Name$) is used to ensure that no end action is placed in the plan without its corresponding start action. This proposition is an add effect of the start action, and a condition and delete effect of the end action. Once again the dummy proposition is unique to the split actions; Therefore, the only way that the precondition for the end action can be met is for start action to already be present in the plan. As with the LPGP translation, this is deleted by the end action so there is only one end action per start action. $iAction_Name_inv$, present in the LPGP translation, is no longer required since the durative action is now only split into two actions rather than three.

Another dummy proposition ($gAction_Name$) is added by the end action and deleted by the start action. The role of this is the converse of $iAction_Name$: that if a start action is

present in the plan, then so also is its corresponding end action¹. `gAction.Name` is added to both the initial and goal states. Selecting a start action deletes the goal `gAction.Name`, which can only then be re-achieved by selecting the corresponding end action.

To summarise, the envelope analysis stage will either compress a durative action into a single STRIPS action, or, if it is a single hard envelope, split it into two STRIPS actions: one each for the start and end of the envelope. This leaves only STRIPS actions in the problem.

4.1.2 Planning in Version 1

As in FF, searching is Enforced Hill Climbing (EHC) followed by Best First Search (BFS) should EHC fail to find a plan. The heuristic estimate is the length of a relaxed plan², extracted from a relaxed planning graph where the delete effects of actions are ignored. As proved in [45], this takes polynomial time to compute. In the same way as FF, helpful actions (actions in the relaxed plan that appear in the first layer of the relaxed planning graph) are used in EHC, but not in BFS.

States in the search contain open envelopes — split durative actions that have started but have not yet completed. Content actions that must go in these envelopes, are checked to ensure that they fit. This is formalised and described in detail in the rest of this section.

Definition 4.3 — Planning State _____

A planning state S is

$$S = (F, \xi)$$

where F is the set of true facts and ξ , the set of open envelopes.

Envelopes

ξ is the set of open envelopes. They are “open” in the sense that the start action has been selected (and so present in the plan), but not the end action.

Definition 4.4 — Open Envelope Version 1 _____

An open envelope \mathcal{E} in version 1 is

$$\mathcal{E} = (\mathcal{DA}, \mathcal{C}, \mathcal{TC})$$

where \mathcal{DA} is the single hard envelope durative action, \mathcal{C} is a list of content actions that must go inside the envelope, and \mathcal{TC} is the set of temporal constraints between the content actions and also the envelope action.

¹The LPGP translation does not guarantee this, as explained in Section 3.1.

²The relaxed plan may not be optimal.

An open envelope is effectively just a part of the plan containing co-ordination. It is partially ordered so that its consistency can be tested. Importantly, the consistency is only tested when there is an envelope action (i.e. when there is co-ordination) and only for the envelope and its contents (i.e. only for that part of the plan).

Definition 4.5 — Consistency Function

The function *consistent*(*nodes*, *edges*) returns the consistency of an STN, where *nodes* are the action end points in the network, and the *edges* are the temporal constraints.

Therefore, an envelope is consistent if

$$\text{consistent}(\{\mathcal{DA}\} \cup \mathcal{C}, \mathcal{TE})$$

A consistent envelope means that the contents “fit in” the envelope. Consistency is tested by performing Bellman-Ford’s Single Source Shortest Path algorithm from \mathcal{DA}_{\neg} (i.e. from the end of the envelope). Any negative cycles for this envelope must involve this end action as this will have a positive edge directed out of it for the maximum time difference from its start action, and then negative edges leading back to it for the minimum duration of the contents.

The Veloso function is used to decide whether an action becomes a content action of an envelope.

Definition 4.6 — Veloso Function

The Veloso function returns a set of temporal constraints *tc* between an action a_j and an open envelope $e = (\mathcal{DA}, \mathcal{C}, \mathcal{TE})$ with its contents.

$$tc = \text{vcloso}(a_j, e)$$

The function is defined as

- (a) if $\mathcal{DA}_{\neg} \prec a_j$ then $\{\mathcal{DA}_{\neg} \prec a_j\} \subseteq tc$
- (b) if $a_j \prec \mathcal{DA}_{\neg}$ then $\{a_j \prec \mathcal{DA}_{\neg}\} \subseteq tc$
- (c) $\forall a_i \in \mathcal{C}$ if $a_i \prec a_j$ then $\{a_i \prec a_j\} \subseteq tc$

One iteration of the Veloso algorithm (Figure 3.4) decides whether $a_i \prec a_j$.

Part (a) adds a constraint if a_j must follow the start of the envelope action, part (b) adds a constraint if a_j must precede the end of the envelope action, and part (c) adds a constraint if a_j must follow any of the content actions already in the envelope. ³

³As in the LPGP Hybrid, \preceq constraints are used where invariants are involved (see Section 3.1).

If the *veloso* function returns no constraints, then the action (a_j) is not a content action for the envelope (e).

$$veloso(a_j, e) = \emptyset$$

Invariants

Where durative actions are compressed into instantaneous STRIPS actions, their invariants cannot be broken, since the start and the end of the action are in effect applied one after the other, leaving no chance to break the invariants in between. However, where the durative action has been split there is a possibility that the invariant could be broken and then reacheived (as possible in the LPGP/FF Hybrid in Figure 3.5). To ensure this does not occur, an action, $a = (cond, add, del)$, must not delete any invariant of the open envelopes in state $s = (F, \xi)$.

$$\forall e \in \xi \cdot del \cap cond_{\leftarrow}(\mathcal{DA}(e)) = \emptyset$$

Applicability of Action

Definition 4.7 — Applicability

An action a is applicable in state s if

- (a) $cond \subseteq F$
- \wedge (b) $\forall e \in \xi \cdot del \cap cond_{\leftarrow}(\mathcal{DA}(e)) = \emptyset$
- \wedge (c) $\forall e \in \xi \cdot consistent(\{\mathcal{DA}(e), a\} \cup \mathcal{C}(e), \mathcal{IC}(e) \cup veloso(a, e))$

This states that (a) a 's preconditions must be met, (b) a must not delete any invariants that are currently protected, and (c) a must be consistent with all currently open envelopes in s . That is to say, if a must go in any of the currently open envelopes, then there is enough time to execute a and the other contents concurrently with the envelope action before the end of the envelope.

Application of Actions

Definition 4.8 — Update Envelope

$update(e, a)$, where an action a is placed in an open envelope, $e = (\mathcal{DA}, \mathcal{C}, \mathcal{IC})$, to produce e' is defined as:

$$\begin{aligned} e' &= e && \leftarrow veloso(a, e) = \emptyset \\ &= (\mathcal{DA}, \mathcal{C} \cup \{da(a)_{\vdash}, da(a)_{\dashv}\}, \\ &\quad \mathcal{IC} \cup veloso(a, e) \\ &\quad \cup \{da(a)_{dur} \leq da(a)_{\dashv} - da(a)_{\vdash} \leq da(a)_{dur}\}) && \leftarrow \text{otherwise} \end{aligned}$$

where $da(a)$ is the corresponding durative action for a

The $update(c, a)$ function places a content action in an open envelope if necessary, or if not, leaves the envelope unchanged. If it must go in, it updates the temporal constraints for precedence relationships between a and the rest of the contents and envelope action. A temporal constraint for the duration of a is appended to the set of temporal constraints. Note that regardless of whether a is a split action or a compressed action, the corresponding durative action is split (using Definition 4.2) and these two actions are placed as contents in the envelope. Therefore, the envelope only contains split actions.

Definition 4.9 — Result

The result, $Result(s, \langle a \rangle)$, of applying a single STRIPS action $a = (cond, add, del)$ in state $s = (F, \xi)$ is $s' = (F', \xi'')$ where

- (a) $F' = (F \cup add) \setminus del$
- (b) $\xi' = \xi \cup \{(da(a), \emptyset, \{da(a)_{dur} \leq da(a)_{\neg} - a \leq da(a)_{dur}\})\} \quad \leftarrow a = \neg$
- (c) $\quad = \xi \setminus \{e\} \cdot a = \mathcal{DA}_{\neg}(e) \quad \leftarrow a = \neg$
- (d) $\quad = \xi \quad \leftarrow \text{otherwise}$
- (e) $\xi'' = \{update(e, a) \mid e \in \xi'\}$

where $a = \neg$ denotes a is a start action, and $a = \neg$ denotes a is an end action.

Part (a) is the logical effects of the action a on s . It adds the add effects and then removes the delete effects of a from the set of true facts. Part (e) stipulates that where necessary, the action must be placed in the open envelopes to become a content action, using the $update$ function defined above. Part (b) adds a new open envelope to the state if the action is the start of a single hard envelope. If a is the end of a single hard envelope, then part (c) “closes” this envelope and removes it from the state. No additional content actions can now be placed in this envelope. If a is a compressed action, then no new open envelopes are either created or removed from the state (part (d)).

For completeness, a planning problem and its solution are defined.

Definition 4.10 — Planning Problem

A planning problem is

$$P = (O, I, G)$$

where O is a set of STRIPS actions (Definition 3.1), I is the initial state and G is the goal state.

Definition 4.11 — Goal State

A goal state $g = (F, \xi)$ must satisfy all the goal conditions, and the set of open actions must be empty (since the PDDL2.1 semantics require that all actions must complete).

$$F \subseteq G \wedge \xi = \emptyset$$

Definition 4.12 — Valid Plan

A solution to a planning problem is a plan p

$$p = \langle a_1, \dots, a_n \rangle$$

where $\langle a_1, \dots, a_n \rangle$ is an ordered list of actions. The result of applying a plan on a state s is defined recursively

$$Result(s, \langle a_1, \dots, a_n \rangle) = Result(Result(s, \langle a_1, \dots, a_{n-1} \rangle), \langle a_n \rangle)$$

A plan p is valid if a goal state is reached when each action is applied in sequence from the initial state.

$$Result(I, p)$$

Relaxed Plan

The relaxed plan is calculated in the standard way, using the compressed and split actions. The length of the relaxed plan gives the heuristic estimate of the distance from the current state to the goal.

Metrics

CRIKEY can handle metric variables as defined in PDDL2.1 by the fluents flag. Each state keeps a record of the current resource levels. These are changed by the operators in the effects of actions, and tested by conditional statements in the conditions.

The metric aspects have been omitted from the reasoning and definitions presented so far for simplicity and ease of understanding. There are two areas of note when considering metrics in CRIKEY. The first is in the compression and splitting of durative actions. Metrics involved in both the start effects and invariants of an action must be treated in a similar fashion to where invariants met by a start effect do not become conditions of the compressed

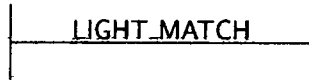
or start action (Definitions 4.1 & 4.2). For example, if an action has a start effect to increase a resource by 2 and an invariant requiring that the resource be less than 10, then the conditions of the compressed action or start action becomes that the resource should be less than 8.

The second area that metrics complicate is in the lifting of the partial order. Any precedence relationship in the total order between two actions that either test or change the same resource is kept in the partial order.

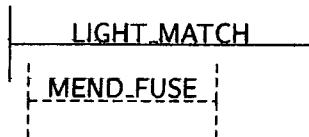
Metrics are incorporated into the heuristic in a similar fashion to MetricFF. At each fact layer of the relaxed planning graph, the maximum and minimum possible levels of each resource is calculated based on the values at the previous fact layer and the actions available in the previous action layer. For an action to be applicable in the relaxed planning graph, either the maximum or minimum level must meet the metric condition.

Version 1 and the Match Domain

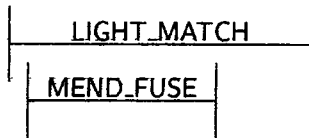
In the case of the match domain (as described in Section 3.2.1), assuming it is part of a bigger domain, CRIKEY will search forward ignoring temporal information.



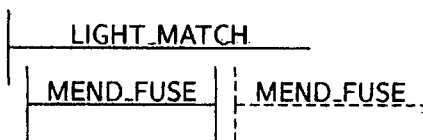
When it comes to put in the start action to the LIGHT_MATCH action (a single hard envelope), it will create a new open envelope.



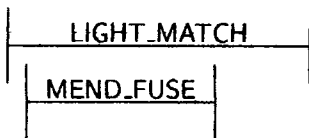
It will then test to see if a MEND_FUSE action need go in this envelope, and if so, if it is consistent..



Indeed, it fits, so the action is applicable and selected for the plan.



It will then test the second MEND_FUSE action. This is not consistent with the envelope (there is not enough time left to fix it before the match burns out), so cannot be inserted in the plan. (If the fuses could be fixed in parallel, then this second action would be consistent).



The end of the light action could then be selected and the envelope closed. CRIKEY would then proceed to either light a second match (and so start a new envelope) or solve another part of the problem. In this way a schedulable plan is produced.

4.1.3 Scheduling in Version 1

All the compressed actions in the total order plan are split into a start and end action as in Definition 4.2. Scheduling is then identical to the scheduling in the LPGP/FF Hybrid system. A partial order is lifted from the total order plan by an implementation of the Veloso algorithm and is translated into temporal constraints. As with the LPGP/FF Hybrid, these constraints are put into a 2-D matrix, representing the graph of the STN and the shortest distance is found between the start actions and X_0 , calculated by Floyd-Warshall's algorithm. Once again, the temporal plan is then output as a list of time stamped actions with their durations.

Although the same scheduling process is followed in version 1 as in the LPGP/FF Hybrid, unschedulable plans cannot be produced because the planner has already checked at the critical points (where there is co-ordination) that a schedule can be found through the detection of envelopes.

A Note on the Implementation The formalisation of this first version is closely linked to the implementation of the planner. In particular, CRIKEY has an envelope class that contains small STNs, which are discarded once the envelope is closed, and then has a totally separate scheduling phase. This has the disadvantage of not showing clearly exactly how the planner and scheduler communicate. To address this, an alternative formalisation is given in Appendix D where the scheduling is integrated into the planning phase, and a partial order built rather than a total order. One STN (representing the whole plan) is kept for the state, rather than many smaller STNs (in the case of the envelopes). However, once again, the partial order is only checked for consistency when and where absolutely necessary (that is, where the actions are involved in co-ordination). *This makes it conceptually exactly the same as the above formalisation.* The disadvantage of implementing the planner in this way is that the STN must be duplicated for each state in the search. This would be expensive, both in terms of memory and CPU time. In the current implementation, only the small envelope STNs are duplicated and then discarded once the envelope is closed.

4.2 Characteristics of Version 1

This new planner has a number of advantages over the LPGP/FF Hybrid system. By compressing actions where there is no co-ordination, the search space becomes smaller. States where actions have been started but not completed are no longer in the search space: intuitively a good idea since any action must complete at some point. In searching, it effectively skips through this intermediate state and applies both the start action and end action at once. Compressed actions are effectively blackbox actions where the state of the world is not known while the action is being executed. However, CRIKEY only compresses these actions where it is safe to do so (i.e. *not* in the case of single hard envelopes).

Splitting the action into only a start and end, and not also an invariant action (as the LPGP/FF Hybrid does) reduces the search space by a third. This applies to both the planning search space and also the relaxed plan graph. This is only a polynomial reduction in size, but will have an impact on performance in practice.

The main advantage (and purpose) of this planner is that it can handle domains with co-ordination where the LPGP/FF Hybrid cannot. To do this the search must have access to the internal state of an action (by splitting the durative actions into two). Whilst the hybrid does this, it cannot guarantee that the resulting plan is schedulable, since it does not test the consistency of the schedule until the plan is fully built. CRIKEY, however, detects single hard envelopes in advance and so can realise where the internal state of an action needs to be known and also when and where to test this consistency. In the benchmark domains, where there is no co-ordination, the consistency will never be tested, but CRIKEY minimises the consistency testing in domains where there is co-ordination. It will only test for consistency where there are envelopes (and so co-ordination), and will only check the consistency on that part of the plan which needs to be checked, that is, only on the part of the plan containing the envelope and contents.

There is one major disadvantage of this version of CRIKEY. As discussed in the previous chapter, there are many places where envelopes could occur, and so many places where it could, potentially, be necessary to test for consistency. However, CRIKEY, in the form described above, only detects one such instance. Albeit a common instance that it recognises, it does not preclude the fact that this makes the temporal planner incomplete. Furthermore, this version is unable to handle envelopes that are more than one action long.

One possible way to extend CRIKEY would be for it to perform further envelope analysis to detect other (possibly common) potential envelopes. This would still not make CRIKEY complete, unless all envelopes were found and this (as discussed previously) would mean seeing every action as a potential envelope.

The rest of this chapter looks at an extension of CRIKEY which splits all actions, but detects envelopes “on the fly” during planning. This enables it still to minimise consistency checking, once again performing it only when and where necessary.

Another weakness of this architecture is that it is hard to find good quality plans since the metric (and specifically the temporal information, for minimising the total execution time of a plan) is ignored during planning. The second version takes steps to remedy this.

4.3 Version 2

This second version of CRIKEY performs no envelope analysis to find envelopes in advance. Instead it splits all durative actions into two actions as in Definition 4.2. This increases the size of the search space compared to version 1. Since incompleteness occurs where states do not appear in the search space, this increase in size is inevitable. The main difference between version 1 and version 2 is the ability to handle all envelopes, even those which are multiple actions long.

4.3.1 Envelope Management

The form that envelopes take is different in this version.

Definition 4.13 — Open Envelope Version 2

An open envelope, e , in version 2 is

$$e = (\mathcal{S}, \mathcal{E}, \mathcal{F}, \mathcal{P}, \mathcal{TC})$$

where \mathcal{S} and \mathcal{E} are the start and end actions of the envelope respectively. \mathcal{F} is the list of content actions that must follow the start of the envelope, and \mathcal{P} is the list of content actions that must precede the end of the envelope. \mathcal{TC} is once again the set of temporal constraints both between the envelope actions and also the content actions.

Open envelopes in version 2 allow for envelopes that are many actions long, and not just single hard envelopes. \mathcal{S} and \mathcal{E} need not belong to the same durative action.

The consistency function remains the same as in the first version, so to test the consistency of an open envelope e , it is now:

$$\text{consistent}(\{\mathcal{S}, \mathcal{E}\} \cup \mathcal{F} \cup \mathcal{P}, \mathcal{TC})$$

The two lists of actions (those that must follow the start and those that must precede the end) keep the transitive closure for these end actions. If the intersection of these two sets is not empty (i.e. $\mathcal{F} \cap \mathcal{P} \neq \emptyset$), then the consistency of the envelope must be checked, again using Ford-Bellman's SSSP algorithm from the end (\mathcal{E}) of the envelope. If the intersection is empty, then there is no need to check the consistency, as the contents have a minimum time of zero and the envelope will definitely be consistent.

The Veloso function must also change as open envelopes are different in this version.

Definition 4.14 — Veloso Function

The veloso function returns a set of temporal constraints tc between an action a_j and an open envelope $e = (\mathcal{S}, \mathcal{E}, \mathcal{F}, \mathcal{P}, \mathcal{TC})$ with its contents.

$$tc = \text{veloso}(a_j, e)$$

The function is defined as

- (a) if $\mathcal{S} \prec a_j$ then $\{\mathcal{S} \prec a_j\} \subseteq tc$
- (b) $\forall a_i \in \mathcal{F}$ if $a_i \prec a_j$ then $\{a_i \prec a_j\} \subseteq tc$
- (c) if $a_j \prec \mathcal{E}$ then $\{a_j \prec \mathcal{E}\} \subseteq tc$
- (d) $\forall a_i \in \mathcal{P}$ if $a_j \prec a_i$ then $\{a_j \prec a_i\} \subseteq tc$

One iteration of the Veloso algorithm (Figure 3.4) decides whether $a_i \prec a_j$.

Applicability of Action

Definition 4.7 part (c), the applicability of an action, changes to reflect the fact that the *consistent* function is now used differently with the new envelopes.

Definition 4.15 — Applicability

An action $a = (cond, add, del)$ is applicable in state $s = (F, \xi)$ if

- (a) $cond \subset F$
 - (b) $\wedge \forall e \in \xi \cdot del \cap cond_{\mapsto}(da(\mathcal{E}(e))) = \emptyset$
 - (c) $\wedge \forall e \in \xi \cdot consistent(\{\mathcal{S}(e), \mathcal{E}(e), a\} \cup \mathcal{F}(e) \cup \mathcal{P}(e),$
 $\mathcal{TC} \cup veloci(a, e))$
-

Application of Actions

Definition 4.8 must be revised.

Definition 4.16 — Update Envelope

$update(e, a)$, where an open envelope, $e = (\mathcal{S}, \mathcal{E}, \mathcal{F}, \mathcal{P}, \mathcal{TC})$, has an action a placed in it to produce $e' = (\mathcal{S}', \mathcal{E}', \mathcal{F}', \mathcal{P}', \mathcal{TC}')$ is defined as

$$\begin{aligned}
 \mathcal{S}' &= \mathcal{S} \\
 \mathcal{E}' &= \mathcal{E} \\
 \mathcal{F}' &= \mathcal{F} \cup \{a\} && \leftarrow \exists a_i \in \mathcal{F} \cup \{\mathcal{S}\} \cdot a_i \prec a \\
 &= \mathcal{F} && \leftarrow \text{otherwise} \\
 \mathcal{P}' &= \mathcal{P} \cup \{a\} && \leftarrow \exists a_j \in \mathcal{P} \cup \{\mathcal{E}\} \cdot a \prec a_j \\
 &= \mathcal{P} && \leftarrow \text{otherwise} \\
 \mathcal{TC}' &= \mathcal{TC} \cup veloci(a, e) \\
 &\quad \cup \{da(a)_{dur} \leq da(a)_{\neg} - da(a)_{\neg} \leq da(a)_{dur}\}
 \end{aligned}$$

This has the effect of adding the action a to the list of followers \mathcal{F} , if it must follow either the start of the envelope or any other action in the list of followers. Additionally, it is added to the list of precursors \mathcal{P} should it precede any other action in that list or the end of the

envelope. A temporal constraint for the duration of the action is added to \mathcal{TE} in addition to the constraints returned by the *Veloso* function.

Another function is needed to create new envelopes, multiple actions long, by expanding other open envelopes through the addition of new envelope actions.

Definition 4.17 — Expand Envelope

$expenv(e, a)$, where an open envelope, $e = (\mathcal{S}, \mathcal{E}, \mathcal{F}, \mathcal{P}, \mathcal{TE})$ has an action a placed in it, to produce e' is defined as:

$$\begin{aligned} e' &= (\mathcal{S}, da(a)_{\neg}, \mathcal{F} \cup \mathcal{P}, \emptyset, \\ &\quad \{da(a)_{dur} \leq da(a)_{\neg} - a \leq da(a)_{dur}, \mathcal{E} - a < \varepsilon\} \cup \mathcal{TE}) \\ &\quad \leftarrow \exists a_j \in \{\mathcal{E}\} \cup \mathcal{F} \cdot a \prec^{max} a_j \\ &= e \quad \quad \quad \leftarrow \text{otherwise} \end{aligned}$$

If an action forms a maximum precedence relationship (Definition 3.6) with either the end of a currently existing envelope, or any action that precedes it, then a new envelope is created which is the combination of the original envelope and the new action. The new envelope is a copy of the original envelope, however the envelope's end action is set to the end of the new action, and the actions that preceded the end now follow the start. A temporal constraint is also added for the duration of this new action, and also to specify that the two (or more) envelope actions are of the maximum precedence type.

Definition 4.18 — Result

The result, $Result(s, \langle a \rangle)$, of applying a single STRIPS action $a = (cond, add, del)$ in state $s = (F, \xi)$ is $s' = (F', \xi''')$ where

- (a) $F' = (F \cup add) \setminus del$
 - (b) $\xi' = \xi \cup \{expenv(e, a) \mid e \in \xi'\}$
 - (c) $\xi'' = \xi' \cup \{(a, da(a)_{\neg}, \emptyset, \emptyset, \{da(a)_{dur} \leq da(a)_{\neg} - a \leq da(a)_{dur}\})\} \quad \leftarrow a = \vdash$
 - (d) $\quad = \xi' \setminus \{e\} \cdot a = \mathcal{DA}_{\neg}(e) \quad \leftarrow a = \neg$
 - (e) $\xi''' = \{update(e, a) \mid e \in \xi''\}$
-

New envelopes are created in a state in one of two ways. In the first case (c), a new start action is chosen. This is the same as for single envelopes as described in version 1 where the start and end actions in the envelope correspond to the start and end actions of the durative action. The two sets of actions that precede and follow the extremes of the envelope are

initially empty. The temporal constraint set contains only one constraint corresponding to the duration of the envelope action. Alternatively, new envelopes can be created where envelopes are multiple actions long (d). Again, open envelopes are removed from a state (closed) when the end action to an envelope is chosen (d). Part (e) places content actions in open envelopes if necessary.

In cases where there is no co-ordination (and so no envelopes), as in the traditional benchmark domains, envelopes are created when the start action is chosen. If the end action is not immediately chosen next, then an action may have to follow the start of the envelope (say, if it has a start effect) or precede the end of the envelope (say, if there is a condition to meet). However the intersection of the two sets will remain empty and consistency checking will not be performed.

To summarise, in this version CRIKEY again only communicates with the scheduler where absolutely necessary and only on that part of the plan where there is danger of producing an unschedulable plan. This version, however, can deal with all types of envelope including those which are many actions in length. If, when putting a content action in the envelope, there is a maximum precedence relationship, then a new envelope (many actions long) is created.

4.3.2 Scheduling

Scheduling in the second version differs from the first. As before, the Veloso algorithm lifts a partial order from the total order plan, however the resource reasoning is performed with precedence graphs. As this is not strictly in the scope of this thesis and not a novel technology, but rather an new application of it, it is not presented in detail here. Precedence graphs are summarised below and described in full in [50]. The rest of this section describes how they are integrated into CRIKEY including the changes to [50] that had to be made, followed by an example of how they operate.

Precedence Graphs

Most resource scheduling approaches reason with the actual timing bounds of actions. However, Precedence Graphs look at their relative positions. Each resource in the plan has its own graph, where the nodes are action end points that contain either a condition relating to that resource, or a resource operator in the effect. Each node is labelled with the minimum and maximum production or consumption of the resource at that node. Edges between the nodes are precedence orderings. These graphs need not be represented explicitly but can be deduced from the STN that holds this information.

The “balance constraint” is calculated for each node in each graph⁴. The basic idea of

⁴For reservoir resources (as PDDL2.1 fluent variables are), the balance constraint requires the resource to be closed, i.e. there are no more nodes to be added to the graph. This is the case in CRIKEY, since the resource reasoning is performed after the planning is complete.

the balance constraint is to compute a lower and upper bound on the resource level just before and just after each event (i.e. $x \pm \varepsilon$). To calculate an upper bound, all maximum production levels of all events that *could* happen before the event are summed with the minimum consumption levels of all events that *must* happen before the event. In a similar way the other balance constraints are calculated.

In fact, precedence graphs as described in [50] use a slightly different model of resources to PDDL2.1. In that model, all resources have a maximum possible level and a minimum possible level that is always zero. PDDL2.1 does not explicitly model resources, and does not have maximum and minimum possible levels encoded in. Instead, the resources must meet conditions which can change from action to action. This has the effect of changing the minimum and maximum possible levels of the resource throughout the plan.

For example, the model used in [50] would specify a fuel tank to have a minimum level of zero and some constant maximum capacity. In PDDL2.1, this maximum capacity can change during the plan, as can the minimum.

For this reason, some simple changes are made to the reasoning presented [50]. Instead of calculating balance constraints at every node in the graph, it only calculates them for those nodes that contain conditions. The maximum and minimum levels must then meet these conditions, (and not, as in the model in [50], keep the maximum and minimum between zero and the maximum level). Secondly, when calculating the minimum and maximum values, it only considers nodes that contain resource operators.

The balance constraints can then be used to discover:

- dead ends
- new precedence relations
- new bounds on resource usage
- new bounds on time variables

Dead ends (where the conditions cannot be met) are not found in CRIKEY, since it keeps track of metric values during the planning phase to ensure that there is always adequate resource. Resource reasoning is not separated out (unlike the temporal reasoning) so there is no chance of finding an un-schedulable plan due to lack of resources. In the worst case, the precedence graphs will order all the actions identically to the total order plan produced. However, it will find concurrency where possible.

CRIKEY does discover new precedence relations. For each condition, it is made sure that either the maximum and minimum resource levels must meet the condition and if not, precedence relations are put in to ensure that the condition is met (by ordering producers or consumers to occur before the condition).

CRIKEY can use the balance constraints to find new bounds on both the time variables (which can be propagated through to the STN) and resource usage variables. This only

occurs where there are duration inequalities in the domain, as this is the only case where operators in the plan can produce or consume variable amounts of resource with actions of variable duration.

An example precedence graph is given in Figure 4.4(a) for the fuel level of a car. There are two move actions, both of which consume between 10 units of fuel. There is also a refuel action (not presently ordered with respect to the move actions) that can produce between 0 and 20 units of fuel (depending on the length of the action).

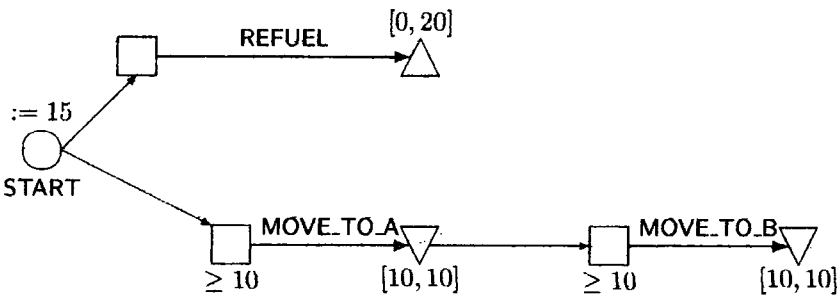
Firstly, in Figure 4.4(b), the precedence graph is able to reason that the REFUEL action must happen before the second MOVE.TO.B action and so the appropriate precedence relationship is added. This in turn allows reasoning for the resource bounds of the REFUEL action, as it must now produce a minimum of 5 units. The refuel action must now be of sufficient length to supply the 5 units, and this information can be propagated up to the STN.

Duration Inequalities

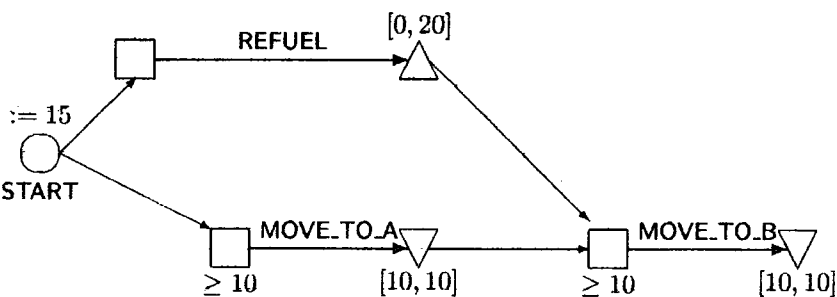
PDDL2.1 allows the specification of duration inequalities. Rather than fixing the duration of a durative action, these allow bounds to be put on the duration. These bounds can be a function of other metric values (for example, you cannot drive for longer than the amount of fuel available). However, resource change can also be dependent on the duration of an action (for example, the longer you heat water for, the hotter it becomes). The duration of an action now effectively becomes a hidden parameter of the action. This allows resource change to be decided by the planner. For example, it is possible to decide how long to fill the tank up for (the duration of the refuel action) and so therefore how full the tank is at the end of the action. The possible combinations are summed up in Table 4.1.

The (c) and (f) cases then present resource scheduling problems where it would intuitively seem illogical to decide exactly how long an action should be and exactly how much resource should be produced or consumed until after the plan is produced (i.e. the problems should be separated out). This version of CRIKEY provides the ideal architecture for this since both the STN and the precedence graphs handle upper and lower bounds on both resource production and consumption and also on time. Through these, contents can be made to fit exactly in envelopes, and resources can be maximised and minimised. For example, in the match domain, if the duration of the match is set to `:duration (<= ?duration 8)` it would be possible to “blow out” the match once the fuse is fixed.

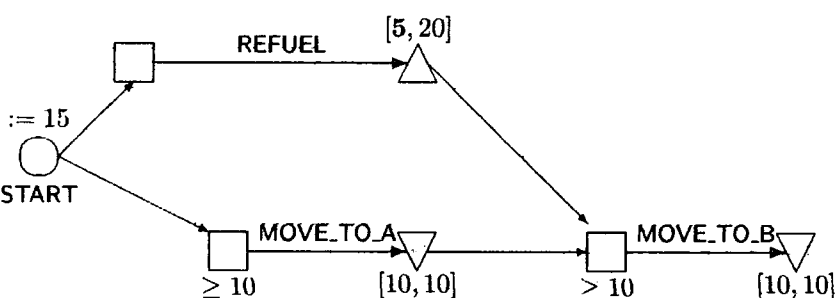
CRIKEY reads the quality metric in the PDDL2.1 problem file to decide what to maximise or minimise in the precedence graphs. This could be a resource or the total time. If it is a resource that is to be maximised, then that precedence graph is selected and the producers maximised and the consumers minimised (by changing the duration of their corresponding actions). If it is to be minimised, then the converse happens. After calculating this, CRIKEY propagates the results through to the STN and the other precedence graphs.



(a) Precedence Graph for the Fuel Level of a Car



(b) A Precedence Relationship is Added



(c) The Resource Bounds change

KEY:

○

□

△

▽

Initial Resource Level

Action End Point (with Condition)

Action End Point with Increase in Resource Level

Action End Point with Decrease in Resource Level

[min,max]

min and max Resource Change

→

Precedence Relationship

Figure 4.4: Example Precedence Graph

Table 4.1: Possible Specifications of Durations and Resource Conditions and Operators

Specification	Example	Notes
Durations		
(a) Fixed	<code>(= ?duration 5)</code>	The duration of the action is always known and does not change.
(b) Function	<code>(= ?duration (fuel ?t))</code>	The duration of the action will depend on the state.
(c) Condition	<code>(≤ ?duration (fuel ?t))</code>	The duration is a choice of the planner.
Resource Conditions and Operators		
(d) Fixed	<code>(≥ (fuel ?t) 0)</code> <code>(increase (fuel ?t) 3)</code>	The value of the operator or condition is always known and does not change.
(e) Function	<code>(≥ (fuel ?t) (fuel_required ?t))</code> <code>(decrease (fuel ?t) (fuel_used ?t))</code>	The value of the operator or condition is dependent on the state.
(f) Function of Duration	<code>(increase (fuel ?t) (* (refuel_rate) ?duration))</code>	The resource change is dependent on the duration.
Combinations		
(f) & (b)		equivalent to (e)
(f) & (c)		The resource change is a choice of the planner

If it is the total-time to be minimised, then the duration of each durative action is set to its minimum. The default behaviour is to minimise the total-time and the resource levels.

An example of this is the Café Domain (see Appendix E) where the object is to deliver breakfast to a table in a café, as drawn diagrammatically in Figure 4.5⁵. However, due to there only being one electrical socket in the kitchen, the toast and the tea cannot be made simultaneously. Once either is made, it starts to cool, until delivered to the table. Whilst it is preferable to have them as hot as possible when delivered, it is also preferable to deliver them at the same time (or as close to each other as possible). There are three possible metrics, one is to minimise the heat lost by each item whilst it is in the kitchen, another is to have them delivered as close as possible together (i.e. minimising the delivery window), and finally simply to minimise the total-time of the whole plan.

For each metric the same partial order plan is lifted, with the same bounds on both the resource levels and the action times. However, if the first metric is chosen, then the `LOSING_HEAT` actions are minimised. This has the effect of delivering the tea and toast as soon as they are made. This is propagated through to the precedence graph with the

⁵This domain contains maximum orderings (the `LOSING_HEAT` and `DELIVERY_WINDOW` actions) and so also co-ordination.

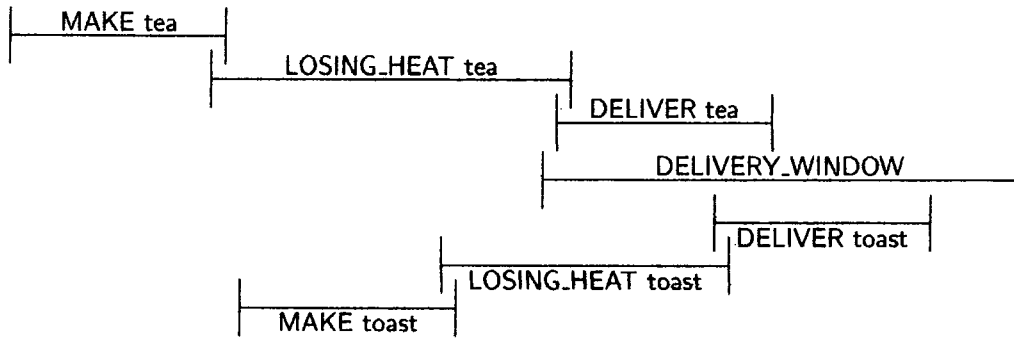


Figure 4.5: A Partial Order for the Café Domain

DELIVERY_WINDOW, which will mean this can no longer be as short as it could have been. Then, by default the DELIVERY_WINDOW is minimised and then the total-time. If the second metric is chosen, first the DELIVERY_WINDOW action is minimised (resulting in the tea waiting and cooling whilst the toast is prepared) and then the LOSING_HEAT actions are minimised. Finally, if the total time is to be minimised, the precedence graphs are ignored, the actions' duration minimised, and then the earliest start times chosen for each action. Figure 4.6 shows two plans. One where the heat lost is minimised, and one where the delivery window is minimised.

(:metric minimize (total_delivery_window)) (:metric minimize (total_heat_lost))	
0.01: (MAKE_TEA tea1 socket1) [1.00]	0.01: (MAKE_TEA tea1 socket1) [1.00]
1.00: (LOSING_HEAT tea1) [2.04]	1.00: (LOSING_HEAT tea1) [0.03]
1.02: (MAKE_TOAST toast1 socket1) [2.00]	1.01: (DELIVERY_WINDOW table1) [4.03]
3.01: (LOSING_HEAT toast1) [0.03]	1.02: (DELIVER tea1 table1) [2.00]
3.02: (DELIVERY_WINDOW table1) [2.02]	1.02: (MAKE_TOAST toast1 socket1) [2.00]
3.03: (DELIVER tea1 table1) [2.00]	3.01: (LOSING_HEAT toast1) [0.03]
3.03: (DELIVER toast1 table1) [2.00]	3.03: (DELIVER toast1 table1) [2.00]
Total Delivery-Window: 2.02	Total Delivery-Window: 4.03
Total Heat-Lost: 2.07	Total Heat-Lost: 0.06

Figure 4.6: Two Plans with Identical Goals but Different Metrics

Some assumptions were made in the implementation of the precedence graphs that limit what can be expressed in the problem. Firstly a resource operator's change cannot be a function of another resource that is also a function of an action's duration. This means that once a change has been made in a precedence graph (i.e. a new resource bound found or a new limit on the duration of an action), this will propagate only up to the STN, and will not affect any other resource changes in other precedence graphs. There is no reason why CRIKEY cannot be extended to relax this assumption, meaning that the propagation must

happen also between precedence graphs, but this is not in the scope of this thesis. Secondly, resource change that is a function of the duration, cannot be a binary function of the duration. Once again, there is no reason why this cannot be relaxed, but has been kept for ease of implementation. Finally, the metrics in PDDL2.1 allow functions of resources to be optimised, but this implementation only allows for a single resource to be optimised. Once more, there is no reason for this apart from ease of implementation. These assumptions can all be relaxed to allow for the full expressive power of PDDL2.1 with no additional complexity.

4.4 Comparison with Sapa

Similarities and differences can be observed between Sapa (as described in Section 2.5.2) and CRIKEY. They are both able to plan with problems that contain concurrency, and in particular, co-ordination⁶. Both perform forward chaining state space search using a relaxed plan as an heuristic, with both having a similar notion of state. They both take a histories view of change since they both keep a record of the past (i.e. the current plan) and both keep a record of propositions that are currently true, the values of the metric resources and the invariants that must not be broken in the current state. But it is in the view of the future that they differ. Sapa associates a time with each state. This is not the case in CRIKEY's states, since the actual times are scheduled during a separate scheduling phase. Secondly, whereas Sapa keeps a list of time stamped updates scheduled to happen at a particular point in the future, CRIKEY keeps a set of updates that will happen *at some undetermined point* in the future.

The consequence of these two differences is that Sapa does not separate the scheduling from the planning whilst CRIKEY does. CRIKEY orders its actions and puts times on them only after the actions have been chosen whereas Sapa does this simultaneously with choosing the actions. If there are two non-interfering actions, and in one state they are occurring in parallel and in another, sequentially, Sapa will consider these to be two different states, whereas CRIKEY will consider them to be the same state, and make this decision during the scheduling phase. This means an increase in the state space for Sapa, which in turn could mean more to backtrack over (i.e. not just the planning decision made, but also the scheduling decisions as well).

Key is Sapa's "advance-time" action that changes the state to the next update in the queue. Sapa discourages its use by not re-calculating the heuristic after using it, and in doing so favours concurrency in its plans. If it did not do this, then it would always be advantageous (in terms of the heuristic value) to move the state onto the next timepoint. As the states in CRIKEY are not time stamped, there is no need for the "advance-time" action. The choosing of an end action is CRIKEY's equivalent. It takes an update which is

⁶In fact, Sapa contains a bug that results in invalid plans being produced for domains with co-ordination. This is discussed in the next chapter.

known to happen in the future and advances the state to that point. Only counting the start actions in a relaxed plan to form the heuristic has the same effect as Sapa not recalculating the heuristic. This is not necessary though, as CRIKEY puts in the concurrency after planning.

Since scheduling happens during the search for a plan with Sapa, envelopes are handled in the search. Alternatively put, there is no need to check the schedulability of the state, because the schedule *is part* of the state.

An advantage of CRIKEY's states is where there are duration inequalities with resource operators and constraints dependent on the duration of the action, as discussed in the previous section. These effectively allow the parameters of an action to take numeric values. Unlike Sapa⁷, which would be forced to decide on a duration there and then (and so also on the resource levels), CRIKEY need not commit at this point. Sapa could then have to backtrack to change this decision. CRIKEY, through the use of an STN and precedence graphs, can keep the plan unconstrained in this respect. The usual pitfall of this approach is that the planner must make sure that the STN is consistent through communication with the scheduler. This communication is minimised by only checking when and where it is necessary, through the detection of envelopes.

Sapa has the advantage of being able to know the quality of the plan during the search since it calculates the schedule as it plans. This can be used to guide the search to better quality plans. As CRIKEY ignores all temporal information during planning, it is unable to do this, potentially leading to inferior plan quality.

4.5 Chapter Summary

Two versions of CRIKEY were built and formalised here. Both use the theory presented in the previous chapter to minimise the communication between the planner and scheduler (or by an alternatively view; minimise the search space). The first version only handled one type of single envelope, the second any envelope, including those which are multiple actions long. This second version results in a larger search space but this is necessary since the inability to handle them in the first version was due to missing states. The second version is also able to handle duration inequalities as it does not specify the future timings of known actions.

⁷In fact, Sapa is unable to handle duration inequalities, but it could be extended to use this feature.

Chapter 5

Results

This chapter presents and analyses empirical results from testing both versions of CRIKEY on a variety of domains and comparing them with results from other temporal planners. Firstly, the capabilities of the planners are listed and compared. This is followed by results from the 4th International Planning Competition (IPC'04) in which the first version of CRIKEY competed. These domains contain no co-ordination, which CRIKEY is specifically designed for, so this is tested through some new domains. Finally, the second version of CRIKEY's ability to use the plan quality metric provided is examined. In all cases, both the speed of the temporal planners and the quality of their plans are compared.

The aim of this chapter is not to evaluate the planning and scheduling technology used in CRIKEY, but rather to evaluate the interaction between them, especially in domains where the components of planning and scheduling are highly coupled.

Points to Note The temporal planners being compared are written by different people and in different languages. Some implementations are more highly optimised than others, especially to certain domains (namely, the competition domains). Both these facts will affect the performance of the planners, not making it a completely fair comparison. Ideally, it is the core algorithms of the planners and their complexity that needs to be compared (for example, the number of states visited, or the complexity of the heuristic). Empirically testing them is only, albeit strongly, indicative of this. For this reason, it is sometimes better to view the rate of change of the planners performance as the complexity of the problems increases, rather than the actual timings. For all comparisons, the planners are run on the same machine with the same resources.

Just as CRIKEY is not designed with planning and scheduling in mind, but instead the communication between them, so other planners also have their own agendas. This will affect the performance of planners on the general problems.

5.1 Capabilities

A variety of planners have been chosen to compare their capabilities in temporal planning problems against those of both versions of CRIKEY. Only original planners are used (i.e. not extensions to planners that explore some non-temporal aspect of planning). Also, only planners where there is sufficient documentation or the source code is available are included. The documentation and previously published results are used to determine the capabilities, alongside testing the planners on a simple set of domains with the characteristics under comparison. In all cases, descriptions of the planners can be found in Section 2.5.

Table 5.1 compares the capabilities of different planners with regard to the complexity of concurrency that they can handle. Only CRIKEY, Sapa, VHPOP, and LPGP can handle domains with co-ordination. MIPS, LPG and TP4 cannot, and it is not thought that there are any other temporal planners that are able to (including the SAT-based planners). The planners that cannot find plans in these cases assume a blackbox durative action model, and fail to take into account start effects and end conditions.

Table 5.1: Temporal Planner Concurrency Capabilities

Temporal Planner	PDDL2.2 Timed Initial Literals (TIL)	TIL compiled to PDDL2.1	Single Hard Envelopes	Complex Multiple Envelopes
CRIKEY V1	✗	✓	✓	✗
CRIKEY V2	✗	✓	✓	✓
Sapa	✗	✗	✓	✗
MIPS	✓	✗	✗	✗
LPGP	✗	✓	✓	✓
LPG	✓	✗	✗	✗
TP4	✗	✗	✗	✗
VHPOP	✗	✓	✓	✓

Sapa uses a slightly different model of durative action to PDDL2.1. Effects can happen at any time during the duration of the action (and so the end effects of PDDL2.1 can be easily translated into Sapa's language). Conditions and invariants can hold for any arbitrary length of time but must start from the beginning of the action. This makes it impossible to correctly translate the end conditions which are not invariants. For this reason, Sapa is marked as not being able to solve envelopes many actions long since this often requires the use of end conditions. For example, Sapa cannot find a plan for Figure 5.1(b) (that contains end conditions), but can for Figure 5.1(a), whereas CRIKEY version 2 and VHPOP can find plans for both.

Sapa, whilst it should theoretically be able to plan with co-ordination where there are single hard envelopes, in practice cannot. The reason for this is two fold. When Sapa first finds a plan it does not respect the tolerance value correctly. This is partly because the

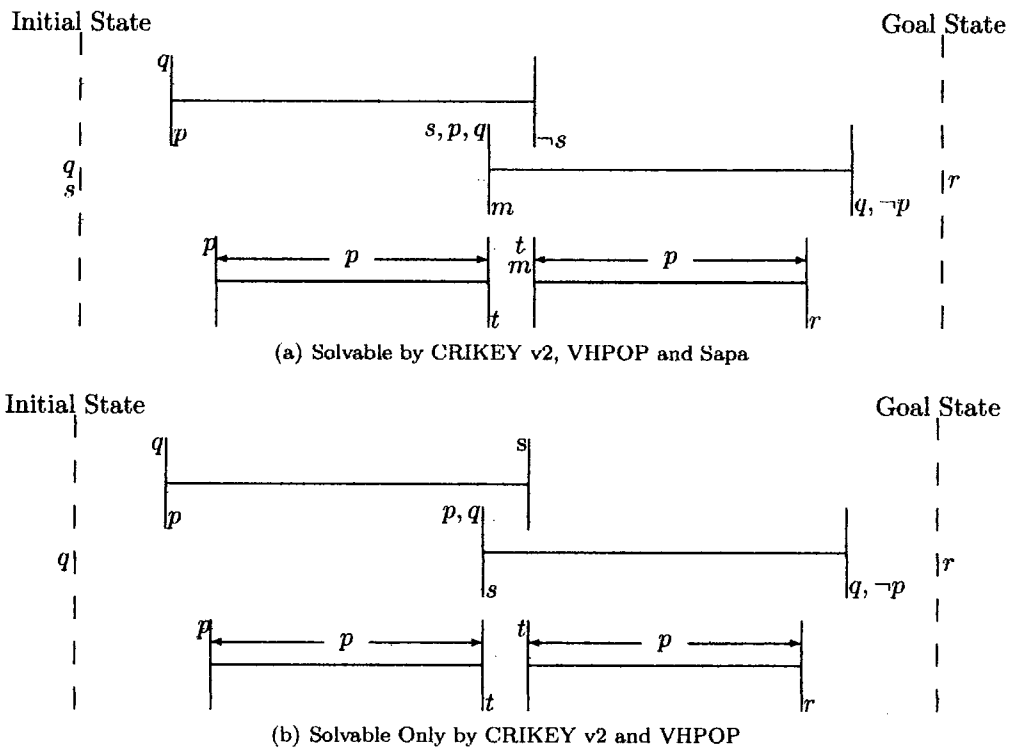


Figure 5.1: Two Possible Complex Envelopes

“advance-time” action would then only take the time forward by ϵ rather than to the next event in its queue. It post-processes the plan to optimise it and separate the actions by ϵ . However, this post-processing does not account for start effects (even though Sapa does whilst planning), and so wrongly places the content actions outside the envelope actions. Secondly, Sapa contains a bug whereby when it first searches for a plan, it can fail to check that an invariant of an action is not deleted by an action already in the queue.

Also LPGP theoretically should be able to find plans in domains containing co-ordination. However, a few modifications to the planner are needed. Often in domains involving co-ordination, it is the start effects of an envelope action that are required and not the ends. Since LPGP searches backwards in finding a plan, it must choose to place the (unwanted) end action in before it realises that it needs the start of the envelope, and so fails to find a plan.

Table 5.2 looks at the capabilities of these planners with respect to the kind of durative action it can support in PDDL2.1 (see Table 4.1).

Only CRIKEY version 2, Sapa and MIPS can plan with durative actions where the resource change is reliant on the duration of the action. Only CRIKEY version 2 and MIPS

Table 5.2: Temporal Planner Temporal Capabilities

Temporal Planner	Encoding	Resources	State Dependent Durations	Duration Dependent Resource Change	Duration Inequalities
CRIKEY V1	STRIPS [†]	✓	✓	✗	✗
CRIKEY V2	STRIPS [†]	✓	✓	✓	✓
Sapa	STRIPS [†]	✓	✓	✓	✗
MIPS	ADL	✓	✓	✓	✓
LPGP	STRIPS	✗ ¹	✗	✗	✗
LPG	ADL ²	✓	✓	✗	✗
TP4	STRIPS [†]	✓ ³	✗	✗	✗
VHPOP	ADL	✗	✗	✗	✗

[†]Can handle Typed STRIPS domains

can handle duration inequalities. CRIKEY makes some assumptions as to the nature of these as set out in Section 4.3.2. Again it is not thought that there are any other planners with these capabilities.

5.2 IPC'04

The competition was run over a period of approximately three months during which time competitors ran their planners on a series of problems on a Linux PC with two CPUs running at 3GHz. For each problem, planners were limited to 1GB of memory and 30 minutes of CPU time. During the competition, competitors were allowed to modify their planners to correct bugs and optimise them for the domains.

There are 7 domains: airport, pipesworld, promela, PSR, satellite, settlers and UMTS. These are described below and in more detail in [44]. The domains are split into “domain versions”, which relates to the number of PDDL2.2 features in the problem (for example, STRIPS only, fluents, durative actions etc...). Competitors were encouraged to tackle as many versions as their planner could handle. They then choose a “version formulation”. Each formulation had equivalent problems, but expressed differently. The formulation refers to STRIPS, ADL, and whether the new features in PDDL2.2 of derived predicates and timed initial literals are compiled down to PDDL2.1. Some of the domains (satellite and settlers) did not have non-ADL formulations and so CRIKEY could not compete in these domains.

There is no co-ordination in any of the competition domains, except for where PDDL2.2 timed initial literals are compiled into PDDL2.1 domains. In these cases, the dummy actions

¹LPGP can handle static fluents that do not change during planning.

²LPG is unable to handle conditional effects.

³TP4 can only handle resources that model reservoir resources and not the full range of fluent variables possible in PDDL2.1.

involved in the compilation require envelopes. The temporal aspect of the domains are further limited since there are no state dependent durations.

Planners were compared with those that used both the same version and formulation as itself. On quality, it was decided by the competition organisers to only compare planners that were trying to optimise the same criteria. There were three options: the makespan of the plan, the number of actions in the plan, or the quality metric provided in the problem. The theory behind this decision is that it does not make sense to compare two planners that are trying to solve different problems (by optimising different factors). CRIKEY was evaluated by the total number of actions in the plan it produced and this is what is referred to by “quality” in the results presented here.

Furthermore, in the competition, optimal planners were compared separately from sub-optimal planners. Comparisons for the competition were done informally, by simply looking at the results and judging who performed best.

Results from the competition are presented here to show that CRIKEY is competitive in general benchmark domains. The planning and scheduling technology is not novel or “cutting edge” but simple and well known in such domains. For this reason and the fact that CRIKEY was not optimised during the competition, it was not expected to perform outstandingly.

PSR

In this domain the goal is to resupply a number of lines in a faulty electricity network. The flow of electricity through the network, at any point in time, is given by a transitive closure over the network connections, subject to the states of the switches and electricity supply devices. The problems rely heavily on derived predicates, of which only the smallest could be translated. All versions of this domain are non-temporal.

Results for performance and plan quality are shown in Figures 5.2(a) and 5.2(b). There is little difference between the competing planners. No planner performs consistently better than any other, and, with the exception of LPG, the quality is comparable for all planners.

The domain shows CRIKEY performing competitively against state of the art planners in classical propositional planning and solving 29 of the 50 problems.

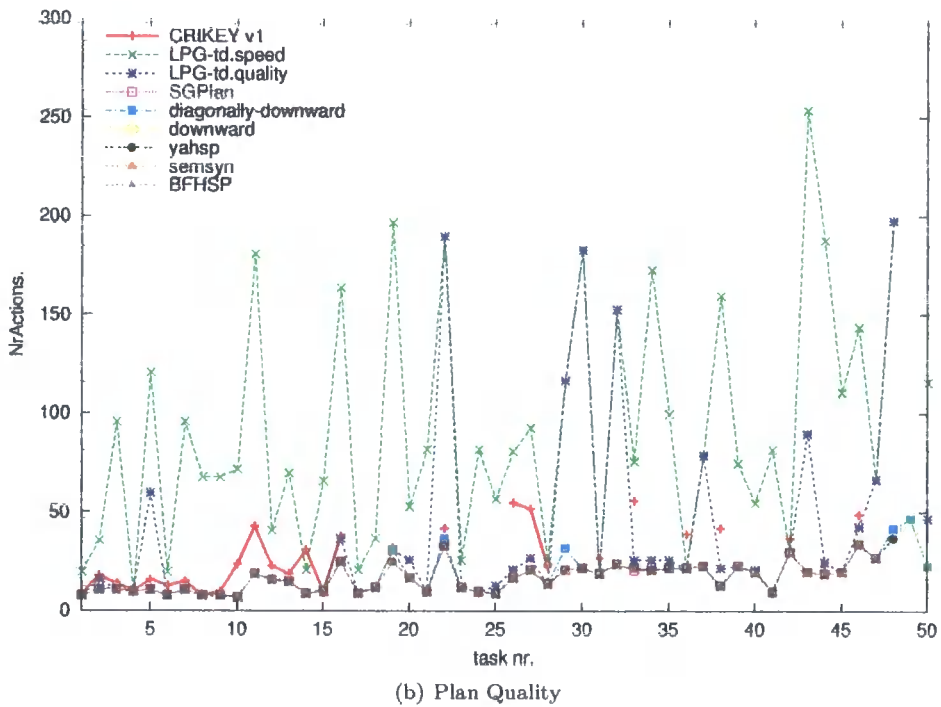
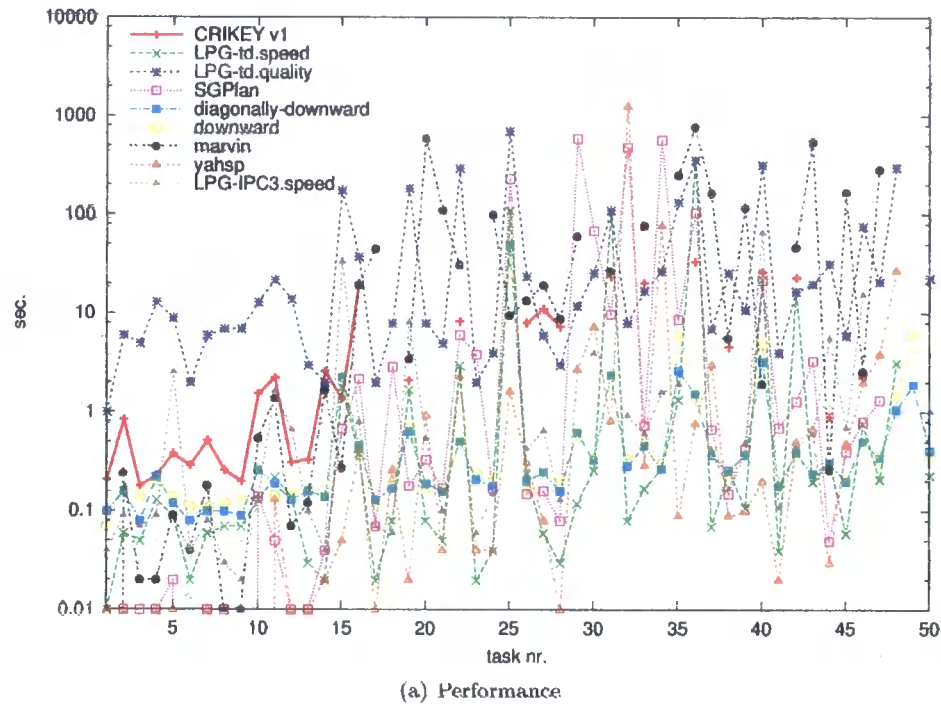


Figure 5.2: Non-temporal Small PSR Domain

Promela Domain

The goal in this domain is to find deadlocks in communication protocols, translated into PDDL from the Promela specification language. The communication protocols used in the competition were the dining philosophers problem, and an optical telegraph routing problem. CRIKEY only competed in the non-temporal domain versions since the other versions all contained ADL, which was impractical to compile to STRIPS.

Figures 5.3 and 5.4 show the results for these domains. The plans are all of the same length as there is only one solution (plan) to bring the system to deadlock (the goal). In problem 7 of the dining philosophers domain, CRIKEY put in some irrelevant actions due to a bug in the code. CRIKEY, as with Macro-FF and P-MEP, has a significantly greater gradient than FAP, SGPlan, and YAHSP in the performance graph, where the scale is logarithmic. This shows that the performance is much worse. This is further shown as CRIKEY solves fewer problems than those planners. Given more resources, CRIKEY would have continued to solve the problems but with a continued deterioration in performance.

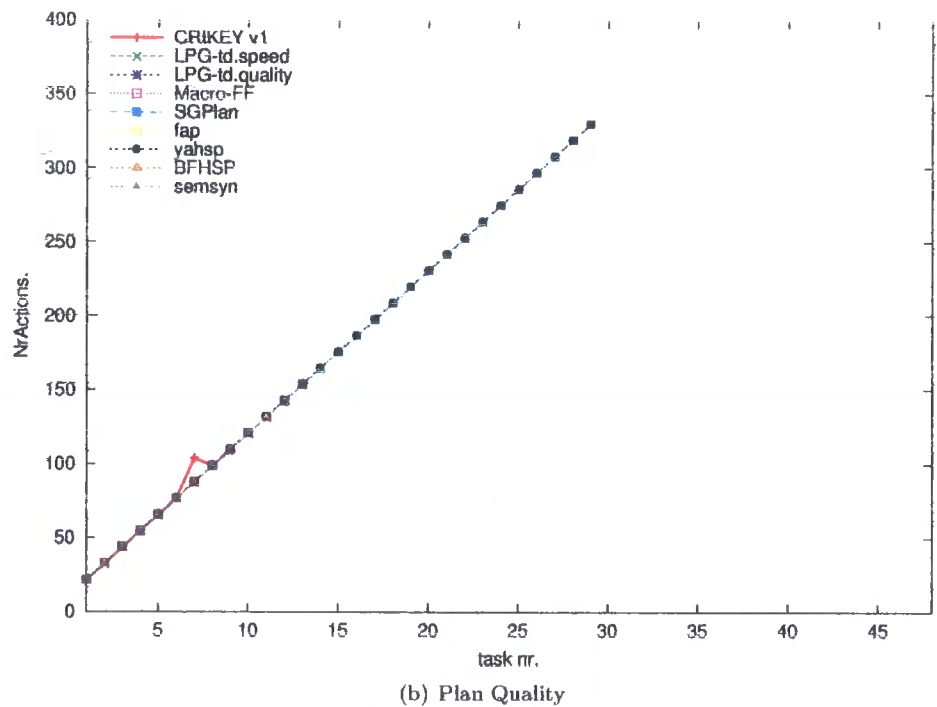
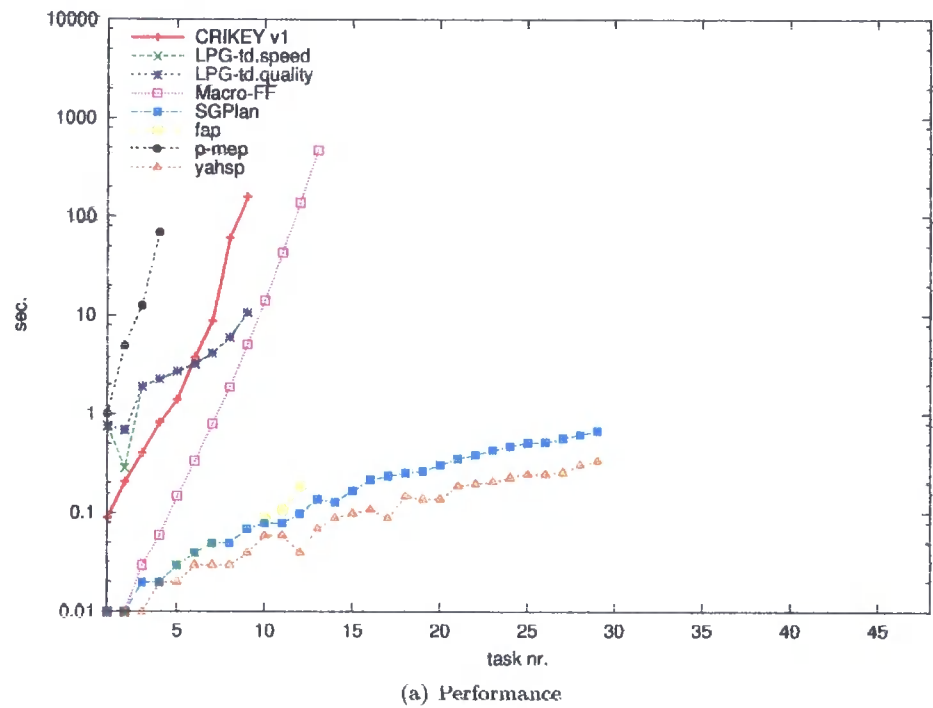


Figure 5.3: Non-temporal Dinning Philosophers Domain

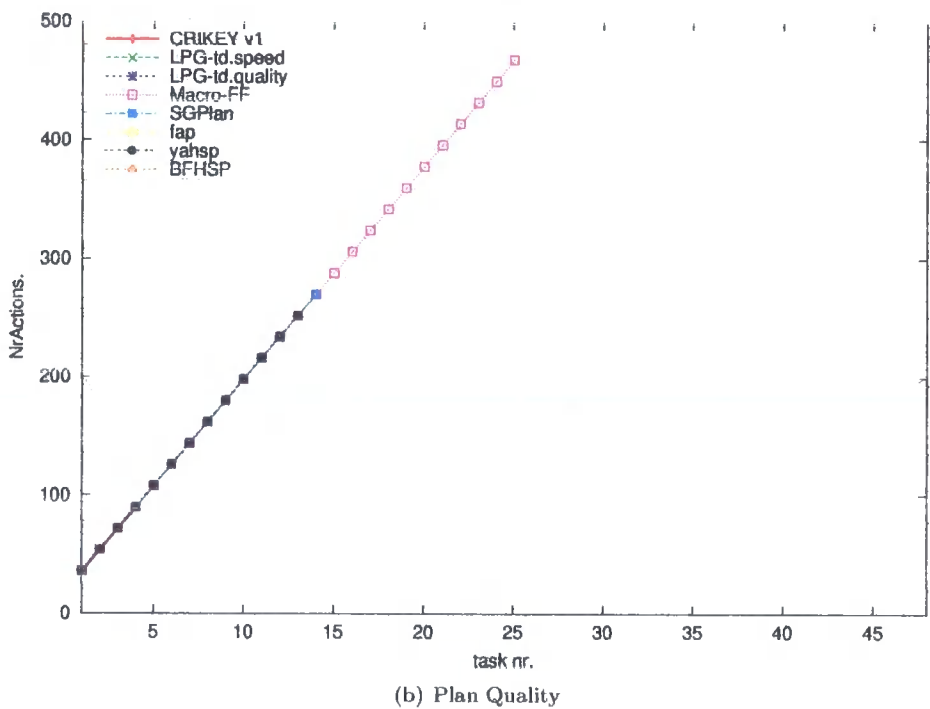
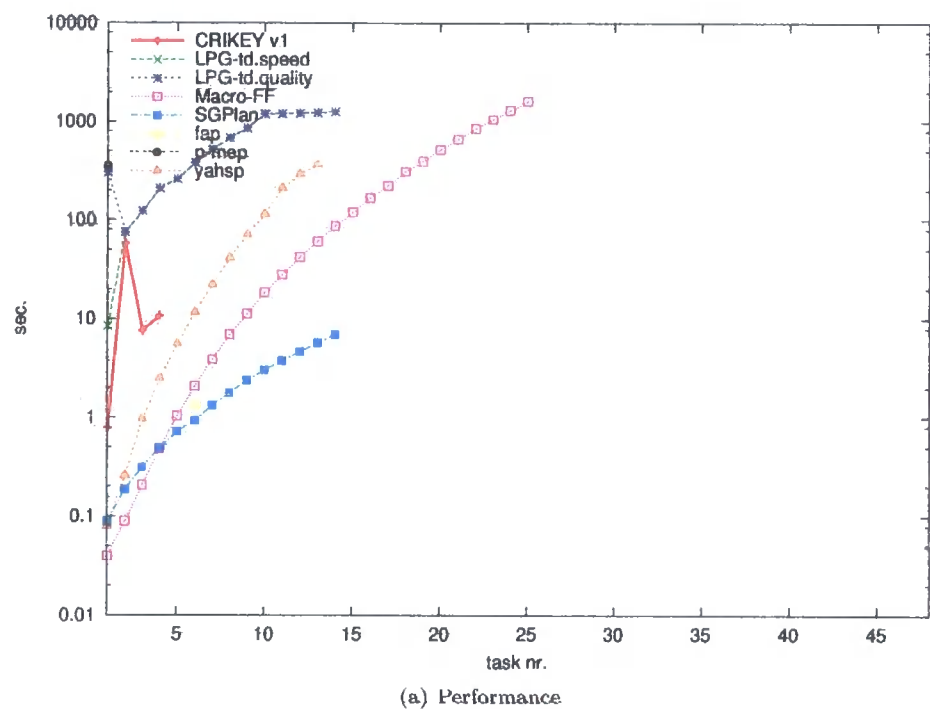


Figure 5.4: Non-temporal Optical-Telegraph Domain

Pipesworld

In this domain, the object is to control the flow of oil derivatives through a pipeline network, obeying various constraints such as product compatibility and tankage restrictions. One interesting aspect of the domain is that if something is inserted into one end of a pipeline segment, something potentially completely different can come out at the other end. CRIKEY competed in four domains, two without resources (no-tankage) and two with resources (tankage). Of these domains, one was non-temporal, the other was temporal. The results are shown in Figures 5.5, 5.6, 5.7, and 5.8.

In all these versions CRIKEY performed competitively showing that it can compete in both temporal and metric planning problems. In the temporal metric version it solves problems that no other planner does. It proves that the decomposition of temporal planning into planning and scheduling is a viable solution.

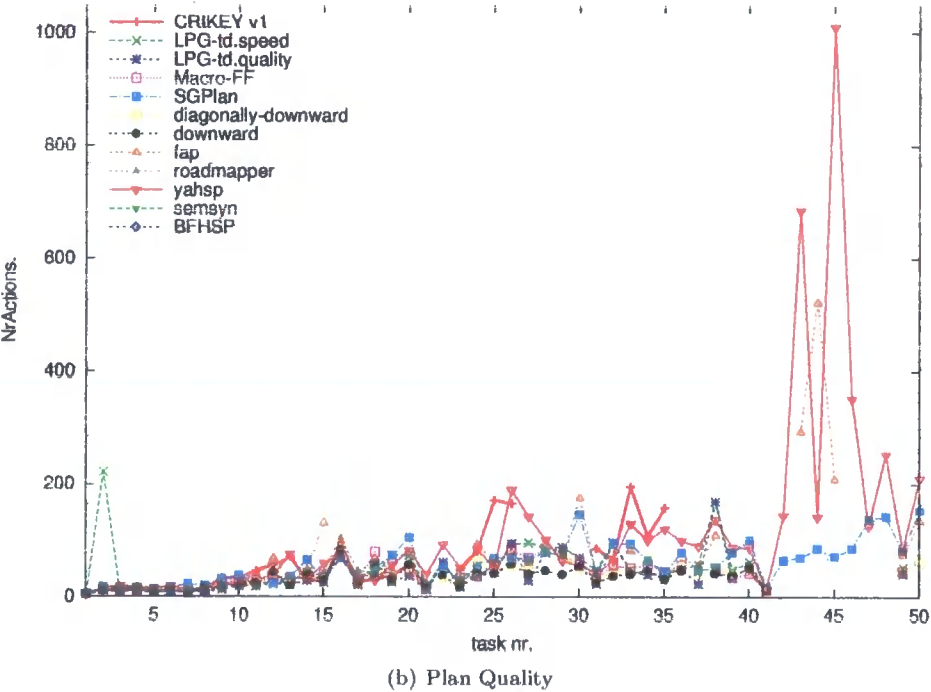
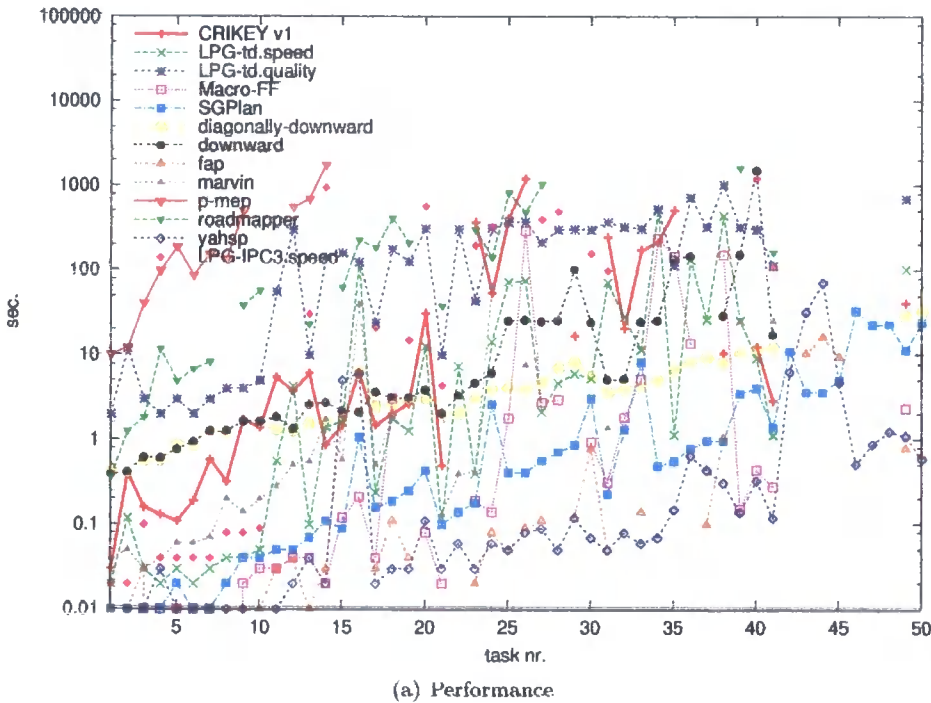


Figure 5.5: Non-temporal No Tankage Pipesworld Domain

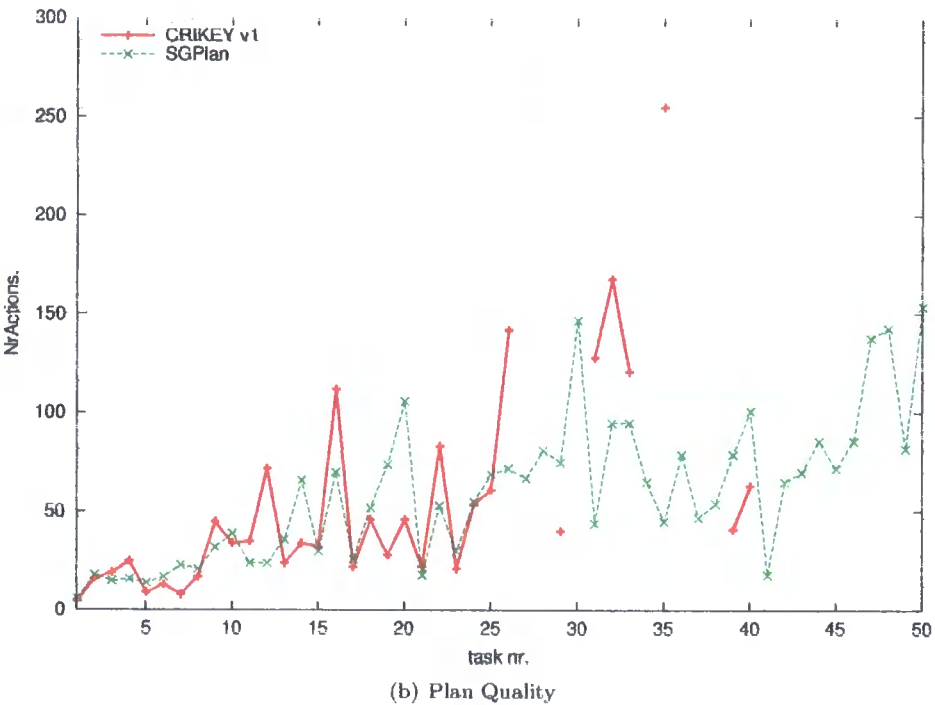
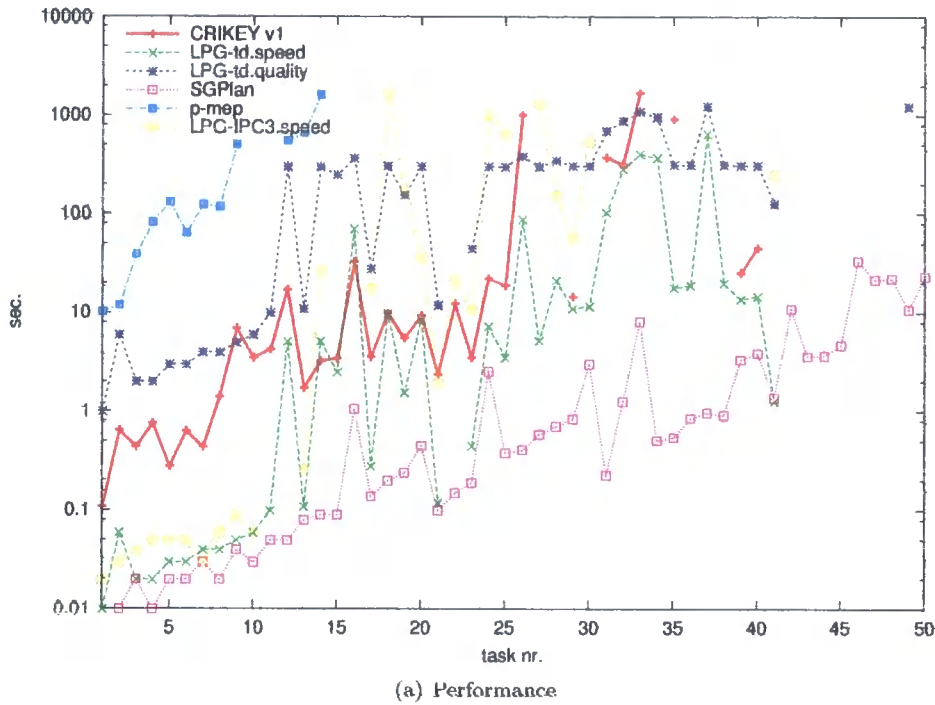
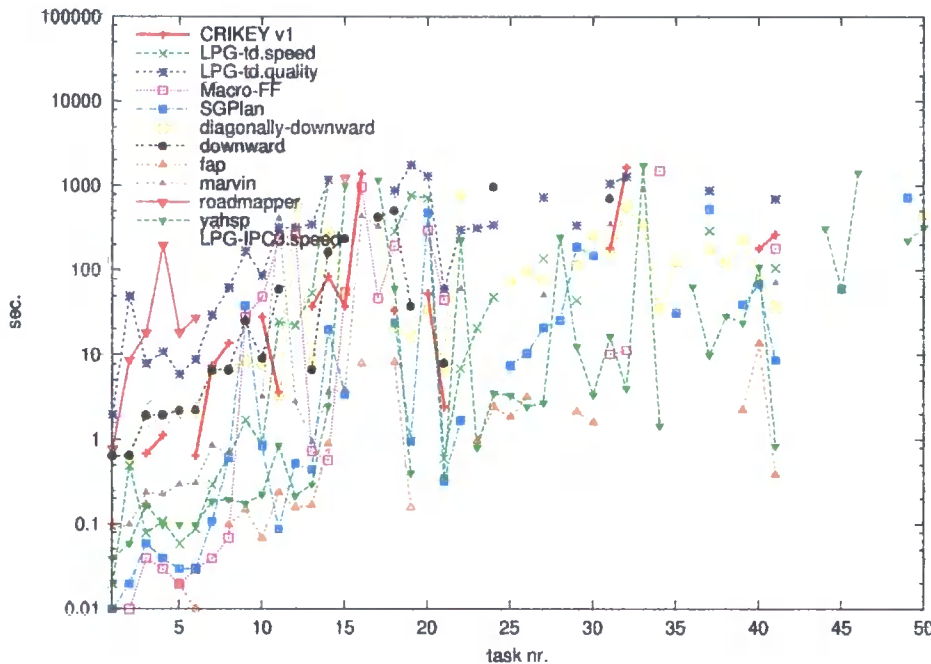
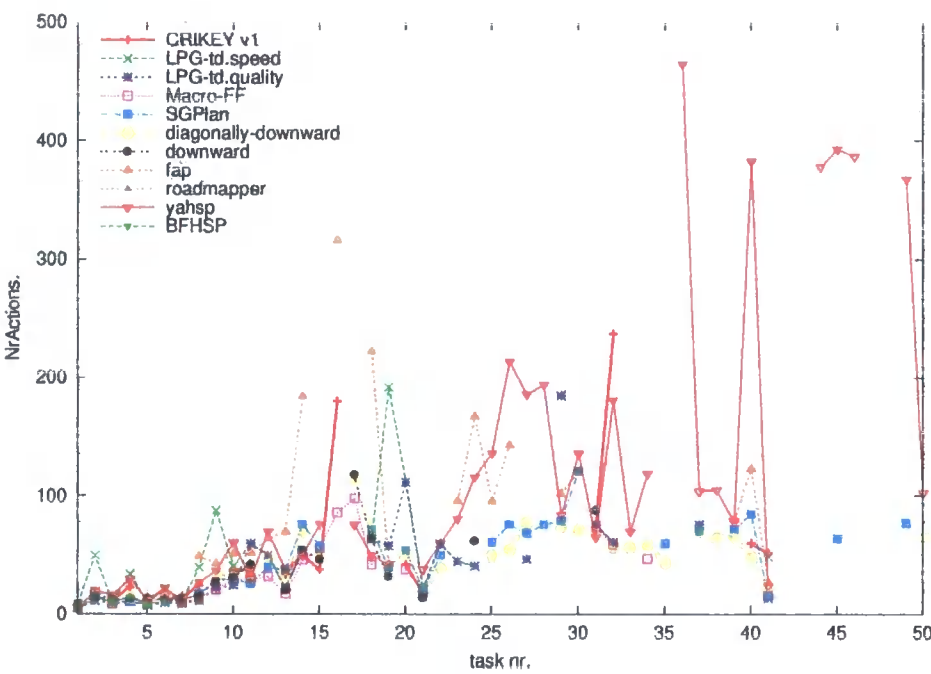


Figure 5.6: Temporal No Tankage Pipesworld Domain



(a) Performance



(b) Plan Quality

Figure 5.7: Non-temporal Tankage Pipesworld Domain

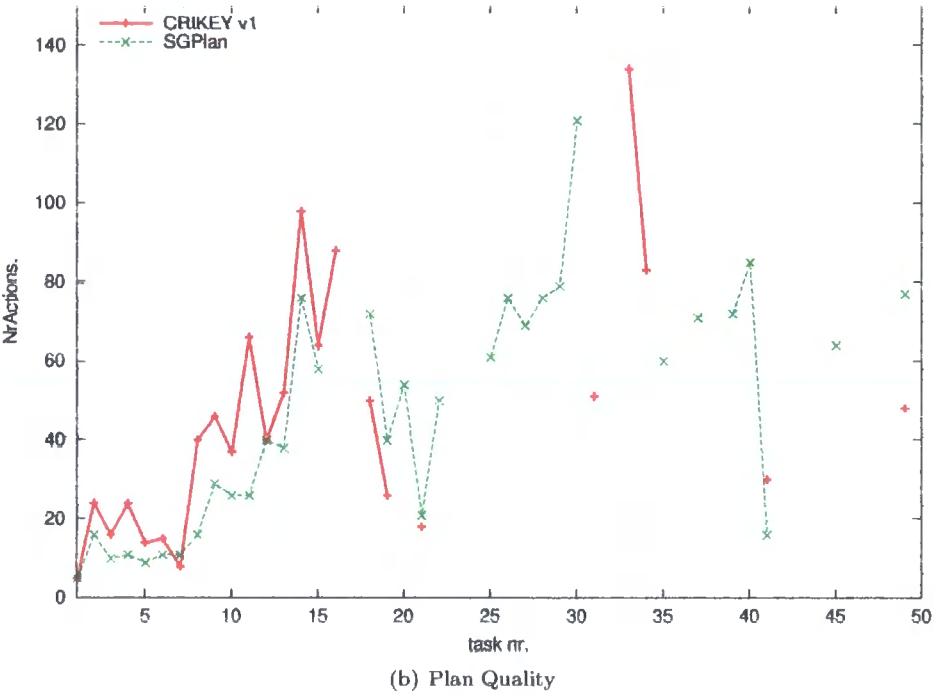
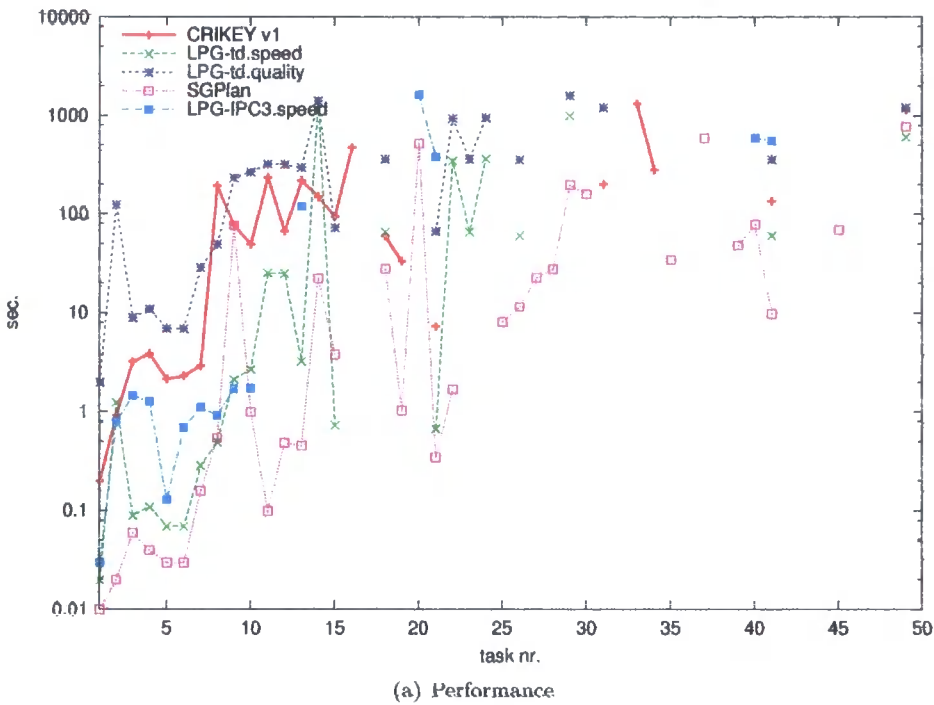


Figure 5.8: Temporal Tankage Pipesworld Domain

UMTS

In the UMTS domain, the task is to set up applications for mobile terminals. The objective is to minimise the time needed for the set up, i.e. to minimise the makespan of the plan. If this objective is ignored then the planning is trivial. CRIKEY competed in three versions of this domain: a temporal domain (Figure 5.9), a flawed temporal domain (Figure 5.10) and a temporal domain with time windows which had been compiled down to PDDL2.1 (Figure 5.11).

CRIKEY managed to solve almost all the problems in this domain⁴. Whilst it is not as quick as other planners competing in the standard temporal version, its performance degrades at a similar rate and so this could be due to implementation differences.

The flawed temporal domain was created to deliberately disrupt planners guided by relaxed temporal plans, such as CRIKEY. It has an action that could achieve a goal in one step, but this deletes other goals and so cannot be used. CRIKEY resorts to Best First Search which leads to a deterioration both in performance and in the quality of the plan. This shows the fragility of the relaxed plan heuristic.

Only two planners competed with the time windows compiled into PDDL2.1: CRIKEY and SGPlan. Whilst CRIKEY solved all problems in reasonable time (less than 100 seconds), SGPlan is faster still. In fact, CRIKEY and SGPlan were finding the same plans. The reason for the difference in the number of actions is that CRIKEY includes the dummy actions in the total action count, whereas SGPlan does not.

This domain shows that CRIKEY can handle co-ordination when it is in the form of time initial literals compiled into PDDL2.1.

⁴Those it did not solve were discovered later to be due to a bug in detecting repeated visited states.



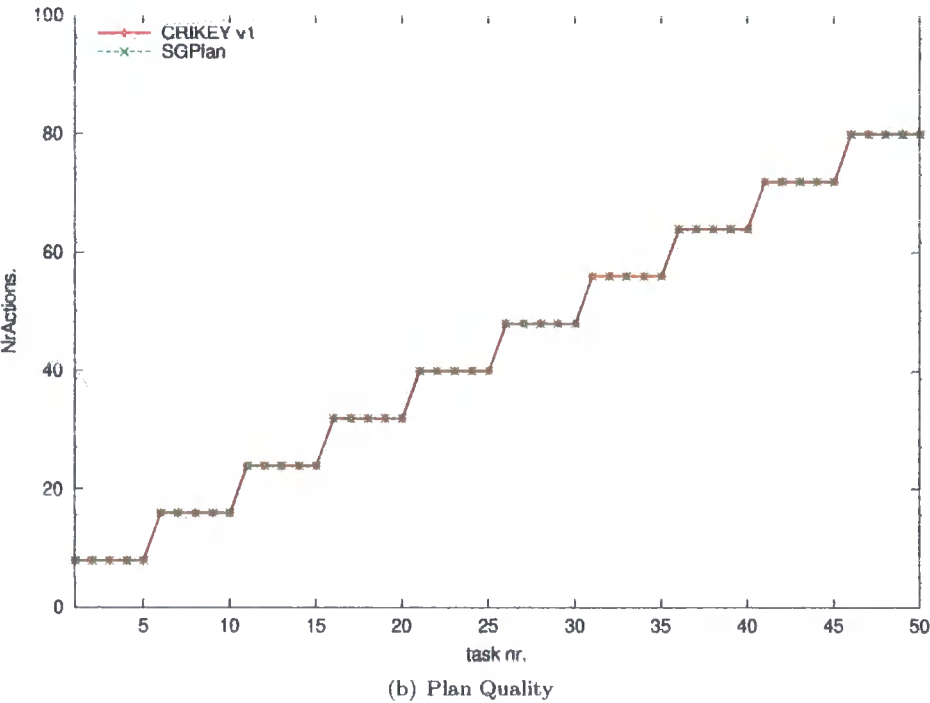
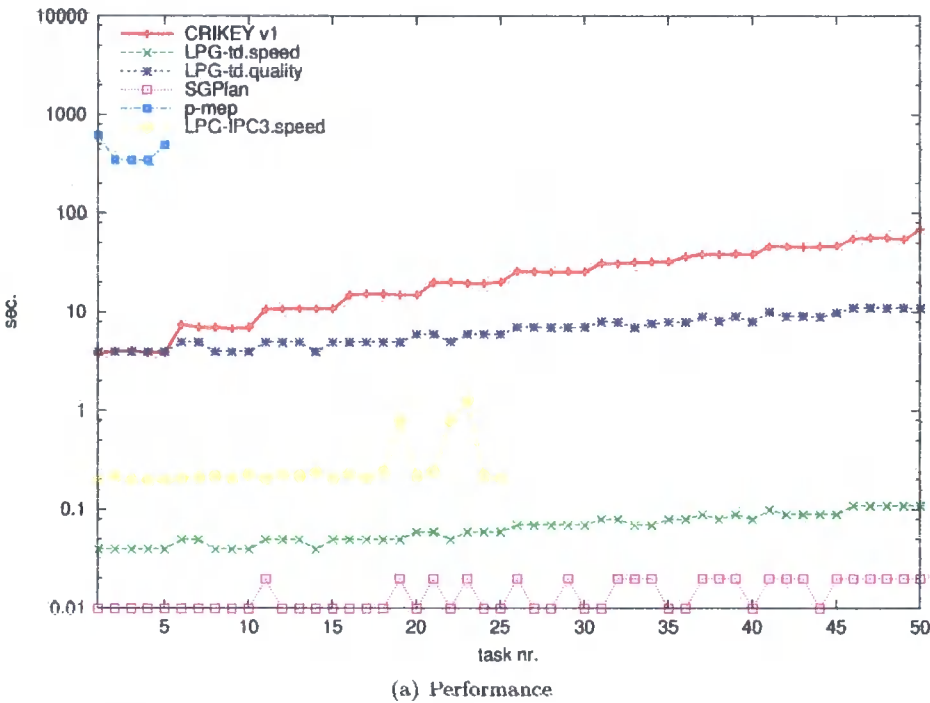


Figure 5.9: Temporal UMTS Domain

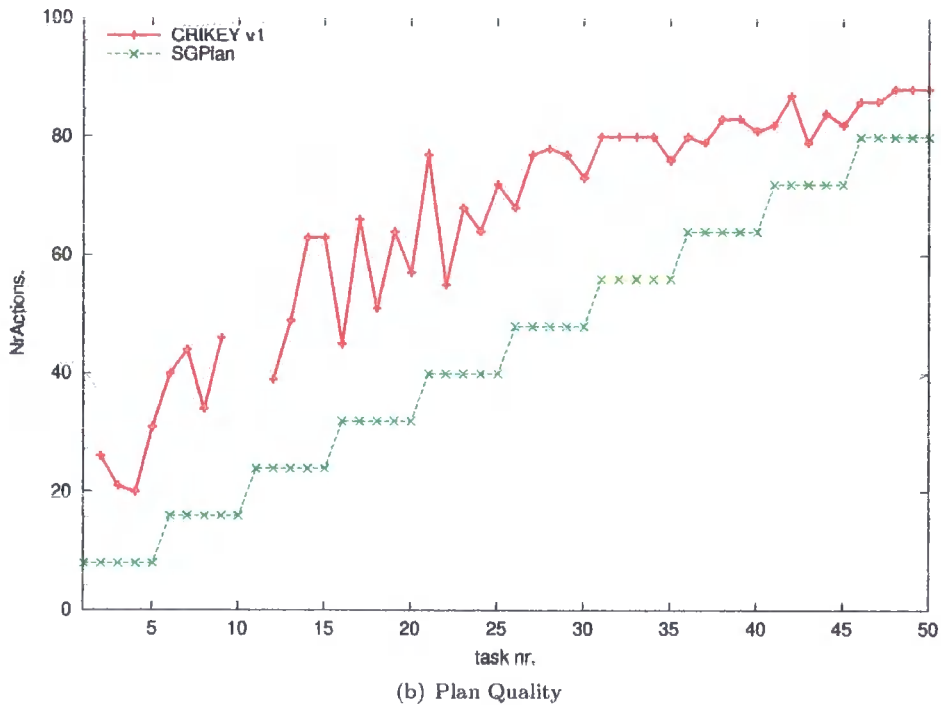
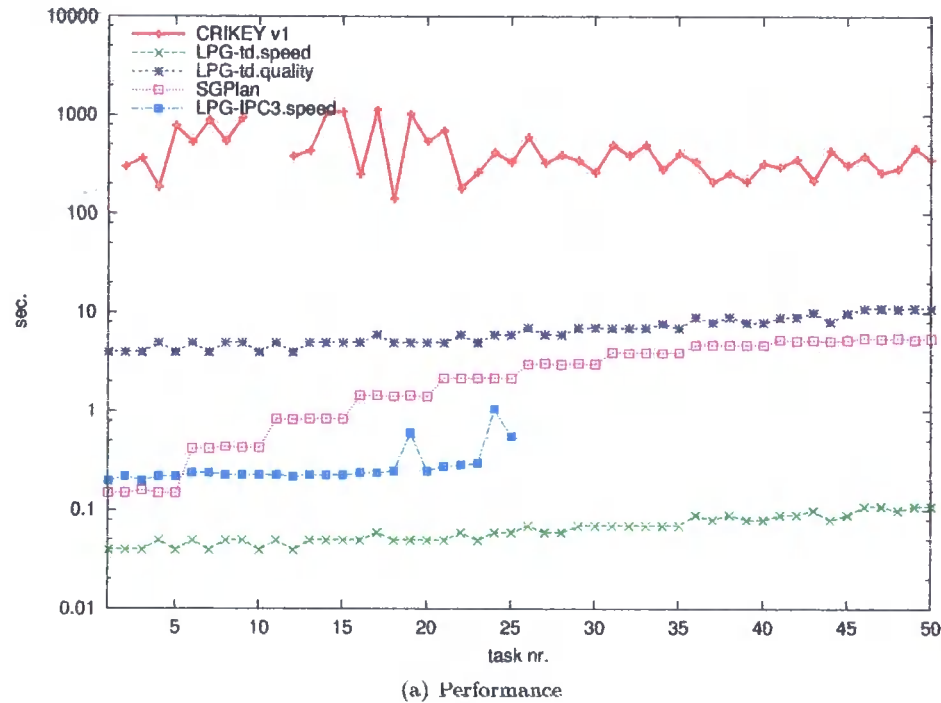
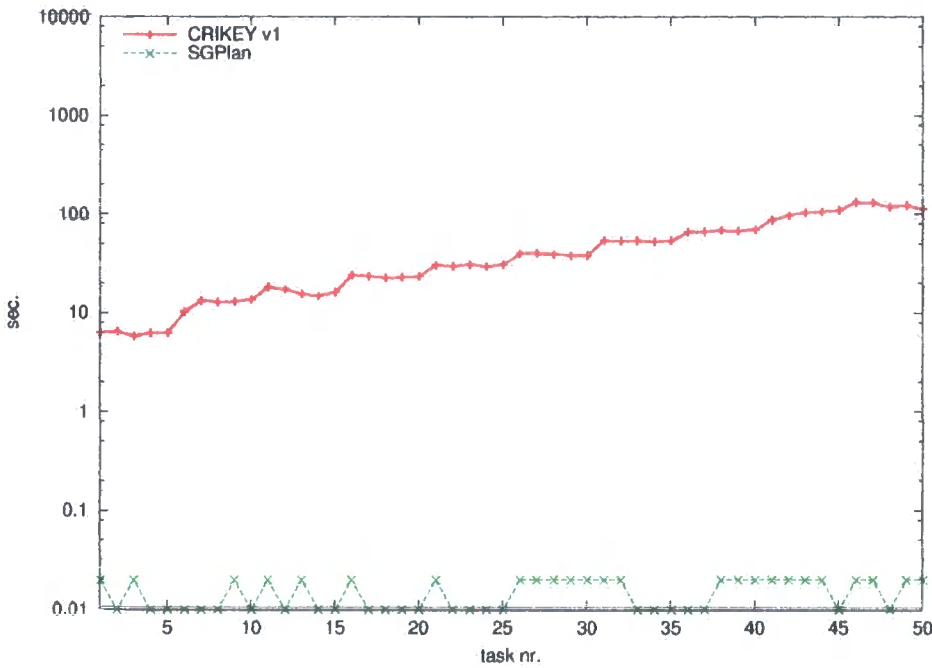
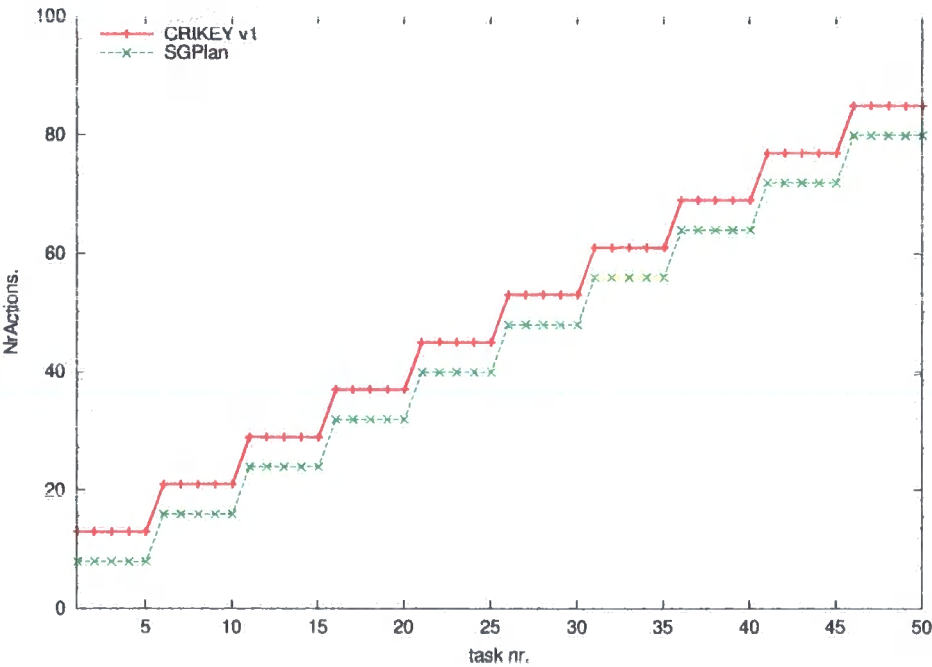


Figure 5.10: Temporal Flawed UMTS Domain



(a) Performance



(b) Plan Quality

Figure 5.11: Temporal UMTS Domain with compiled Time Windows

Airport Domain

The purpose in this domain is to control ground traffic in airports, moving planes between gates and runways safely. The largest instances (problem numbers 21–50) in the test suites are realistic encodings of Munich airport. CRIKEY competed in the non-temporal version (Figure 5.12), the temporal version (Figure 5.13) and the temporal version with deadlines compiled into PDDL2.1 (Figure 5.14).

Again, CRIKEY performs competitively in all versions of this domain and was ranked second in the competition for the propositional, sub-optimal airport domain. Where there is co-ordination in the compiled time windows versions, CRIKEY finds solutions that other planners do not.

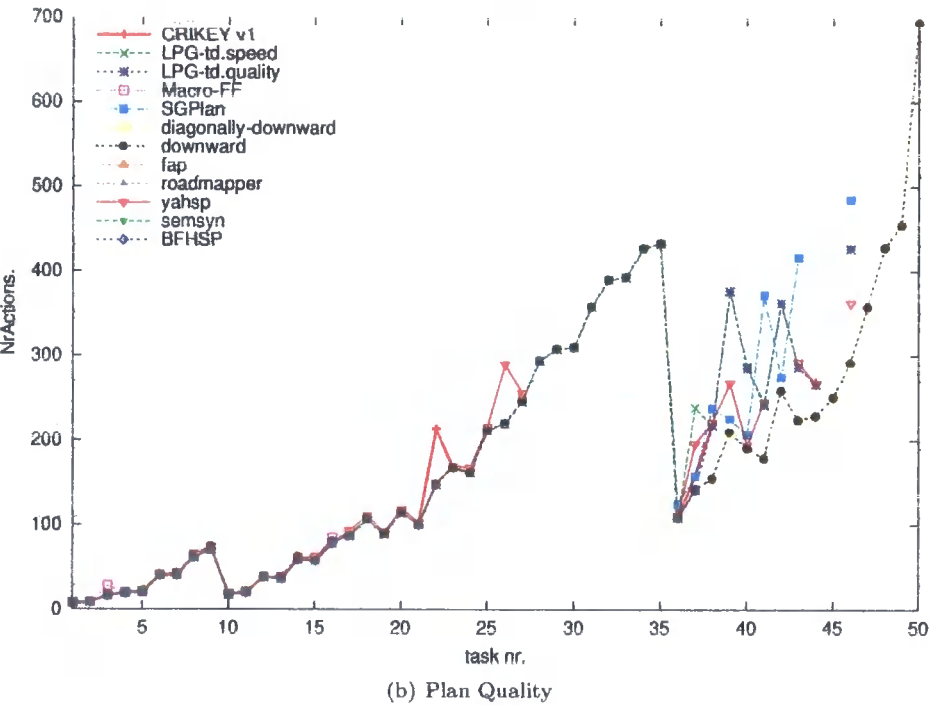
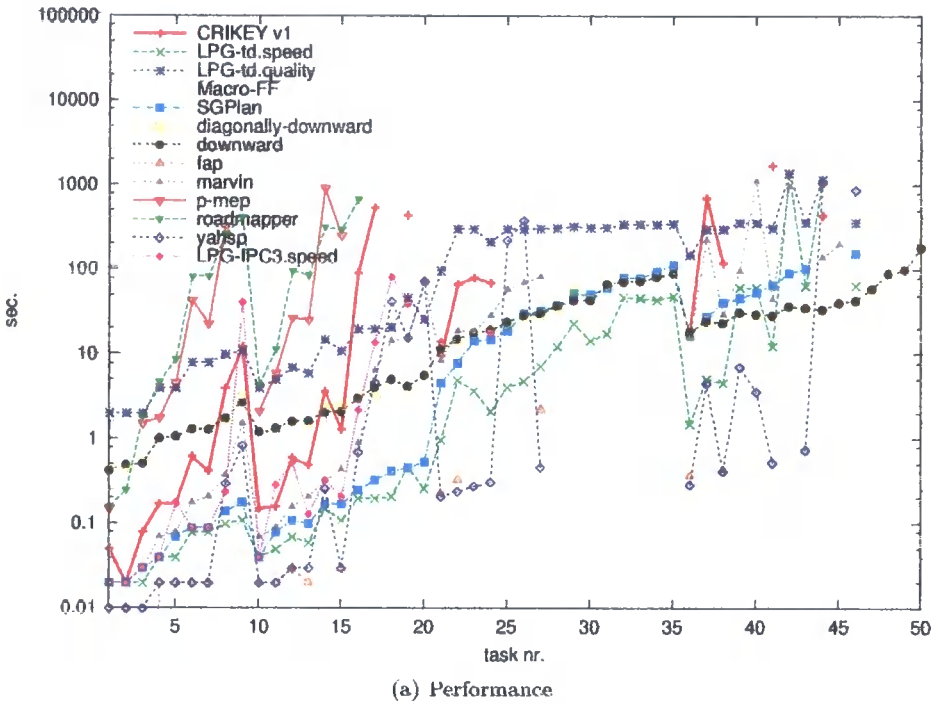


Figure 5.12: Non-temporal Airport Domain

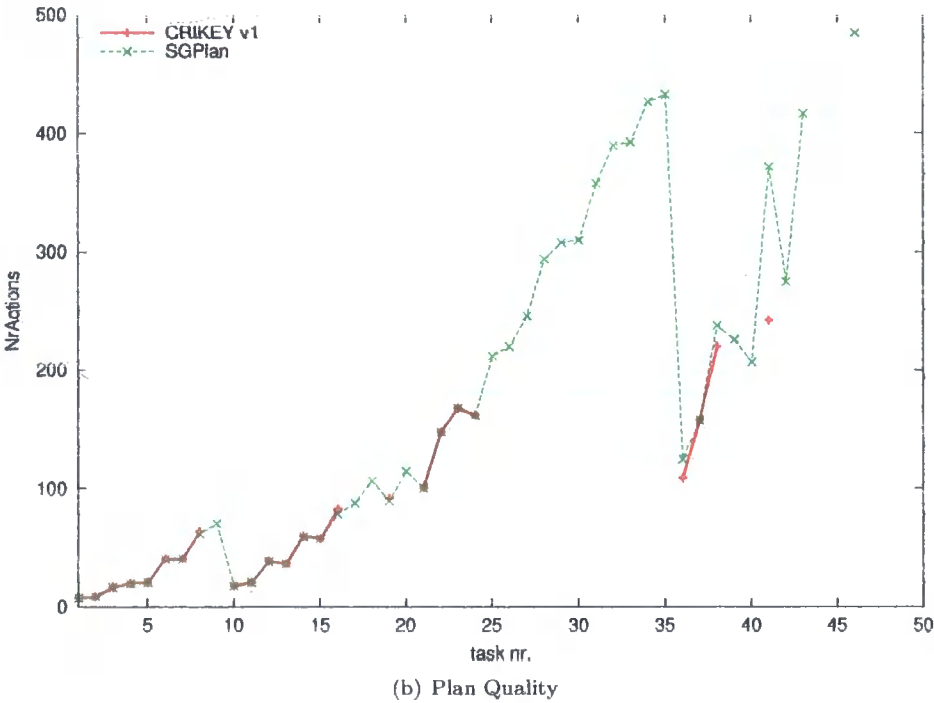
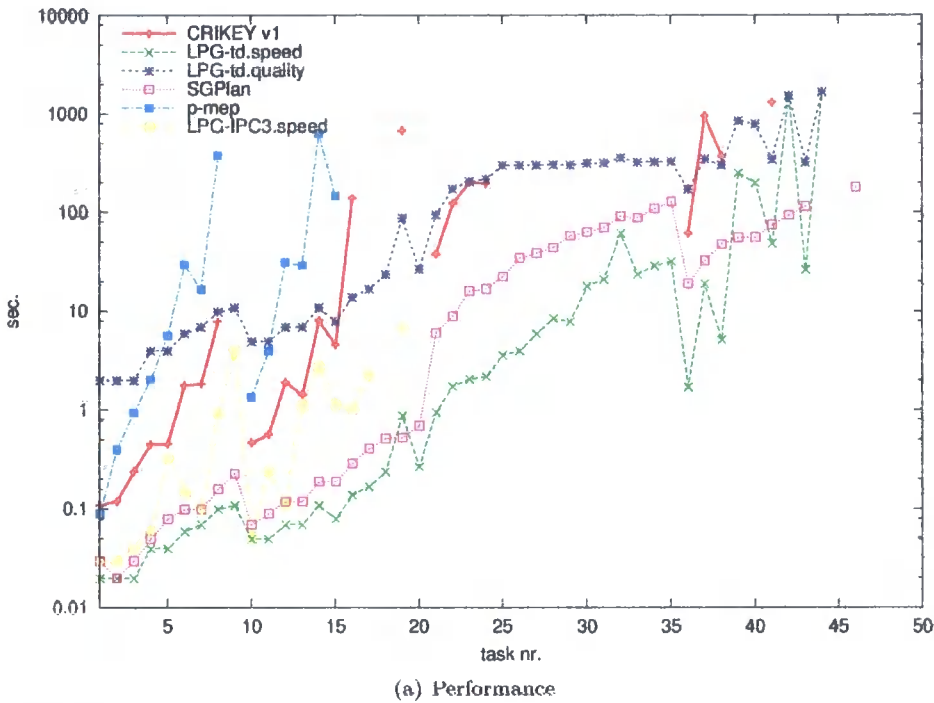


Figure 5.13: Temporal Airport Domain

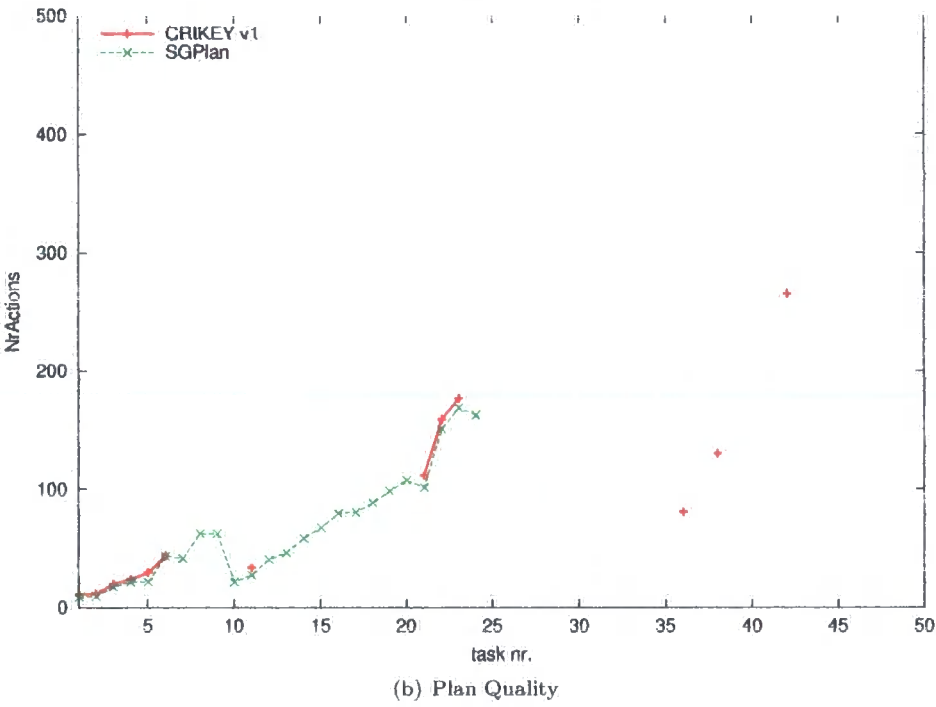
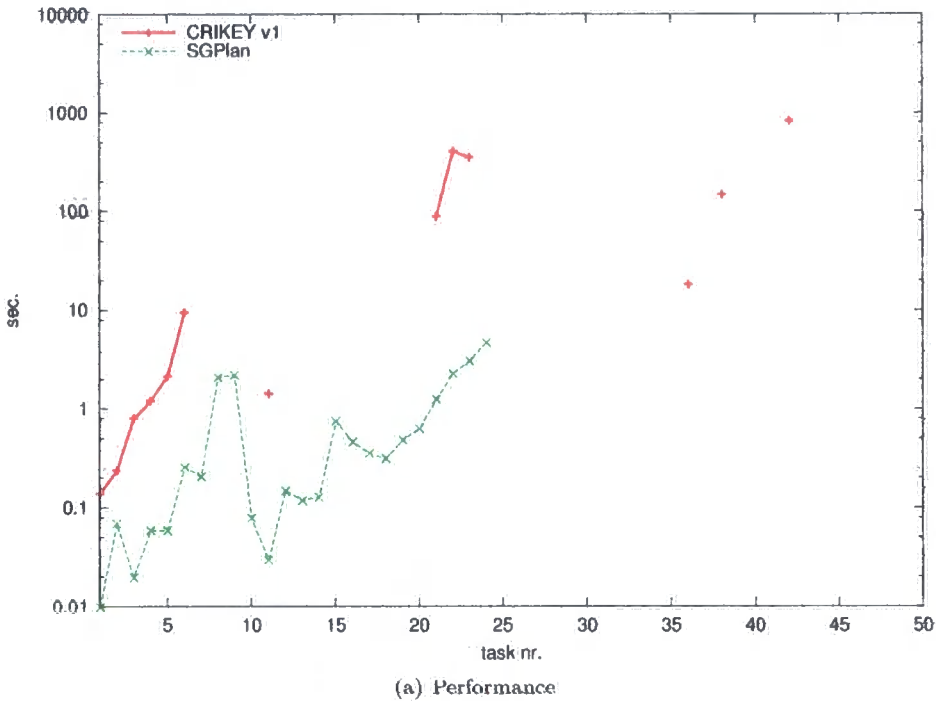


Figure 5.14: Temporal Airport Domain with Time Windows

5.2.1 Analysis Overview of IPC'04 Domains

These competition results show that CRIKEY is a temporal planner that performs reasonably well in propositional, metric and temporal benchmark domains. CRIKEY's implementation is not optimised and its design and algorithm are not intended to be outstanding. Particular issues (namely co-ordination) not present in the domains were focused on instead. However, CRIKEY is more expressive than the other planners competing (as shown in Section 5.1 at the beginning of this chapter), with the possible exception of the poorly performing P-MEP. By limiting the expressive power of the problems, assumptions are made as to the nature of the problems. These assumption lead to a decrease in the computation necessary which in turn leads to better performance. CRIKEY does not make these assumptions, and so whilst it can plan for more domains (see the rest of this chapter) it pays for it in its performance in general domains.

5.3 Co-ordination

In this section, those planners that can handle co-ordination are compared against one another as before, but on domains specifically designed to contain co-ordination. The machine from the IPC'02 competition was used and is a Linux PC running at 800Mhz. The planners had 500MB of memory and a time limit of twenty minutes. This is significantly less resources than for the IPC'04. To compensate for this the problem instance sizes are smaller. One reason for the reduction in resources available is that the difficulty in the problems does not come from the size of the instance but from the interaction between the planning and scheduling (i.e. the co-ordination) and it is of more interest to know whether the planners easily find the solution in the search space (if at all) and not actually how long the planners take.

The performance graphs are now on a linear scale (not logarithmic), and the quality is no longer calculated by the number of actions in the plan but by the temporal length of the plan. The domains contain forced concurrency (in the form of co-ordination) and so the temporal length of the plan is quite separate from the number of actions in the plan. Content actions are effectively not counted as it is the envelope actions that account for the length of the plan. It is these actions that a good planner will want to minimise. By comparing the temporal length, it is the scheduler, planner and their interaction which is really being tested, whereas when comparing the number of actions, the scheduler has no impact on the quality. It is also important to compare the temporal length of the plan where there are duration inequalities, since the number of actions will remain the same regardless of their duration.

5.3.1 The Match Domain Revisited

Both versions of CRIKEY, VHPOP and Sapa were all tested on 4 variations of the match domain, based on the domain initially presented in Section 3.2 and in full in Appendix C. However, VHPOP and Sapa do not obey PDDL2.1 semantics quite as closely as CRIKEY, and where an invariant of an action is achieved by the start effects of an action (as in the LIGHT_MATCH action), the planners report that no plan can be found. For these tests these invariants are removed. This does not affect the meaning of the domain as the fuses must still be fixed within the burning of the match. The necessary changes to LPGP could not be made and so is not included in these domains.

Figure 5.15 presents results for the standard match domain. In this domain it takes five time units to fix a fuse, and a match burns for eight time units. The number of matches and fuses in the instance is twice the task number (e.g. problem number 5 has 10 matches and 10 fuses to fix).

As discussed earlier, Sapa fails to find valid plans. It realises that more than one match is required, but produces plans where two fuses are fixed by the light of one match, resulting in plans with half the number of matches required. For this reason, Sapa is compared only for its performance.

VHPOP can use a multitude of search strategies, flaw selection preferences and heuristic guidance. Some experimentation was performed to find out which combination work best in the match domain and it was found that A* search with the ADD heuristic and preferring plans with few open conditions.

As can be seen, CRIKEY version 1 performs significantly better than version 2. This is because a plateau is reached in the search space when one fuse has been fixed by the light of one match and the planner is trying to fix another fuse. The heuristic in this case does not guide the planner to close the envelope and light another match. Instead the planner must perform a small amount of search to discover this, including checking that all of the unfixed fuses are not able to be fixed using the rest of the available light. Version 2 splits all its actions into 2 actions, whereas version 1 compresses the fix fuse actions into a single action. For this reason, the size of the search at every plateau where a new match is needed is twice as big for version 2 as version 1, and so takes longer.

Table 5.3 shows the percentage of time that both versions of CRIKEY spend on parsing and instantiation, planning, and scheduling. The reason that version 2 spends proportionally longer planning is again attributed to the larger search space. Both versions spend most of their time planning as this is the harder problem to solve. The scheduler is not complex, finding a quick greedy solution.

All planners find the same (optimal) solution.

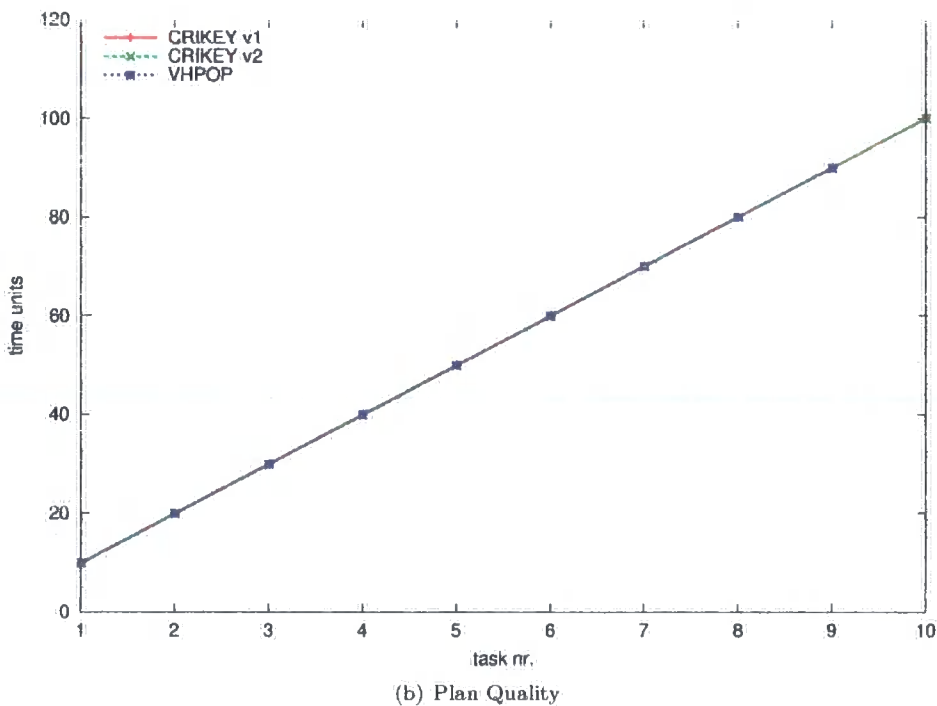
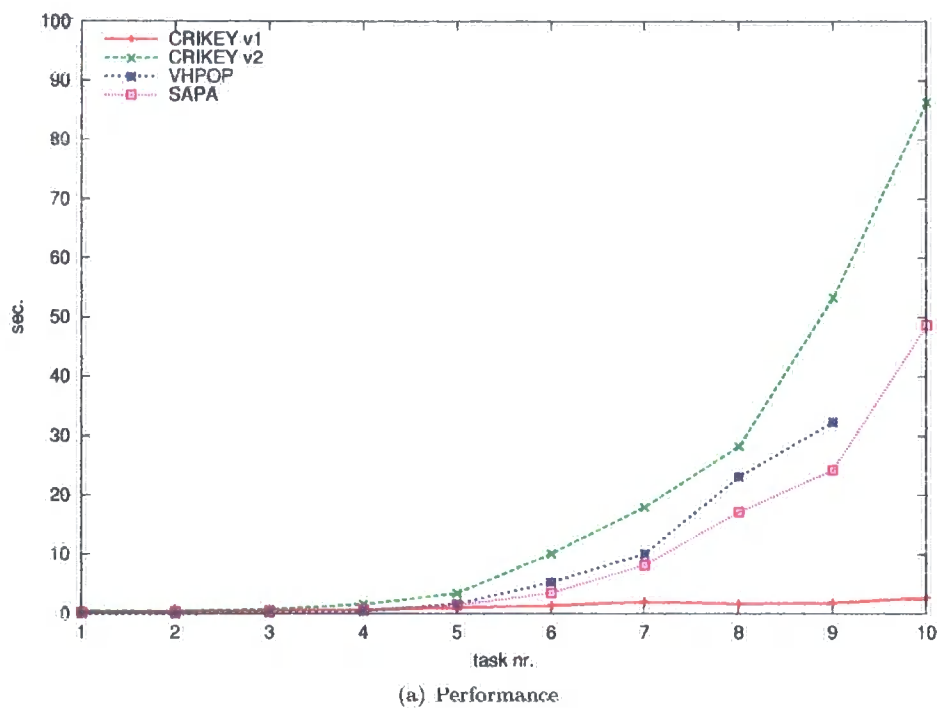


Figure 5.15: Standard Match Domain

Table 5.3: Percentage of Time Spent in Temporal Planning by CRIKEY in the Match Domain

Problem	CRIKEY v1			CRIKEY v2		
	Parsing & Grounding	Planning	Scheduling	Parsing & Grounding	Planning	Scheduling
1	33.33%	33.33%	33.33%	33.33%	33.33%	33.33%
2	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
3	0.00%	100.00%	0.00%	0.00%	66.67%	33.33%
4	0.00%	66.67%	33.33%	12.50%	62.50%	25.00%
5	0.00%	50.00%	50.00%	5.56%	66.67%	27.78%
6	0.00%	37.50%	62.50%	2.94%	76.47%	20.59%
7	8.33%	33.33%	58.33%	0.17%	97.97%	1.86%
8	6.25%	31.25%	62.50%	0.79%	84.25%	14.96%
9	5.00%	30.00%	65.00%	0.37%	83.52%	16.10%
10	4.00%	32.00%	64.00%	0.36%	91.55%	8.09%

Match Domain with Variable Durations

Figure 5.16 show the results for a variant of the standard match domain. In this domain, the fuses take different times to fix and different matches also burn for different durations. A fuse that takes a long time to fix must be fixed by the light of a match that burns for a sufficient amount of time. This match must therefore not be wasted on another shorter fuse.

The light match action is changed⁵ so that only one match can be alight at any one time. This makes it advantageous to fix as many fuses as possible by the light of one match, in order to minimise the temporal length of the plan. This variant is effectively a bin packing problem.

This variant uses fluents to model the burning time of the match and also the mending time of the fuse. VHPOP cannot handle fluents and so could not be tested on this domain. Again, Sapa produces invalid plans, but is plotted to give an approximate comparison of time.

Once again, CRIKEY finds plateaux in the search space and this again is the reason why the second version of CRIKEY performs worse since it has a bigger search space to explore at this point.

Figure 5.16(b) shows the quality of the solutions produced, including the optimal quality achievable. Both versions of CRIKEY produce the same solutions. In some problems, this is the optimal solution. This is usually where the problem is highly constrained and the

⁵The proposition `light` no longer takes the match providing it as a parameter. This prevents two `LIGHT_MATCH` actions executing concurrently, as one would delete the "light" from the other when they burnt out. Whilst this may not be what is intended, it does mean that the `FIX_FUSE` action does not need to specify where the light comes from to fix the fuse. Ideally a combination of these two models is needed: two matches could burn at once and not delete each other's light at the end of the action, whilst also not specifying where the light comes from for the fix fuse action. To model this, conditional effects are needed which CRIKEY is unable to handle.

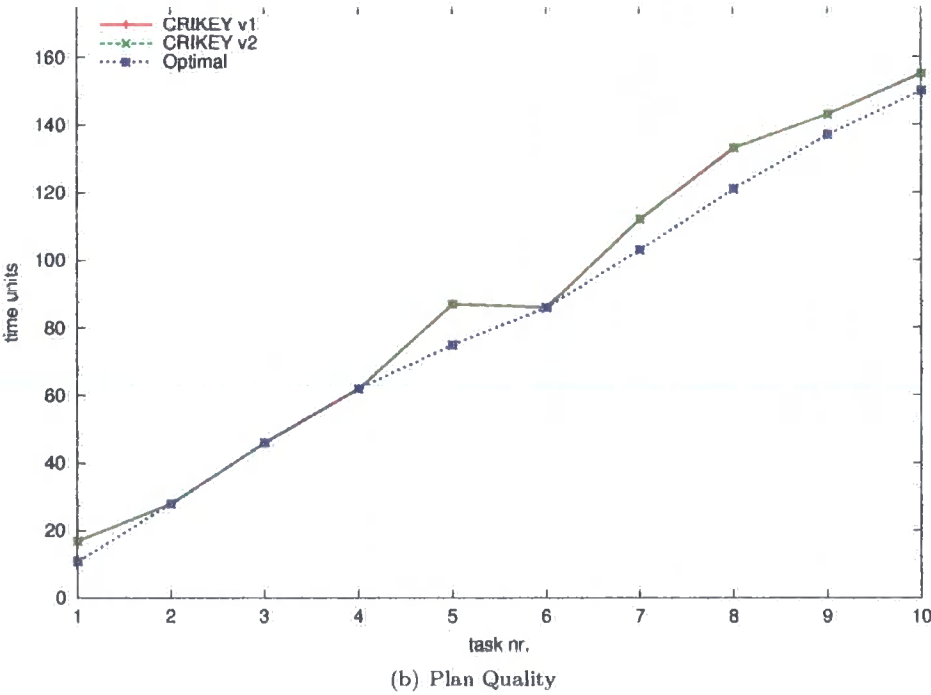
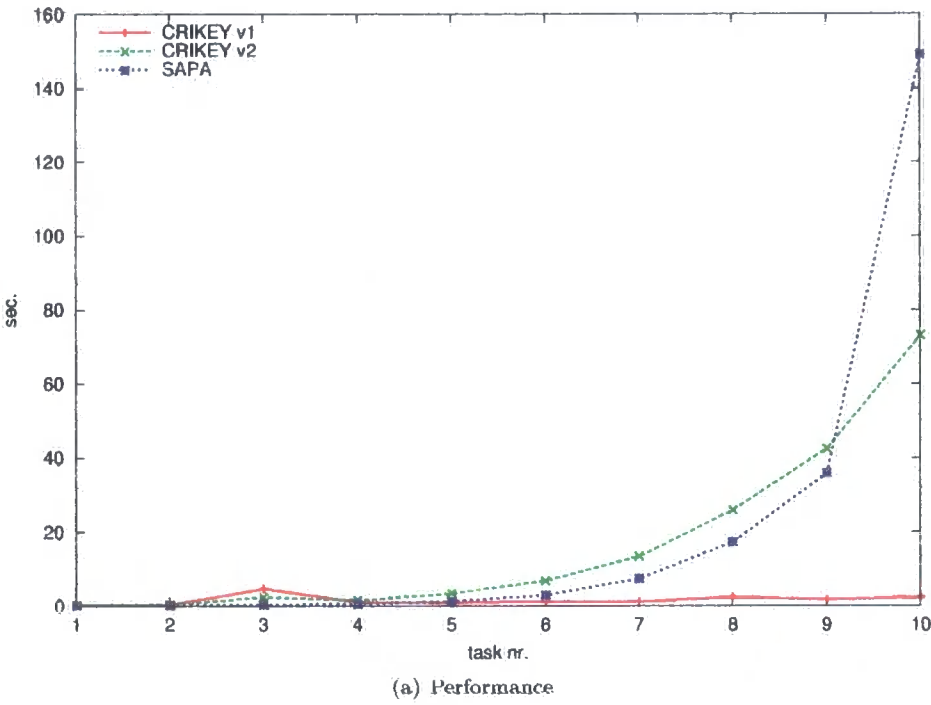


Figure 5.16: Variable Time Match Domain

optimal solution is the *only* solution. (On other problems it may have found the optimal solution purely by accident). In cases where the problem is highly constrained (i.e. some fuses must be fixed by one of the matches), the planner must perform BFS in order to find a solution as EHC fails, since the heuristic ignores the temporal information and pairs the wrong match with the wrong fuse. In these cases, it takes longer to find a solution.

The Lift-Match Domain

So far, the match domains have only contained co-ordination. It is more likely that a “real-life” domain with co-ordination will also have some actions that are not co-ordinated (i.e. need not happen concurrently). The next variant of the match domain (Appendix F) reflects this. As before, electricians must fix fuses by the light of matches. The fuses however, are distributed about rooms in a building which the electricians must navigate around using the corridors and lifts. This navigation is not co-ordinated. Since there is now more than the one electrician, more fuses can be fixed concurrently by the light of one match, so long as the fuses, light and electricians are all present in the same room. Figure 5.17 shows the results from this domain.

This is a much more complex domain and the planners do not fair so well on it. Again, failure occurs most where the problems are highly constrained and there are fewer matches than fuses. In this case, both electricians must be in the same room at the same time to fix fuses by the light of only one match. In the previous match domains there have only been two operators (LIGHT_MATCH and MEND_FUSE), two types (match and fuse) and four predicates (mended, light, handfree and unused). In this domain there are seven operators, six object types and eleven predicates. This makes the search space bigger and so the problems take longer to solve.

As in the previous match domains, the relaxed plan heuristic is of little help since it ignores delete effects, but the LIGHT_MATCH action deleting the light is critical to plan. As a consequence, CRIKEY often fails in EHC and instead must resort to BFS. This is a poor search strategy when the problem is big. A more informed heuristic is needed.

In an attempt to reduce the size of the domain, the match objects are turned into a numeric value where only the number of matches unburnt is recorded. Since all matches are symmetric, this reduces the symmetry in the problem and so the size of the search space. The LIGHT_MATCH action reduces the number of unused matches by one and has a condition that there is at least one match left (see Appendix F.1). Figure 5.18 shows how this reduces the time needed to solve the problems.

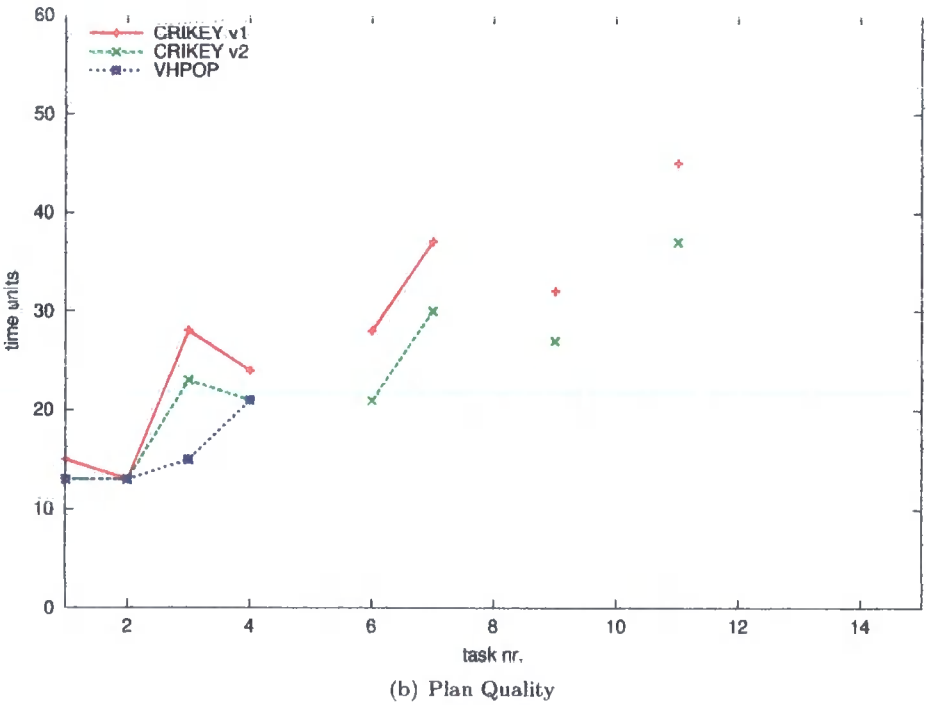
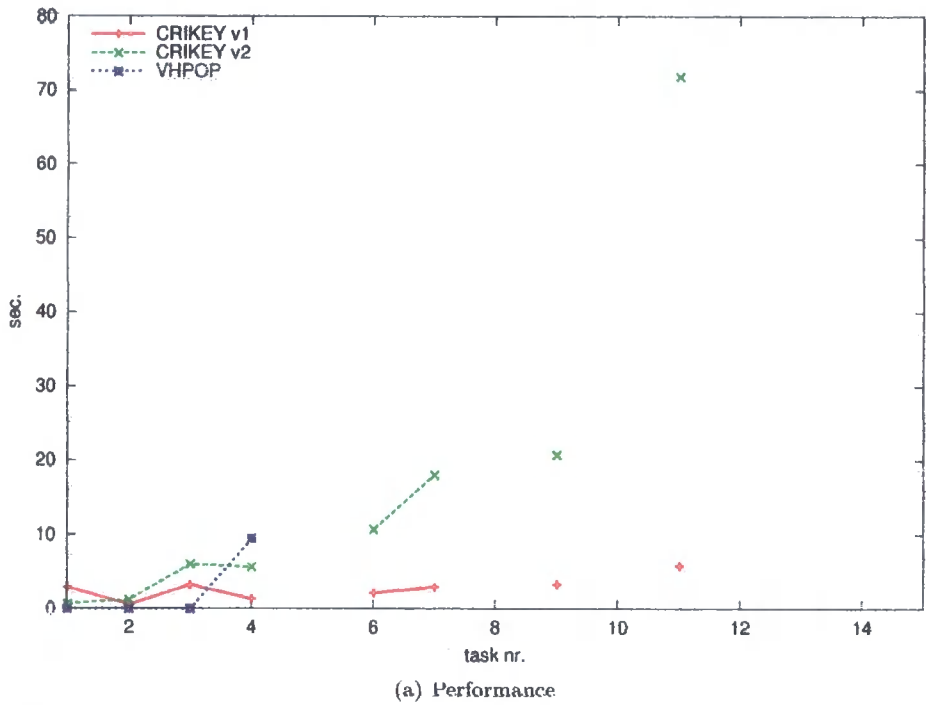


Figure 5.17: The Lift Match Domain

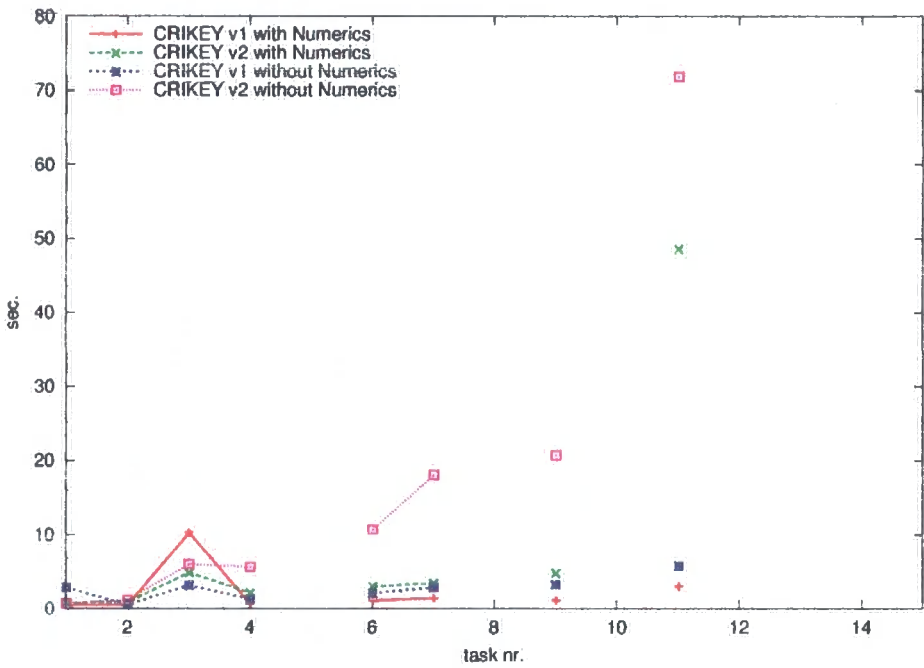


Figure 5.18: Performance of CRIKEY with and without matches encoded using fluents

5.4 DriverLog Shift

The Driver Logistics domain was used at IPC'02 (as used in Appendix A for the example LPGP translation). It involves moving packages around cities using trucks and drivers to transport them. This domain, with the problems used in the competition, has been transformed to the DriverLog Shift domain (Appendix G), where drivers can only work for a certain amount of time before they must take a break and have a rest. This involves co-ordination, as the shift action is an envelope, into which must fit the contents of driving and walking.

Figure 5.19 shows the performance of VHPOP and CRIKEY on the original first fifteen problems of the Simple Time domain. Figure 5.20 shows the performance of the planners on the same problems converted into shift problems. Figure 5.21 shows how the performance of the planner deteriorates. This shows how much harder the problems become once co-ordination is introduced into the problem. VHPOP in particular performs much worse even though it is using the flags that worked best on this domain in IPC'02⁶.

The search must sometimes take arbitrary decisions on which branch on the search tree to explore first where they have the same heuristic value. Problems (e.g. problem 10) where the planner actually performed better on the shift domain is thought to be due to luckily choosing the correct path at this point.

Since the temporal length of the plan is dictated by the shift envelope action, all planners find the same quality of plan, except in Problem 7 where VHPOP finds a plan that can use one less shift.

Table 5.4 shows the proportion of time spent by CRIKEY in the planning and scheduling phases. Again, the second version of CRIKEY spends more of its time proportionally planning, than the first version. This is again due to the increased search space in planning. They are both performing exactly the same task for the scheduling so you would expect this to be equivalent.

Neither the match domain nor the driverlog shift domains can be handled by any of the planners in Table 5.1 that cannot plan with either the single hard envelopes or the complex multiple envelopes. A variation of the driverlog shift domain, originally presented in [13], is where the times of the shift are fixed and cannot not be moved as in the variation presented here. This fixed shift variation can be encoded using PDDL2.2 timed initial literals and so any planner able to handle these could tackle that domain. The variation presented here cannot be represented using timed initial literals since the times of the shifts are not known in the initial state, but are a choice of the planner.

Two more domains are presented in the rest of this section that again involve co-ordination and so again can only be handled by those planners which do not assume a blackbox model of durative action.

⁶VHPOP used grounded actions with A* search, the ADDR heuristic and the MC-Loc-Conf flaw selection criteria.

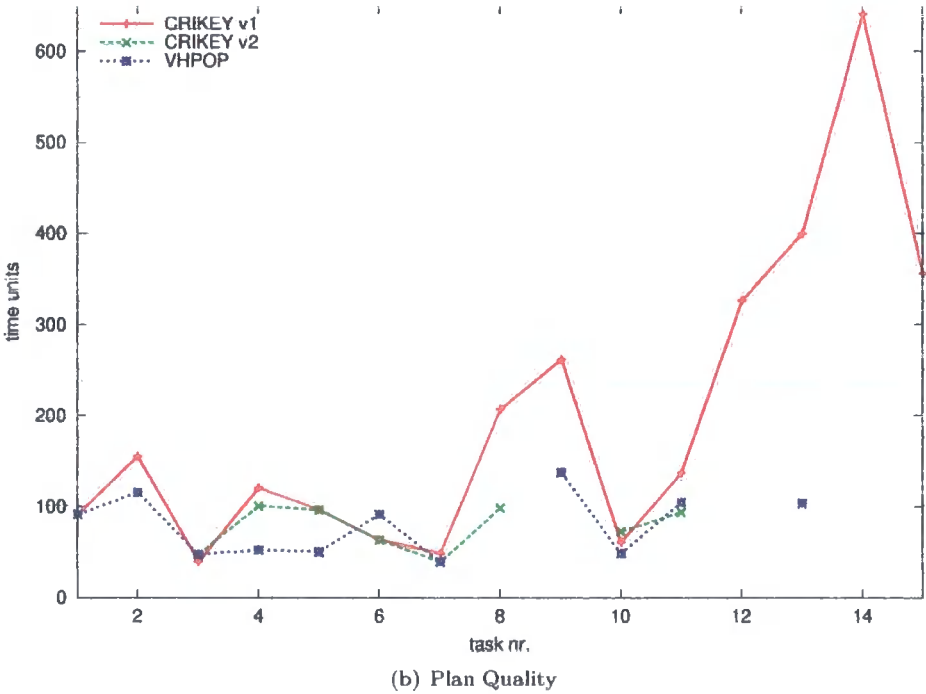
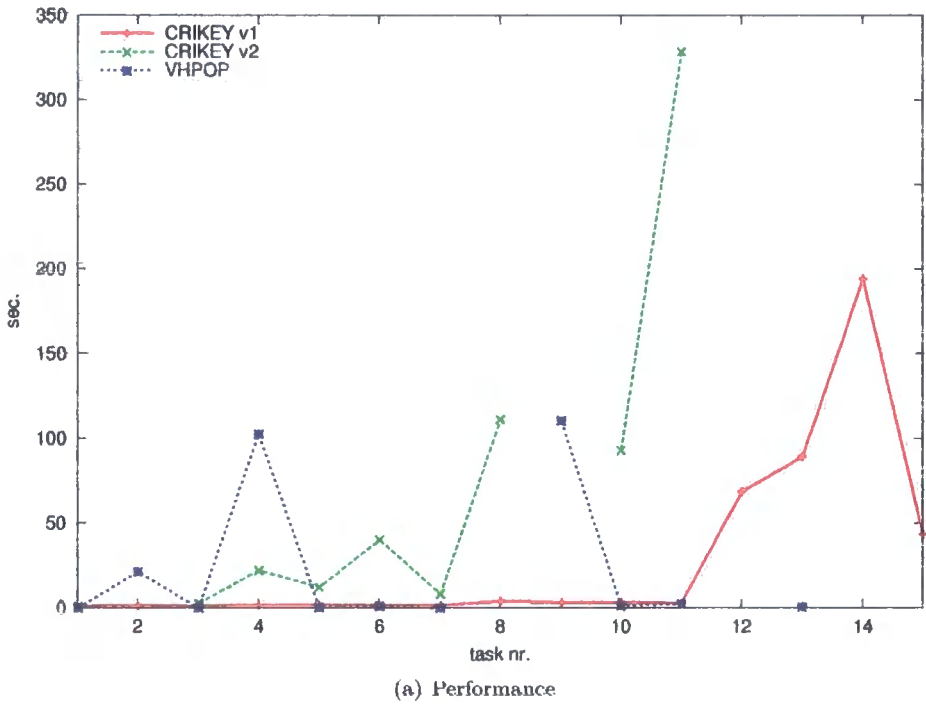


Figure 5.19: Standard DriverLog domain as used in IPC'02

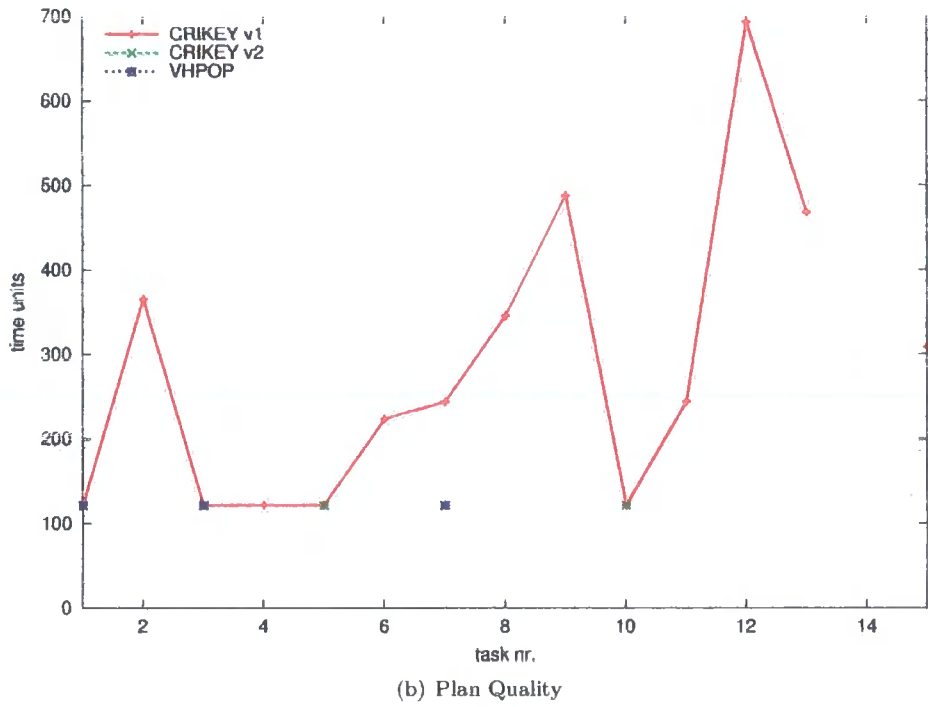
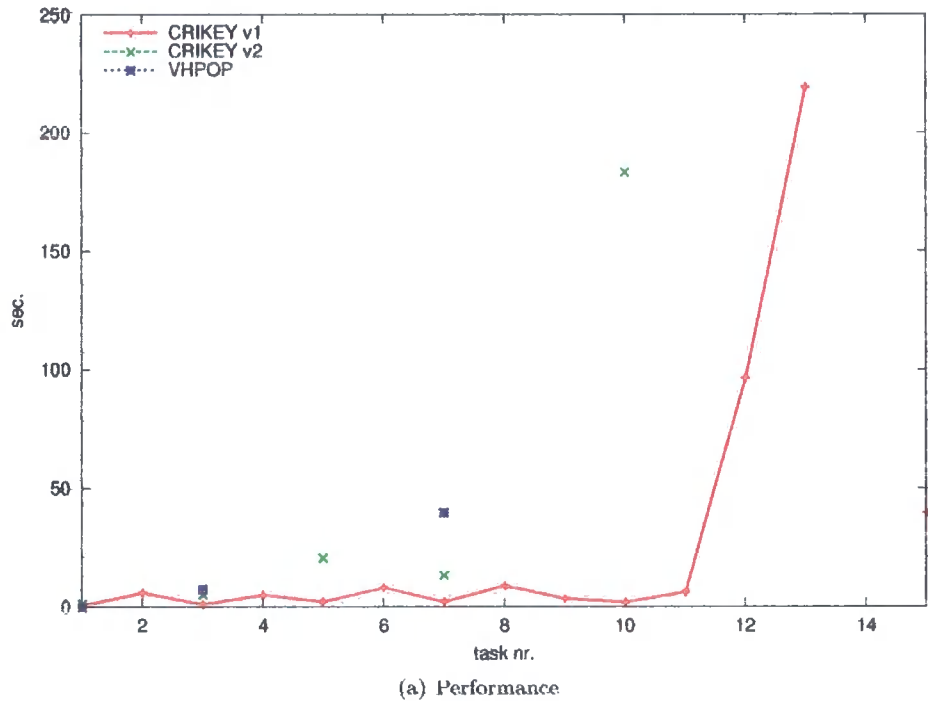


Figure 5.20: DriverLog Simple Time Domain Converted to use Shifts

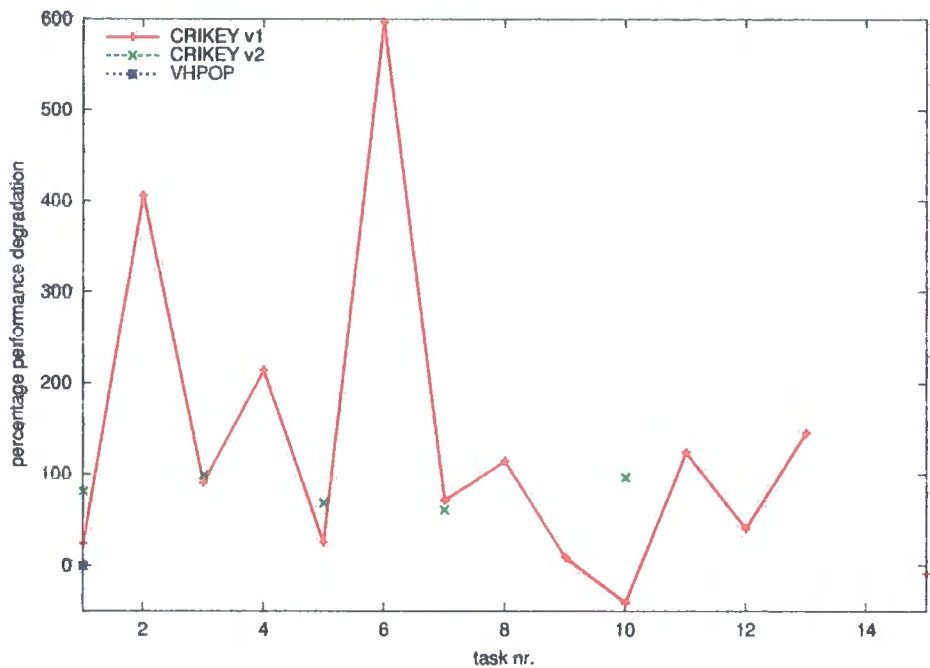


Figure 5.21: Degradation of Performance when DriverLog Domain Converted to use Shifts

5.4.1 Mousetrap

The mousetrap domain (see Appendix H) is inspired by the board game of the same name. Mice must navigate around a maze to find and eat cheese. Travelling down some routes can trigger devices to activate. These devices trigger further devices (in a “cartoon-like” manner) until after some time a trap falls over the cheese along with any mouse eating it. Sometimes it may be possible (or even necessary) to trigger a trap, and then quickly run in, eat the cheese and run away, before the trap falls. Other times this will not be possible, and an alternative route must be found. The co-ordination in this domain occurs between the contraptions and trap actions (these are the envelope), and the mouse’s (content) actions.

It is not possible to encode this using timed initial literals since the timing of the traps falling is dependent on the actions chosen by the planner. It would seem like an ideal usage of the derived predicates also expressible in PDDL2.2. However, these are unable to handle temporal aspects and so not suitable in this case.

Any planner unable to handle domains containing co-ordination, is unable to tackle this domain. It could arise in a “real-world” domain where processes are triggered and the agent must complete some task before the process ends.

Table 5.4: Percentage of Time Spent in Temporal Planning by CRIKEY in the Driverlog Domain

Problem	CRIKEY v1			CRIKEY v2		
	Parsing & Grounding	Planning	Scheduling	Parsing & Grounding	Planning	Scheduling
1	33.33%	33.33%	33.33%	50.00%	50.00%	0.00%
2	14.29%	28.57%	57.14%			
3	25.00%	25.00%	50.00%	5.56%	88.89%	5.56%
4	14.29%	28.57%	57.14%	0.88%	97.37%	1.75%
5	10.00%	40.00%	50.00%	2.08%	93.75%	4.17%
6	12.50%	37.50%	50.00%	0.55%	98.90%	0.55%
7	20.00%	20.00%	60.00%	2.56%	94.87%	2.56%
8	5.26%	42.11%	52.63%	0.16%	99.20%	0.64%
9	3.23%	45.16%	51.61%			
10	15.38%	15.38%	69.23%	1.23%	97.95%	0.82%

5.4.2 Baseball

This domain (in full in Appendix I) models an innings of baseball. The batsmen must travel around all four bases which is dependent on the speed they can run. The duration the ball is in the air is dependent both on how well the batsman can bat and how fast the pitcher is able to throw the ball. The longer the ball is in the air, the more time available for the batsmen to run around the course. Batsmen must reach a base before the ball is retrieved by the fielders. The model deliberately does not allow for one batsman to overtake another.

The co-ordination in this domain is in the running of the players (some actions may happen concurrent, as with two hitters running to different bases, some must occur sequentially, where one hitter runs first to base one and then to base two) and the action that represents the ball travelling through the air.

Again, this domain can only be handled by planners that can reason with co-ordinated actions. i.e. CRIKEY, Sapa, VHPOP and LPGP should (at least in theory) all be able to represent and reason with this domain model.

5.5 Using the Metric

CRIKEY does not hold a queue (or schedule) of exactly when future events happen, allowing it to easily be extended to use Precedence Graphs and handle domains with duration inequalities. It looks at the quality metric given to decide on the duration of actions. As with temporal information, this metric is ignored during the planning phase, so no guarantee of quality can be given.

Few planners automatically consider the metric given (only LPG claims to use it in IPC'04), but this could be because most temporal domains specify minimising the temporal length of the plan. Only MIPS is known to handle duration inequalities (but as previously

observed, cannot handle domains with co-ordination).

The goal in the Café domain (as introduced in Section 4.3.2 and in full in Appendix E) is to deliver breakfast (tea, toast and a cooked breakfast) to tables in a café. The plans are constrained in the number of electrical sockets and chefs available in the kitchen. Two possible metrics for this domain include minimising the heat lost by the breakfast items before they are delivered to the table, and minimising the total time window over which items are delivered to a table.

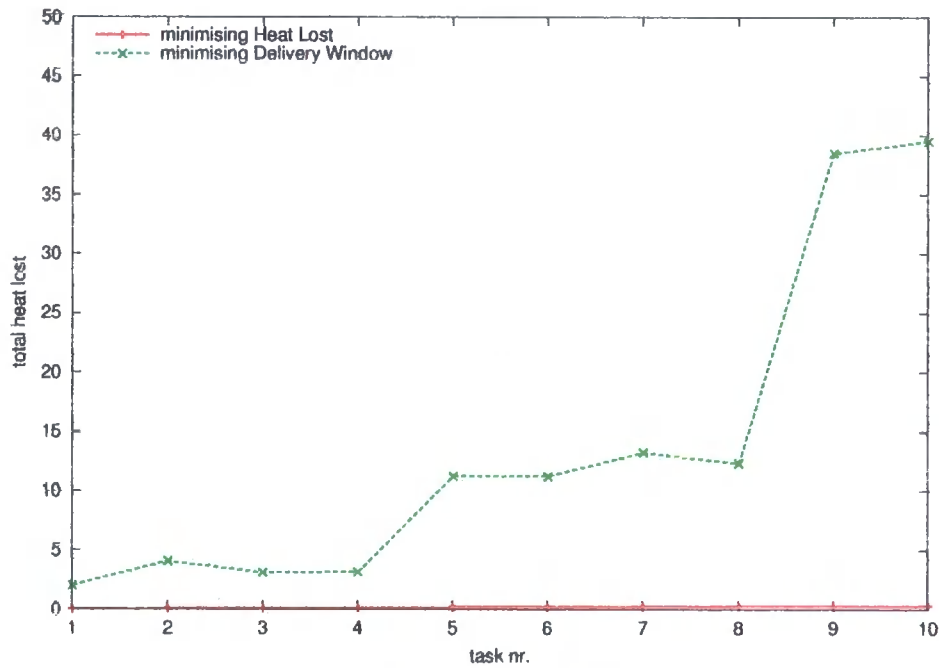
Figure 5.22 shows plan quality with respect to a metric for ten problems in the Café domain. Both graphs show results from exactly the same problems, however in Figure 5.22(a), the metric is set to minimise the heat lost, and in Figure 5.22(b), the metric is set to minimise the delivery window. CRIKEY is trying to minimise the delivery window in the green line, and heat loss in the red line. (Thus the two red lines are for the same plans, and the two green lines for the same plans).

As can be observed, CRIKEY finds a better plan with respect to the metric, when it considers that metric in the scheduling (as should be expected). In each case (for the four lines) the planner produces the same totally ordered actions, but makes different choices when it comes to deciding on the duration of actions where duration inequalities are present.

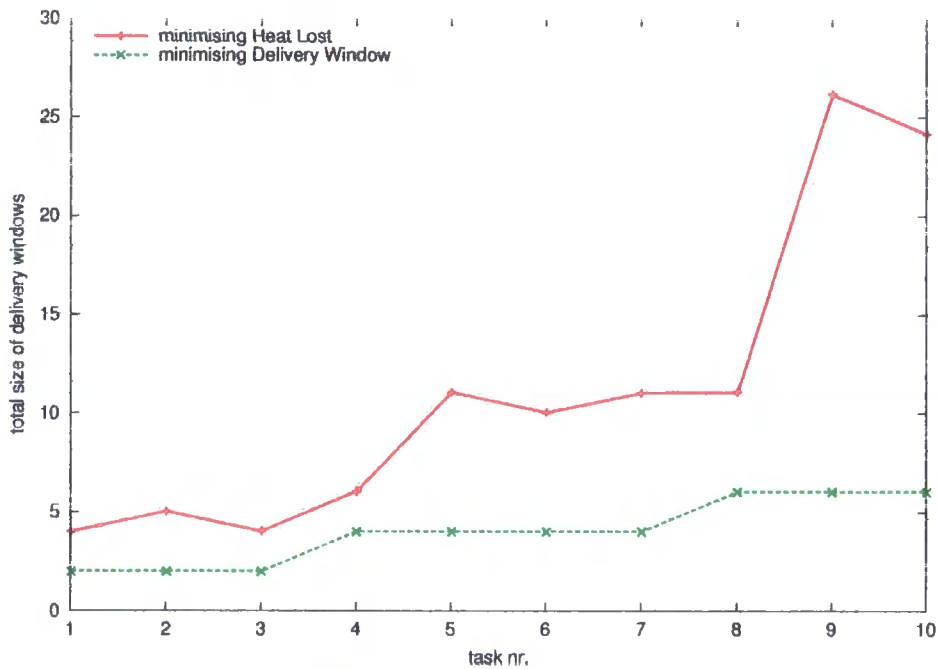
Again, this domain contains temporal constraints, represented using duration inequalities that cannot be encoded using timed initial literals (since heat loss and delivery windows can occur at any time).

5.6 Chapter Summary

CRIKEY is a competitive planner in the benchmark domains that do not contain any co-ordination. Its non-exceptional performance is explained in Section 5.2.1. However, CRIKEY is able to solve problems containing co-ordination that other planners do not do correctly (if at all) and solve them quicker.



(a) Minimising Heat Lost



(b) Minimising Delivery Windows

Figure 5.22: Plan Quality in the Café Domain with CRIKEY version 2

Chapter 6

Conclusions

6.1 Summary

Temporal planning is made up of two components; planning and scheduling. Many temporal planners decompose the problem into these two sub-problems. Where these two sub-problems interact, the separate solvers must communicate and this can be expensive, both in terms of CPU time and memory. Most planners, even if they use a PDDL2.1 model of time, assume “blackbox” durative actions where the internal state of an action is not known. This greatly simplifies how the problems can be coupled and does not permit the modelling of co-ordination. Those temporal planners that do plan with more expressive durative actions resort back to solving both components at once.

This thesis has examined where in temporal planning the planning and scheduling components interact. In cases of co-ordination, not only is the quality of the schedule affected by the plan, but the possibility of finding a schedule at all. This occurs where content actions must execute within the duration of envelope actions. It is in these situations, with both logical and temporal constraints, that the sub-solvers must communicate, and this theory is put to use in a temporal planner called CRIKEY.

CRIKEY does not assume “blackbox” durative actions, but still decomposes the temporal planning problem. Communication between the two sub-solvers is minimised and occurs only where strictly necessary and checking only the part of the plan that contains the co-ordination.

In Chapter 4, CRIKEY is compared to its nearest relative: Sapa. One of the advantages of CRIKEY over Sapa, is the ability to not specify the exact time in the future that effects occur. This allows CRIKEY to handle duration inequalities.

To summarise once again what the contribution to the planning community is, this thesis has an indepth study of the how temporal constraints appear in temporal planning and the nature of them. It is through these and logical constraints that the planning and scheduling interact. A planner was written that uses this theory to minimise the communication be-

tween the planner and scheduler and so solve problems that are not solved by other planners. A novel search state that does not specify the future timings allows for duration inequalities. Through this, the scope, aims, motivation and objectives as set out in Section 1.3 are met.

6.2 Critique of CRIKEY

Whilst CRIKEY performed competitively in IPC'04, it cannot be classed as a leader, either in the performance of the planner or in the quality of the plans it produced. However, CRIKEY obeys the semantics of the competition language and, unlike all other participants in the temporal domains¹, CRIKEY does not make assumptions as to the nature of the problems (i.e. that the problems contain no co-ordination). In assuming “blackbox” durative actions, the other competitors are effectively making the problem easier and so it is no surprise when they perform better than CRIKEY.

There are another two good reasons for the non-exceptional performance by CRIKEY, both are connected with the split of the planning and scheduling. The first is a practical point. CRIKEY was written to explore the interaction between planning and scheduling, and not the two problems themselves. In this, the work has succeeded. However it is reasonable to say that there is not a great amount of scheduling occurring in CRIKEY and that simply lifting a partial order does not qualify as scheduling. This is a valid criticism and could also be extended to the planner as it only performs simple search with a common, yet poor, heuristic.

CRIKEY has a modular architecture, the advantages of which are set out in Section 2.3.4. There is no reason why the planning and scheduling technology aspects cannot be improved. The planner could use a better heuristic and use techniques employed by other planners, such as goal ordering or symmetry detection. The scheduling uses a very simple partial order lifter with the Veloso algorithm. The main problem with this (and other similar polynomial algorithms such as Regnier and Fade algorithm [59]) is that if $a \prec b$ in the total order plan, then in the partial order plan it cannot be the case that $b \prec a$; it must either keep $a \prec b$ or remove it, but it cannot reverse it. As proved in [4], lifting an optimal partial order is a hard problem to solve. Critical Path Analysis (also referred to as PERT scheduling) can go some way to solve this and indeed, this is the approach taken by MIPS. In any case, the theory of where the planning and scheduling interact still holds and can be integrated into any improved technology, such that the planner and scheduler continue to communicate as little as possible. This could all be further work in the development of CRIKEY.

The second reason for the non-exceptional performance of CRIKEY connected with problem decomposition is a theoretical point. Minimising the communication between the planner and scheduler means that they cannot guide each other to good quality solutions where the problems are loosely coupled. The quality of CRIKEY's plans is generally not as

¹with the possible exception of tilSapa. There is little documentation of this system but it is based on Sapa. Whilst Sapa should, in theory, be able to solve such problems, in reality it fails.

good as other planners, since all temporal information is ignored and no scheduling takes place during the planning phase (except in the case of co-ordination, where the plan is checked to make sure that it will be schedulable, but this does not guide the search to a better quality plan). To improve this, communication between the planner and scheduler could be increased, and partially formed plans in the search scheduled to rank them according to their quality. Indeed this is the approach that MIPS takes; It performs a cheap scheduling algorithm on partially built plans, to help guide the search to good quality plans, and then performs the more complex critical path analysis on the final plan.

The heuristic could also be changed to favour plans of better quality. Currently no account is taken of the metric during the search, but the quality of the relaxed plan extracted could be used rather than simply the number of actions in it. A relaxed plan could be drawn from a relaxed temporal planning graph to take some account of temporal aspects in the search (this is the approach taken by Sapa).

Again, this could be further work in the development of CRIKEY. An interesting and useful investigation would be to see how both the quality of the plans and the time taken to find a plan changed relatively to the amount of communication between the sub-solvers. There could be cases where there is great gain on quality with little time lost, and vica versa, there could be cases where the opposite is true. Using this knowledge it could be possible to meet this trade-off with some intelligence.

Another reason for suggesting that there is only limited planning and scheduling taking place is that the temporal planning problems themselves do not contain much planning or scheduling. Benchmark planning problems generally do not contain conflicting goals where the problems are highly constrained and this is reflected in the success of the relaxed planning heuristic and of the planner SPG, both of which rely on this fact.

Temporal planning problems typically contain no, or at best, very little, scheduling. Whilst it is perfectly possible to encode a job shop scheduling problem as a temporal planning problem, it soon becomes apparent that this is not a good method by which to solve the problem (and certainly not with the current technology). This is because planners choose the actions (which in the encoded problem is easy) and do not reason much about their ordering. Not only do the benchmark domains not contain any hard planning problems and little scheduling (as noted several times so far), they do not interact in a hard way either, as is the case in co-ordination.

Currently, as reflected by the planning competition domains, a hard planning problem is equated to a large (in the number of predicates, objects and actions) planning problem. However, I believe that the hardness of a problem should correspond to how constrained the problem is. (Interestingly, problems that become too constrained become easier to solve, since there are fewer choices to be made. In terms of constraint satisfaction problems: too few constraints, and the variables can take any value without impacting on other variables; too many constraints, and the values that the variables take soon propagate through the

rest of the problem resulting in no search being needed). By assuming no hard constraints between the planning and scheduling, the problems become easier.

It is perhaps incorrect to say that CRIKEY completely separates the scheduling from planning, since, as defined at the beginning of this thesis, scheduling is the allocation of resources to actions over time. However, most resource reasoning is performed by the planner, leaving the scheduler to only perform “time” scheduling.

Part of the difficulty arises from how resources are encoded. They are not explicitly modelled, and while there are some good reasons for this (not least because it makes it easier for the domain encoder), this does mean that, say, in planning, processing a job on a machine is seen as a different action for each machine. This should not be the case and the planner should not specify on which machine the job should be processed. The actual identity of the resource is immaterial for the plan, and so also for the action. To realise this automatically is very hard and so restricts how the planning and scheduling can be separated.

The planning community is in an interesting position. On the one hand there has been criticism of the expressive power of PDDL2.1 ([34], [57], [60]), but then on the other hand, there are many assumptions made by people using it, who fail to exploit its full potential. Whilst PDDL2.1 is perfectly capable of modelling many problems (with the notable exception of disjunctive goals that lead to over subscription problems), it may not be the best way of representing such problems. However, domain independent planners should still be able to tackle the full range of problems expressible by a language (and not make assumptions on the input problems) or alternatively limit the language.

CRIKEY addresses this by being the only temporal planner that fully respects the logical semantics of PDDL2.1, by reasoning correctly about start effects, invariants (when achieved by the start effects) and end conditions. Not only does it include those states in its search space that other planners omit, but also reasons intelligently about when and where to check those states for consistency.

Appendix A

Example LPGP Translation

The DriverLog Time Domain as used in IPC'02:

```
(define (domain driverlog)
  (:requirements :durative-actions :fluents)
  (:predicates
    (OBJ ?obj)
    (TRUCK ?truck)
    (LOCATION ?loc)
    (driver ?d)
    (at ?obj ?loc)
    (in ?obj1 ?obj)
    (driving ?d ?v)
    (link ?x ?y)
    (path ?x ?y)
    (empty ?v))

  (:functions
    (time-to-walk ?loc ?loc1)
    (time-to-drive ?loc ?loc1))

  (:durative-action LOAD-TRUCK
    :parameters (?obj ?truck ?loc)
    :duration (= ?duration 2)
    :condition (and
      (at start (OBJ ?obj))
      (at start (TRUCK ?truck))
      (at start (LOCATION ?loc))
      (over all (at ?truck ?loc))
      (at start (at ?obj ?loc)))
    :effect (and
      (at start (not (at ?obj ?loc)))
      (at end (in ?obj ?truck))))

  (:durative-action UNLOAD-TRUCK
    :parameters (?obj ?truck ?loc)
    :duration (= ?duration 2)
    :condition (and
      (at start (OBJ ?obj))
      (at start (TRUCK ?truck))
      (at start (LOCATION ?loc))
      (over all (at ?truck ?loc))
      (at start (in ?obj ?truck)))
    :effect (and
```

```

        (at start (not (in ?obj ?truck)))
        (at end (at ?obj ?loc)))

(:durative-action BOARD-TRUCK
 :parameters (?driver ?truck ?loc)
 :duration (= ?duration 1)
 :condition (and
  (at start (DRIVER ?driver))
  (at start (TRUCK ?truck))
  (at start (LOCATION ?loc))
  (over all (at ?truck ?loc))
  (at start (at ?driver ?loc))
  (at start (empty ?truck)))
 :effect (and
  (at start (not (at ?driver ?loc)))
  (at end (driving ?driver ?truck))
  (at start (not (empty ?truck)))))

(:durative-action DISEMBARK-TRUCK
 :parameters (?driver ?truck ?loc)
 :duration (= ?duration 1)
 :condition (and
  (at start (DRIVER ?driver))
  (at start (TRUCK ?truck))
  (at start (LOCATION ?loc))
  (over all (at ?truck ?loc))
  (at start (driving ?driver ?truck)))
 :effect (and
  (at start (not (driving ?driver ?truck)))
  (at end (at ?driver ?loc))
  (at end (empty ?truck))))

(:durative-action DRIVE-TRUCK
 :parameters (?truck ?loc-from ?loc-to ?driver)
 :duration (= ?duration (time-to-drive ?loc-from ?loc-to))
 :condition (and
  (at start (TRUCK ?truck))
  (at start (LOCATION ?loc-from))
  (at start (LOCATION ?loc-to))
  (at start (DRIVER ?driver))
  (at start (at ?truck ?loc-from))
  (over all (driving ?driver ?truck))
  (at start (link ?loc-from ?loc-to)))
 :effect (and
  (at start (not (at ?truck ?loc-from)))
  (at end (at ?truck ?loc-to))))

(:durative-action WALK
 :parameters (?driver ?loc-from ?loc-to)
 :duration (= ?duration (time-to-walk ?loc-from ?loc-to))
 :condition (and
  (at start (DRIVER ?driver))
  (at start (LOCATION ?loc-from))
  (at start (LOCATION ?loc-to))
  (at start (at ?driver ?loc-from))
  (at start (path ?loc-from ?loc-to)))
 :effect (and
  (at start (not (at ?driver ?loc-from)))
  (at end (at ?driver ?loc-to)))))

```

Problem file 1 for the DriverLog Time Domain as used in IPC'02:

```
(define (problem DLOG-2-2-2)
  (:domain driverlog)
  (:objects
    driver1 driver2
    truck1 truck2
    package1 package2
    s0 s1 s2
    p1-0 p1-2)
  (:init
    (at driver1 s2)
    (DRIVER driver1)
    (at driver2 s2)
    (DRIVER driver2)
    (at truck1 s0)
    (empty truck1)
    (TRUCK truck1)
    (at truck2 s0)
    (empty truck2)
    (TRUCK truck2)
    (at package1 s0)
    (OBJ package1)
    (at package2 s0)
    (OBJ package2)
    (LOCATION s0)
    (LOCATION s1)
    (LOCATION s2)
    (LOCATION p1-0)
    (LOCATION p1-2)
    (path s1 p1-0)
    (path p1-0 s1)
    (path s0 p1-0)
    (path p1-0 s0)
    (= (time-to-walk s1 p1-0) 43)
    (= (time-to-walk p1-0 s1) 43)
    (= (time-to-walk s0 p1-0) 80)
    (= (time-to-walk p1-0 s0) 80)
    (path s1 p1-2)
    (path p1-2 s1)
    (path s2 p1-2)
    (path p1-2 s2)
    (= (time-to-walk s1 p1-2) 29)
    (= (time-to-walk p1-2 s1) 29)
    (= (time-to-walk s2 p1-2) 79)
    (= (time-to-walk p1-2 s2) 79)
    (link s0 s1)
    (link s1 s0)
    (= (time-to-drive s0 s1) 70)
    (= (time-to-drive s1 s0) 70)
    (link s0 s2)
    (link s2 s0)
    (= (time-to-drive s0 s2) 47)
    (= (time-to-drive s2 s0) 47)
    (link s2 s1)
    (link s1 s2)
    (= (time-to-drive s2 s1) 24)
    (= (time-to-drive s1 s2) 24))
  (:goal (and
    (at driver1 s1)
```

```

      (at truck1 s1)
      (at package1 s0)
      (at package2 s0)))

(:metric minimize (total-time)))

```

The domain file after translation:

```

(define (domain driverlog)
  (:requirements )
  (:predicates
    (obj ?obj)
    (truck ?truck)
    (location ?loc)
    (driver ?d)
    (at ?obj ?loc)
    (in ?obj1 ?obj)
    (driving ?d ?v)
    (link ?x ?y)
    (path ?x ?y)
    (empty ?v)
    (load-trucking-inv ?obj ?truck ?loc)
    (iload-trucking-inv ?obj ?truck ?loc)
    (unload-trucking-inv ?obj ?truck ?loc)
    (iunload-trucking-inv ?obj ?truck ?loc)
    (board-trucking-inv ?driver ?truck ?loc)
    (iboard-trucking-inv ?driver ?truck ?loc)
    (disembark-trucking-inv ?driver ?truck ?loc)
    (idisembark-trucking-inv ?driver ?truck ?loc)
    (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver)
    (idrive-trucking-inv ?truck ?loc-from ?loc-to ?driver)
    (walking-inv ?driver ?loc-from ?loc-to)
    (iwalking-inv ?driver ?loc-from ?loc-to))

  (:action load-truck-start
    :parameters (?obj ?truck ?loc)
    :precondition (and
      (obj ?obj)
      (truck ?truck)
      (location ?loc)
      (at ?obj ?loc))
    :effect (and
      (not (at ?obj ?loc))
      (load-trucking-inv ?obj ?truck ?loc)))

  (:action load-truck-inv1
    :parameters (?obj ?truck ?loc)
    :precondition (and
      (at ?truck ?loc)
      (load-trucking-inv ?obj ?truck ?loc))
    :effect (and
      (load-trucking-inv ?obj ?truck ?loc)
      (iload-trucking-inv ?obj ?truck ?loc)))

  (:action load-truck-end
    :parameters (?obj ?truck ?loc)
    :precondition (and
      (load-trucking-inv ?obj ?truck ?loc)
      (iload-trucking-inv ?obj ?truck ?loc))

```

```

      :effect (and
        (in ?obj ?truck)
        (not (load-trucking-inv ?obj ?truck ?loc))
        (not (iload-trucking-inv ?obj ?truck ?loc)))

(:action unload-truck-start
  :parameters (?obj ?truck ?loc)
  :precondition (and
    (obj ?obj)
    (truck ?truck)
    (location ?loc)
    (in ?obj ?truck))
  :effect (and
    (not (in ?obj ?truck))
    (unload-trucking-inv ?obj ?truck ?loc)))

(:action unload-truck-inv1
  :parameters (?obj ?truck ?loc)
  :precondition (and
    (at ?truck ?loc)
    (unload-trucking-inv ?obj ?truck ?loc))
  :effect (and
    (unload-trucking-inv ?obj ?truck ?loc)
    (iunload-trucking-inv ?obj ?truck ?loc)))

(:action unload-truck-end
  :parameters (?obj ?truck ?loc)
  :precondition (and
    (unload-trucking-inv ?obj ?truck ?loc)
    (iunload-trucking-inv ?obj ?truck ?loc))
  :effect (and
    (at ?obj ?loc)
    (not (unload-trucking-inv ?obj ?truck ?loc))
    (not (iunload-trucking-inv ?obj ?truck ?loc))))

(:action board-truck-start
  :parameters (?driver ?truck ?loc)
  :precondition (and (driver ?driver)
    (truck ?truck)
    (location ?loc)
    (at ?driver ?loc)
    (empty ?truck))
  :effect (and
    (not (empty ?truck))
    (not (at ?driver ?loc))
    (board-trucking-inv ?driver ?truck ?loc)))

(:action board-truck-inv1
  :parameters (?driver ?truck ?loc)
  :precondition (and (at ?truck ?loc)
    (board-trucking-inv ?driver ?truck ?loc))
  :effect (and
    (board-trucking-inv ?driver ?truck ?loc)
    (iboard-trucking-inv ?driver ?truck ?loc)))

(:action board-truck-end
  :parameters (?driver ?truck ?loc)
  :precondition (and
    (board-trucking-inv ?driver ?truck ?loc)

```

```

        (iboard-trucking-inv ?driver ?truck ?loc))
:effect (and
        (driving ?driver ?truck)
        (not (board-trucking-inv ?driver ?truck ?loc))
        (not (iboard-trucking-inv ?driver ?truck ?loc))))

(:action disembark-truck-start
 :parameters (?driver ?truck ?loc)
 :precondition (and
        (driver ?driver)
        (truck ?truck)
        (location ?loc)
        (driving ?driver ?truck))
 :effect (and
        (not (driving ?driver ?truck))
        (disembark-trucking-inv ?driver ?truck ?loc)))

(:action disembark-truck-inv1
 :parameters (?driver ?truck ?loc)
 :precondition (and
        (at ?truck ?loc)
        (disembark-trucking-inv ?driver ?truck ?loc))
 :effect (and
        (disembark-trucking-inv ?driver ?truck ?loc)
        (idisembark-trucking-inv ?driver ?truck ?loc)))

(:action disembark-truck-end
 :parameters (?driver ?truck ?loc)
 :precondition (and
        (disembark-trucking-inv ?driver ?truck ?loc)
        (idisembark-trucking-inv ?driver ?truck ?loc))
 :effect (and
        (empty ?truck)
        (at ?driver ?loc)
        (not (disembark-trucking-inv ?driver ?truck ?loc))
        (not (idisembark-trucking-inv ?driver ?truck ?loc))))

(:action drive-truck-start
 :parameters (?truck ?loc-from ?loc-to ?driver)
 :precondition (and
        (truck ?truck)
        (location ?loc-from)
        (location ?loc-to)
        (driver ?driver)
        (at ?truck ?loc-from)
        (link ?loc-from ?loc-to))
 :effect (and
        (not (at ?truck ?loc-from))
        (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver)))

(:action drive-truck-inv1
 :parameters (?truck ?loc-from ?loc-to ?driver)
 :precondition (and
        (driving ?driver ?truck)
        (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver))
 :effect (and
        (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver)
        (idrive-trucking-inv ?truck ?loc-from ?loc-to ?driver)))

```

```

(:action drive-truck-end
  :parameters (?truck ?loc-from ?loc-to ?driver)
  :precondition (and
    (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver)
    (idrive-trucking-inv ?truck ?loc-from ?loc-to ?driver))
  :effect (and
    (at ?truck ?loc-to)
    (not (drive-trucking-inv ?truck ?loc-from ?loc-to ?driver))
    (not (idrive-trucking-inv ?truck ?loc-from ?loc-to ?driver))))

(:action walk-start
  :parameters (?driver ?loc-from ?loc-to)
  :precondition (and
    (driver ?driver)
    (location ?loc-from)
    (location ?loc-to)
    (at ?driver ?loc-from)
    (path ?loc-from ?loc-to))
  :effect (and
    (not (at ?driver ?loc-from))
    (walking-inv ?driver ?loc-from ?loc-to)))

(:action walk-inv1
  :parameters (?driver ?loc-from ?loc-to)
  :precondition (and
    (walking-inv ?driver ?loc-from ?loc-to))
  :effect (and
    (walking-inv ?driver ?loc-from ?loc-to)
    (iwalking-inv ?driver ?loc-from ?loc-to)))

(:action walk-end
  :parameters (?driver ?loc-from ?loc-to)
  :precondition (and
    (walking-inv ?driver ?loc-from ?loc-to)
    (iwalking-inv ?driver ?loc-from ?loc-to))
  :effect (and
    (at ?driver ?loc-to)
    (not (walking-inv ?driver ?loc-from ?loc-to))
    (not (iwalking-inv ?driver ?loc-from ?loc-to))))

```

The durations file created:

```

load-truck = 2
unload-truck = 2
board-truck = 1
disembark-truck = 1
drive-truck s1 s0 = 70
drive-truck s2 s0 = 47
drive-truck s0 s1 = 70
drive-truck s2 s1 = 24
drive-truck s0 s2 = 47
drive-truck s1 s2 = 24
walk p1-0 s0 = 80
walk p1-0 s1 = 43
walk p1-2 s1 = 29
walk p1-2 s2 = 79
walk s0 p1-0 = 80
walk s1 p1-0 = 43
walk s1 p1-2 = 29
walk s2 p1-2 = 79

```

The problem file after the translation:

```
(define (problem dlog-2-2-2)
  (:domain driverlog)
  (:objects
    driver1 driver2
    truck1 truck2
    package1 package2
    s0 s1 s2
    p1-0 p1-2)
  (:init
    (at driver1 s2)
    (driver driver1)
    (at driver2 s2)
    (driver driver2)
    (at truck1 s0)
    (empty truck1)
    (truck truck1)
    (at truck2 s0)
    (empty truck2)
    (truck truck2)
    (at package1 s0)
    (obj package1)
    (at package2 s0)
    (obj package2)
    (location s0)
    (location s1)
    (location s2)
    (location p1-0)
    (location p1-2)
    (path s1 p1-0)
    (path p1-0 s1)
    (path s0 p1-0)
    (path p1-0 s0)
    (path s1 p1-2)
    (path p1-2 s1)
    (path s2 p1-2)
    (path p1-2 s2)
    (link s0 s1)
    (link s1 s0)
    (link s0 s2)
    (link s2 s0)
    (link s2 s1)
    (link s1 s2))
  (:goal (and
    (at driver1 s1)
    (at truck1 s1)
    (at package1 s0)
    (at package2 s0))))
```


Appendix B

The Zeno Travel Domain

The Zeno Travel Time Domain as used in IPC'02:

```
(define (domain zeno-travel)
  (:requirements :durative-actions :typing :fluents)
  (:types aircraft person - locateable city - object)
  (:predicates (in ?p - person ?a - aircraft)
    (at ?x - locateable ?c - city))
  (:functions (fuel ?a - aircraft)
    (distance ?c1 - city ?c2 - city)
    (slow-speed ?a - aircraft)
    (fast-speed ?a - aircraft)
    (slow-burn ?a - aircraft)
    (fast-burn ?a - aircraft)
    (capacity ?a - aircraft)
    (refuel-rate ?a - aircraft)
    (total-fuel-used)
    (boarding-time)
    (debarking-time))

  (:durative-action board
    :parameters (?p - person ?a - aircraft ?c - city)
    :duration (= ?duration (boarding-time))
    :condition (and (at start (at ?p ?c))
      (over all (at ?a ?c)))
    :effect (and (at start (not (at ?p ?c)))
      (at end (in ?p ?a))))

  (:durative-action debark
    :parameters (?p - person ?a - aircraft ?c - city)
    :duration (= ?duration (debarking-time))
    :condition (and (at start (in ?p ?a))
      (over all (at ?a ?c)))
    :effect (and (at start (not (in ?p ?a)))
      (at end (at ?p ?c))))

  (:durative-action fly
    :parameters (?a - aircraft ?c1 ?c2 - city)
    :duration (= ?duration (/ (distance ?c1 ?c2) (slow-speed ?a)))
    :condition (and (at start (at ?a ?c1))
      (at start (>= (fuel ?a)
        (* (distance ?c1 ?c2) (slow-burn ?a)))))
    :effect (and (at start (not (at ?a ?c1)))
      (at end (at ?a ?c2)))
```

```

      (at end (increase total-fuel-used
        (* (distance ?c1 ?c2) (slow-burn ?a))))
      (at end (decrease (fuel ?a)
        (* (distance ?c1 ?c2) (slow-burn ?a)))))

(:durative-action zoom
:parameters (?a - aircraft ?c1 ?c2 - city)
:duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
:condition (and (at start (at ?a ?c1))
  (at start (>= (fuel ?a)
    (* (distance ?c1 ?c2) (fast-burn ?a)))))
:effect (and (at start (not (at ?a ?c1)))
  (at end (at ?a ?c2))
  (at end (increase total-fuel-used
    (* (distance ?c1 ?c2) (fast-burn ?a))))
  (at end (decrease (fuel ?a)
    (* (distance ?c1 ?c2) (fast-burn ?a)))))

(:durative-action refuel
:parameters (?a - aircraft ?c - city)
:duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a)))
:condition (and (at start (> (capacity ?a) (fuel ?a)))
  (over all (at ?a ?c)))
:effect (at end (assign (fuel ?a) (capacity ?a))))

```

An example problem file (problem file 5) taken from IPC'03:

```
(define (problem ZTRAVEL-2-4)
  (:domain zeno-travel)
  (:objects
    plane1 plane2 - aircraft
    person1 person2 person3 person4 - person
    city0 city1 city2 city3 - city)
  (:init
    (at plane1 city1)
    (= (slow-speed plane1) 178)
    (= (fast-speed plane1) 520)
    (= (capacity plane1) 2990)
    (= (fuel plane1) 174)
    (= (slow-burn plane1) 1)
    (= (fast-burn plane1) 3)
    (= (refuel-rate plane1) 1800)
    (at plane2 city2)
    (= (slow-speed plane2) 198)
    (= (fast-speed plane2) 330)
    (= (capacity plane2) 4839)
    (= (fuel plane2) 1617)
    (= (slow-burn plane2) 2)
    (= (fast-burn plane2) 5)
    (= (refuel-rate plane2) 830)
    (at person1 city3)
    (at person2 city0)
    (at person3 city0)
    (at person4 city1)
    (= (distance city0 city0) 0)
    (= (distance city0 city1) 569)
    (= (distance city0 city2) 607)
    (= (distance city0 city3) 754)
    (= (distance city1 city0) 569)
    (= (distance city1 city1) 0)
    (= (distance city1 city2) 504)
    (= (distance city1 city3) 557)
    (= (distance city2 city0) 607)
    (= (distance city2 city1) 504)
    (= (distance city2 city2) 0)
    (= (distance city2 city3) 660)
    (= (distance city3 city0) 754)
    (= (distance city3 city1) 557)
    (= (distance city3 city2) 660)
    (= (distance city3 city3) 0)
    (= (total-fuel-used) 0)
    (= (boarding-time) 0.3)
    (= (debarking-time) 0.6))
  (:goal (and
    (at person1 city2)
    (at person2 city3)
    (at person3 city3)
    (at person4 city3)))
  (:metric minimize (+ (* 1 (total-time)) (* 0.002 (total-fuel-used)))))
```

Appendix C

The Match Domain

The domain:

```
(define (domain matchcellar)
  (:requirements :typing :durative-actions)
  (:types match fuse)
  (:predicates
    (light ?match)
    (handfree)
    (unused ?match - match)
    (mended ?fuse - fuse))

  (:durative-action LIGHT_MATCH
    :parameters (?match - match)
    :duration (= ?duration 8)
    :condition (and
      (at start (unused ?match))
      (over all (light ?match)))
    :effect (and
      (at start (not (unused ?match)))
      (at start (light ?match))
      (at end (not (light ?match)))))

  (:durative-action MEND_FUSE
    :parameters (?fuse - fuse ?match - match)
    :duration (= ?duration 5)
    :condition (and
      (at start (handfree))
      (over all (light ?match)))
    :effect (and
      (at start (not (handfree)))
      (at end (mended ?fuse))
      (at end (handfree)))))
```

A problem instance:

```
(define (problem fixfuse)
  (:domain matchcellar)
  (:objects
    match1 match2 - match
    fuse1 fuse2 - fuse)
  (:init
    (unused match1)
    (unused match2)
    (handfree))
  (:goal (and
    (mended fuse1)
    (mended fuse2)))
  (:metric minimize (total-time)))
```

Appendix D

Alternative Formalisation

Presented here is an alternative formalisation for CRIKEY version 1. In this formalisation the scheduling is done in parallel with the planning, but consistency of the schedule is only checked when and where necessary. Rather than checking whether contents can fit in envelopes at the action applicability stage, the action is added to the state and then the state is considered a “dead-end” in the search if the content action does not fit in the envelope.

Definitions for a STRIPS action (Definition 3.1), durative action (Definition 3.2), single hard envelope (Definition 3.13), compressed action (Definition 4.1) and split action (Definition 4.2) remain the same.

Durative actions are split or compressed as in the formalisation in Chapter 4.

Definition D.1 — Planning State

A planning state S is

$$S = (F, \langle a_1, \dots, a_n \rangle, \mathcal{TC}, \xi)$$

where F is the set of true facts, $\langle a_1, \dots, a_n \rangle$ is the list of split actions so far present in the plan, \mathcal{TC} is the set of temporal constraints between the split actions, and ξ , the set of open envelope durative actions.

Definition D.2 — Applicability of Action

An action $a = (cond, , add, del)$ is applicable in state S if

$$\begin{aligned} & cond \subseteq F \\ \wedge \quad & \forall da \in \xi \cdot del \cap cond_{\hookrightarrow}(da) = \emptyset \end{aligned}$$

Definition D.3 — Result

The result, $Result(s, \langle a \rangle)$, of applying a single STRIPS action $a = (cond, add, del)$ in state $s = (F, \langle a_1, \dots, a_n \rangle, \mathcal{TC}, \xi)$ is $s' = (F', \langle a_1, \dots, a_n, a \rangle, \mathcal{TC}', \xi')$

where

$$\begin{aligned}
 F' &= (F \cup \text{add}) \setminus \text{del} \\
 \xi' &= \xi \cup \{(da(a))\} && \leftarrow a = \vdash \\
 &= \xi \setminus \{(da(a))\} && \leftarrow a = \neg\vdash \\
 &= \xi && \leftarrow \text{otherwise} \\
 \mathcal{FC}' &= \mathcal{FC} \cup \{da(a)_{dur} \leq da(a)_{\neg\vdash} - da(a)_{\vdash} \leq da(a)_{dur}\} \\
 \mathcal{FC}'' &= \mathcal{FC}' \cup \text{veloso}(a, \langle a_1, \dots, a_n \rangle)
 \end{aligned}$$

where $da(a)$ is the corresponding durative action for a and $\text{veloso}(a, \langle a_1, \dots, a_n \rangle)$ returns the temporal constraints found from performing one iteration the Veloso algorithm to see which actions in the list a must follow.

At each stage of the search, a state must not be expanded before it is checked to see if it is a “dead-end”. The consistency is checked from the end of each currently open envelope using a Single Source Shortest Path algorithm. Once the envelope has been closed there is no need to check the consistency. The following definition uses the consistency function (Definition 4.5).

Definition D.4 — Dead end _____

A state s is a dead end (invalid) if

$$\exists e \in \xi \cdot \neg \text{consistent}(\langle a_1, \dots, a_n \rangle, \mathcal{FC})$$

The definition of a planning problem (Definition 4.10) remains the same.

Definition D.5 — Goal State _____

A state $g = (F, \langle a_1, \dots, a_n \rangle, \mathcal{FC}, \xi)$ is a goal state for the problem $P = (O, I, G)$ if

$$F(\text{Result}(I, \langle a_1, \dots, a_n \rangle)) \subseteq G \wedge \xi = \emptyset$$

Appendix E

The Café Domain

The domain:

```
(define (domain CafeDomain)
  (:requirements :typing :fluents :durative-actions :duration-inequalities)
  (:types table chef socket - object tea toast cooked_breaky - item)
  (:predicates
    (delivered ?i - item ?t - table)
    (d_w_available ?t - table)
    (d_w_open ?t - table)
    (ready ?i - item)
    (loosing_heat ?i - item)
    (started_delivery ?i - item)
    (chef_free ?c - chef)
    (socket_free ?s - socket)
    (started_cooking ?i - item))
  (:functions
    (total_delivery_window)
    (total_heat_lost))

  (:durative-action DELIVERY_WINDOW
    :parameters (?t - table)
    :duration (<= ?duration 10000000)
    :condition (and
      (at start (d_w_available ?t)))
    :effect (and
      (at start (not (d_w_available ?t)))
      (at start (d_w_open ?t))
      (at end (not (d_w_open ?t)))
      (at end (increase (total_delivery_window) ?duration))))

  (:durative-action DELIVER
    :parameters (?i - item ?t - table)
    :duration (= ?duration 2)
    :condition (and
      (at end (d_w_open ?t))
      (over all(d_w_open ?t))
      (at start (ready ?i)))
    :effect (and
      (at start (started_delivery ?i))
      (at end (not (started_delivery ?i)))
      (at end (delivered ?i ?t))
      (at end (not (ready ?i)))))
```



```

(:durative-action LOOSING_HEAT
  :parameters (?i - item)
  :duration (<= ?duration 1000)
  :condition (and
    (at start (started_cooking ?i))
    (at end (started_delivery ?i)))
  :effect (and
    (at start (loosing_heat ?i))
    (at end (not (loosing_heat ?i)))
    (at end (increase (total_heat_lost) ?duration))))

(:durative-action MAKE_TEA
  :parameters (?i - tea ?s - socket)
  :duration (= ?duration 1)
  :condition (and
    (at start (socket_free ?s))
    (at end (loosing_heat ?i)))
  :effect (and
    (at start (not (socket_free ?s)))
    (at start (started_cooking ?i))
    (at end (socket_free ?s))
    (at end (ready ?i))))

(:durative-action MAKE_TOAST
  :parameters (?i - toast ?s - socket)
  :duration (= ?duration 2)
  :condition (and
    (at start (socket_free ?s))
    (at end (loosing_heat ?i)))
  :effect (and
    (at start (not (socket_free ?s)))
    (at start (started_cooking ?i))
    (at end (socket_free ?s))
    (at end (ready ?i))))

(:durative-action MAKE_COOKED_BREAKY
  :parameters (?i - cooked_breaky ?c - chef)
  :duration (= ?duration 4)
  :condition (and
    (at start (chef_free ?c))
    (at end (loosing_heat ?i)))
  :effect (and
    (at start (not (chef_free ?c)))
    (at start (started_cooking ?i))
    (at end (chef_free ?c))
    (at end (ready ?i))))

```

A problem:

```
(define (problem CafeProblem1)
  (:domain CafeDomain)
  (:objects
    table1 - table
    tea1 - tea
    toast1 - toast
    chef1 - chef
    socket1 - socket)
  (:init
    (d_w_available table1)
    (chef_free chef1)
    (socket_free socket1)
    (= (total_delivery_window) 0)
    (= (total_heat_lost) 0))
  (:goal (and
    (delivered tea1 table1)
    (delivered toast1 table1)))
  (:metric minimize (total_heat_lost)))
```

An alternative metric could be:

```
(:metric minimize (total_delivery_window))
```

Appendix F

The Lift Match Domain

The domain:

```
(define (domain matchlift)
  (:requirements :durative-actions :typing)
  (:types fuse match lift electrician floor room - object)
  (:predicates
    (light ?match - match ?room - room)
    (handfree ?elec - electrician)
    (unused ?match - match)
    (mended ?fuse - fuse)
    (onfloor ?elec - electrician ?floor - floor)
    (inlift ?elec - electrician ?lift - lift)
    (roomonfloor ?room - room ?floor - floor)
    (liftonfloor ?lift - lift ?floor - floor)
    (inroom ?elec - electrician ?room - room)
    (fuseinroom ?fuse - fuse ?room - room)
    (connectedfloors ?floor1 ?floor2 - floor))

  (:durative-action LIGHT_MATCH
    :parameters (?match - match
                  ?elec - electrician
                  ?room - room)
    :duration (= ?duration 8)
    :condition (and
      (at start (unused ?match))
      (over all (inroom ?elec ?room))
      (over all (light ?match ?room)))
    :effect (and
      (at start (not (unused ?match)))
      (at start (light ?match ?room))
      (at end (not (light ?match ?room)))))

  (:durative-action MEND_FUSE
    :parameters (?fuse - fuse
                  ?match - match
                  ?room - room
                  ?elec - electrician)
    :duration (= ?duration 5)
    :condition (and
      (at start (inroom ?elec ?room))
      (over all (inroom ?elec ?room))
      (at start (fuseinroom ?fuse ?room))
      (at start (handfree ?elec)))
```

```

        (at start (light ?match ?room))
        (over all (light ?match ?room)))
:effect (and
        (at start (not (handfree ?elec)))
        (at end (mended ?fuse))
        (at end (handfree ?elec))))

(:durative-action ENTER_ROOM
 :parameters (?floor - floor
              ?room - room
              ?elec - electrician)
 :duration (= ?duration 1)
 :condition (and
             (at start (onfloor ?elec ?floor))
             (at start (roomonfloor ?room ?floor)))
 :effect (and
         (at end (inroom ?elec ?room))
         (at end (not (onfloor ?elec ?floor)))))

(:durative-action EXIT_ROOM
 :parameters (?floor - floor
              ?room - room
              ?elec - electrician)
 :duration (= ?duration 1)
 :condition (and
             (at start (inroom ?elec ?room))
             (at start (roomonfloor ?room ?floor)))
 :effect (and
         (at end (not (inroom ?elec ?room)))
         (at end (onfloor ?elec ?floor)))))

(:durative-action ENTER_LIFT
 :parameters (?floor - floor
              ?lift - lift
              ?elec - electrician)
 :duration (= ?duration 1)
 :condition (and
             (at start (onfloor ?elec ?floor))
             (at start (liftonfloor ?lift ?floor))
             (over all (liftonfloor ?lift ?floor)))
 :effect (and
         (at end (inlift ?elec ?lift))
         (at end (not (onfloor ?elec ?floor)))))

(:durative-action EXIT_LIFT
 :parameters (?floor - floor
              ?lift - lift
              ?elec - electrician)
 :duration (= ?duration 1)
 :condition (and
             (at start (inlift ?elec ?lift))
             (at start (liftonfloor ?lift ?floor))
             (over all (liftonfloor ?lift ?floor)))
 :effect (and
         (at end (not (inlift ?elec ?lift)))
         (at end (onfloor ?elec ?floor)))))

(:durative-action MOVE_LIFT
 :parameters (?floorfrom ?floorto - floor

```

```

    ?lift - lift)
:duration (= ?duration 2)
:condition (and
  (at start (connectedfloors ?floorfrom ?floorto))
  (at start (liftonfloor ?lift ?floorfrom)))
:effect (and
  (at start (not (liftonfloor ?lift ?floorfrom)))
  (at end (liftonfloor ?lift ?floorto))))

```

Problem 01:

```

(define (problem matchliftproblem01)
  (:domain matchlift)
  (:objects match1 match2 - match
    fuse1 fuse2 - fuse
    lift1 - lift
    elec1 elec2 - electrician
    floor1 floor2 - floor
    room1a room1b room2a room2b - room)
  (:init
    (unused match1)
    (unused match2)
    (handfree elec1)
    (handfree elec2)
    (onfloor elec1 floor1)
    (onfloor elec2 floor1)
    (roomonfloor room1a floor1)
    (roomonfloor room1b floor1)
    (roomonfloor room2a floor2)
    (roomonfloor room2b floor2)
    (liftonfloor lift1 floor1)
    (fuseinroom fuse1 room1a)
    (fuseinroom fuse2 room2b)
    (connectedfloors floor1 floor2)
    (connectedfloors floor2 floor1))
  (:goal (and
    (mended fuse1)
    (mended fuse2)))
  (:metric minimize (total-time)))

```

F.1 Partial Lift Match Numeric Domain

Domain header and LIGHT.MATCH action:

```
(define (domain matchCellarComplexNumeric)
  (:requirements :durative-actions :typing :fluents)
  (:types fuse match lift electrician floor room - object)
  (:predicates
    (light ?room - room)
    (handfree ?elec - electrician)
    (mended ?fuse - fuse)
    (onfloor ?elec - electrician ?floor - floor)
    (inlift ?elec - electrician ?lift - lift)
    (roomonfloor ?room - room ?floor - floor)
    (liftonfloor ?lift - lift ?floor - floor)
    (inroom ?elec - electrician ?room - room)
    (fuseinroom ?fuse - fuse ?room - room)
    (connectedfloors ?floor1 ?floor2 - floor))
  (:functions
    (matchesleft))

  (:durative-action LIGHT-MATCH
    :parameters
      (?elec - electrician
       ?room - room)
    :duration (= ?duration 8)
    :condition (and
      (at start (> (matchesleft) 0))
      (over all (inroom ?elec ?room))
      (over all (light ?room)))
    :effect (and
      (at start (decrease (matchesleft) 1))
      (at start (light ?room))
      (at end (not (light ?room))))))
```

...

Appendix G

DriverLog Shift Domain

The domain:

```
(define (domain driverlogshift)
  (:requirements :typing :durative-actions)
  (:types
    location locatable - object
    driver truck obj - locatable)
  (:predicates
    (at ?obj - locatable ?loc - location)
    (in ?obj1 - obj ?obj2 - truck)
    (driving ?d - driver ?v - truck)
    (link ?x ?y - location)
    (path ?x ?y - location)
    (empty ?v - truck)
    (working ?d - driver)
    (resting ?d - driver)
    (rested ?d - driver)
    (tired ?d - driver))

  (:durative-action WORK
    :parameters
      (?driver - driver)
    :duration (= ?duration 102)
    :condition (and
      (at start (rested ?driver)))
    :effect (and (at start (working ?driver))
      (at end (not (working ?driver)))
      (at start (not (rested ?driver)))
      (at start (not (resting ?driver)))
      (at end (tired ?driver))))

  (:durative-action REST
    :parameters
      (?driver - driver)
    :duration (= ?duration 20)
    :condition (and
      (at start (tired ?driver)))
    :effect (and
      (at start (resting ?driver))
      (at end (not (resting ?driver)))
      (at start (not (working ?driver)))
      (at start (not (tired ?driver)))
      (at end (rested ?driver))))
```

```

(:durative-action LOAD-TRUCK
  :parameters
    (?obj - obj
     ?truck - truck
     ?loc - location)
  :duration (= ?duration 2)
  :condition (and
    (over all (at ?truck ?loc))
    (at start (at ?obj ?loc)))
  :effect (and
    (at start (not (at ?obj ?loc)))
    (at end (in ?obj ?truck))))

(:durative-action UNLOAD-TRUCK
  :parameters
    (?obj - obj
     ?truck - truck
     ?loc - location)
  :duration (= ?duration 2)
  :condition (and
    (over all (at ?truck ?loc))
    (at start (in ?obj ?truck)))
  :effect (and
    (at start (not (in ?obj ?truck)))
    (at end (at ?obj ?loc))))

(:durative-action BOARD-TRUCK
  :parameters
    (?driver - driver
     ?truck - truck
     ?loc - location)
  :duration (= ?duration 1)
  :condition (and
    (over all (at ?truck ?loc))
    (at start (at ?driver ?loc))
    (at start (empty ?truck)))
  :effect (and
    (at start (not (at ?driver ?loc)))
    (at end (driving ?driver ?truck))
    (at start (not (empty ?truck)))))

(:durative-action DISEMBARK-TRUCK
  :parameters
    (?driver - driver
     ?truck - truck
     ?loc - location)
  :duration (= ?duration 1)
  :condition (and
    (over all (at ?truck ?loc))
    (at start (driving ?driver ?truck)))
  :effect (and
    (at start (not (driving ?driver ?truck)))
    (at end (at ?driver ?loc))
    (at end (empty ?truck))))

(:durative-action DRIVE-TRUCK
  :parameters
    (?truck - truck

```



```

        ?loc-from - location
        ?loc-to - location
        ?driver - driver)
:duration (= ?duration 10)
:condition (and
  (at start (at ?truck ?loc-from))
  (over all (driving ?driver ?truck))
  (at start (link ?loc-from ?loc-to))
  (over all (working ?driver)))
:effect (and
  (at start (not (at ?truck ?loc-from)))
  (at end (at ?truck ?loc-to)))

(:durative-action WALK
:parameters
  (?driver - driver
   ?loc-from - location
   ?loc-to - location)
:duration (= ?duration 20)
:condition (and
  (at start (at ?driver ?loc-from))
  (at start (path ?loc-from ?loc-to))
  (over all (working ?driver)))
:effect (and
  (at start (not (at ?driver ?loc-from)))
  (at end (at ?driver ?loc-to))))

```

A problem instance:

```
(define (problem DLOG-2-2-2)
  (:domain driverlog)
  (:objects
    driver1 driver2 - driver
    truck1 truck2 - truck
    package1 package2 - obj
    s0 s1 s2 p1-0 p1-2 - location)
  (:init
    (at driver1 s2)
    (rested driver1)
    (at driver2 s2)
    (rested driver2)
    (at truck1 s0)
    (empty truck1)
    (at truck2 s0)
    (empty truck2)
    (at package1 s0)
    (at package2 s0)
    (path s1 p1-0)
    (path p1-0 s1)
    (path s0 p1-0)
    (path p1-0 s0)
    (path s1 p1-2)
    (path p1-2 s1)
    (path s2 p1-2)
    (path p1-2 s2)
    (link s0 s1)
    (link s1 s0)
    (link s0 s2)
    (link s2 s0)
    (link s2 s1)
    (link s1 s2))
  (:goal (and
    (at driver1 s1)
    (rested driver1)
    (at truck1 s1)
    (at package1 s0)
    (at package2 s0)))
  (:metric minimize (total-time)))
```

Appendix H

Mousetrap Domain

The domain:

```
(define (domain mousetrap)
  (:requirements :durative-actions :typing)
  (:types mouse junction cheese - object
    trap part - contraption
    contraption - object)
  (:predicates
    (at ?m - mouse ?j - junction)
    (cheese_loc ?c - cheese ?j - junction)
    (connected ?j1 ?j2 - junction)
    (trigger_connected ?j1 ?j2 - junction ?p - contraption)
    (eaten ?c - cheese ?m - mouse)
    (clipx)
    (clipy)
    (causes ?p1 ?p2 - contraption)
    (triggered ?p - contraption)
    (trap_up ?t - trap ?c - cheese))
  (:functions
    (contraption_time ?p - contraption)
    (run_time ?j1 ?j2 - junction))

  (:durative-action RUN
    :parameters
      (?m - mouse
       ?from ?to - junction)
    :duration (= ?duration (run_time ?from ?to))
    :condition (and
      (at start (connected ?from ?to))
      (at start (at ?m ?from)))
    :effect (and
      (at start (not (at ?m ?from)))
      (at end (at ?m ?to))))

  (:durative-action RUN_TRIGGER
    :parameters
      (?m - mouse
       ?from ?to - junction
       ?p - contraption)
    :duration (= ?duration (run_time ?from ?to))
    :condition (and
      (at start (trigger_connected ?from ?to ?p))
      (at start (at ?m ?from)))
```

```

        (at end (clipx)))
      :effect (and
        (at start (not (at ?m ?from)))
        (at end (at ?m ?to))
        (at end (triggered ?p))))

(:durative-action CLIP
  :parameters
    ()
  :duration (= ?duration 1)
  :condition (and
    (at start (clipy))
    (at end (clipy)))
  :effect (and
    (at start (clipx))
    (at end (not (clipx)))
    (at start (not (clipy)))))

(:durative-action EAT_CHEESE
  :parameters
    (?m - mouse
     ?j - junction
     ?c - cheese
     ?t - trap)
  :duration (= ?duration 5)
  :condition (and
    (at start (cheese_loc ?c ?j))
    (at start (at ?m ?j))
    (over all (trap_up ?t ?c)))
  :effect (and
    (at end (eaten ?c ?m))))

(:durative-action PERFORM
  :parameters
    (?p1 - part ?p2 - contraption)
  :duration (= ?duration (contraption_time ?p1))
  :condition (and
    (at start (causes ?p1 ?p2))
    (at end (clipx))
    (at start (triggered ?p1)))
  :effect (and
    (at start (clipy))
    (at end (triggered ?p2))))

(:durative-action DROP_TRAP
  :parameters
    (?t - trap
     ?c - cheese)
  :duration (= ?duration (contraption_time ?t))
  :condition (and
    (at start (trap_up ?t ?c))
    (at start (triggered ?t)))
  :effect (and
    (at start (clipy))
    (at end (not (trap_up ?t ?c)))))

```

A problem:

```
(define (problem mousetrapProblem02)
  (:domain mousetrap)
  (:objects
    mouse1 - mouse
    j1 j2 j3 j4 j5 j6 - junction
    cheese1 cheese2 - cheese
    trap1 trap2 - trap
    crank kick_bucket rolling_ball see_saw project_diver - part)
  (:init
    (at mouse1 j1)
    (cheese_loc cheese1 j4)
    (cheese_loc cheese2 j6)
    (trigger_connected j1 j2 crank)
    (trigger_connected j1 j2 crank)
    (trigger_connected j2 j4 see_saw)
    (trigger_connected j4 j2 see_saw)
    (trigger_connected j2 j6 see_saw)
    (trigger_connected j6 j2 see_saw)
    (connected j2 j1)
    (connected j2 j3)
    (connected j3 j2)
    (connected j3 j4)
    (connected j4 j3)
    (connected j5 j4)
    (connected j4 j5)
    (connected j5 j6)
    (connected j6 j5)
    (= (contraption_time crank) 5)
    (= (contraption_time kick_bucket) 2)
    (= (contraption_time rolling_ball) 15)
    (= (contraption_time see_saw) 3)
    (= (contraption_time project_diver) 2)
    (= (contraption_time trap1) 15)
    (= (contraption_time trap2) 4)
    (= (run_time j1 j2) 5)
    (= (run_time j2 j1) 5)
    (= (run_time j2 j3) 10)
    (= (run_time j3 j2) 10)
    (= (run_time j4 j3) 5)
    (= (run_time j3 j4) 5)
    (= (run_time j2 j4) 5)
    (= (run_time j4 j2) 5)
    (= (run_time j4 j5) 10)
    (= (run_time j5 j4) 10)
    (= (run_time j5 j6) 10)
    (= (run_time j6 j5) 10)
    (= (run_time j2 j6) 10)
    (= (run_time j6 j2) 10)
    (causes crank kick_bucket)
    (causes kick_bucket rolling_ball)
    (causes rolling_ball trap1)
    (causes see_saw project_diver)
    (causes project_diver trap2)
    (trap_up trap1 cheese1)
    (trap_up trap2 cheese2)
    (clipy))
  (:goal (and
    (eaten cheese1 mouse1)
    (eaten cheese2 mouse1)))
  (:metric minimize (total-time)))
```

Appendix I

Baseball Domain

The domain:

```
(define (domain baseball)
  (:requirements :durative-actions :typing)
  (:types base runner pitcher)
  (:constants homebase base1 base2 base3 - base)
  (:predicates
    (at ?r - runner ?b - base)
    (free ?b - base)
    (connected ?b1 ?b2 - base)
    (completed ?r - runner)
    (still_to_run ?r - runner)
    (next_pitcher ?p - pitcher)
    (pitcher_order ?p1 ?p2 - pitcher)
    (free_to_pitch)
    (ball_in_air))
  (:functions
    (pitch_speed ?p - pitcher)
    (hit_speed ?r - runner)
    (run_speed ?r - runner))

  (:durative-action RUN
    :parameters
      (?r - runner
       ?from ?to - junction)
    :duration (= ?duration (run_speed ?r))
    :condition (and
      (at start (connected ?from ?to))
      (at start (at ?r ?from))
      (over all (free ?to))
      (over all (ball_in_air)))
    :effect (and
      (at start (not (at ?r ?from)))
      (at start (free ?from))
      (at end (not (free ?to)))
      (at end (at ?r ?to))))

  (:durative-action COMPLETE
    :parameters
      (?r - runner)
    :duration (= ?duration (run_speed ?r))
    :condition (and
      (at start (at ?r base3))
```

```

      (over all (ball_in_air)))
:effect (and
  (at start (not (at ?r base3)))
  (at start (free base3))
  (at end (completed ?r)))

(:durative-action STEP_UP
:parameters
  (?r - runner
   ?p1 ?p2 - pitcher)
:duration (= ?duration 1)
:condition (and
  (at start (free homebase))
  (at start (free_to_pitch))
  (at start (still_to_run ?r))
  (at start (pitcher_order ?p1 ?p2))
  (at start (next_pitcher ?p1)))
:effect (and
  (at end (at ?r homebase))
  (at end (not (still_to_run ?r)))
  (at start (not (free homebase)))
  (at end (not (next_pitcher ?p1)))
  (at end (next_pitcher ?p2)))

(:durative-action HIT
:parameters
  (?r - runner
   ?p - pitcher)
:duration (= ?duration (* 4 (* (hit_speed ?r) (pitch_speed ?p))))
:condition (and
  (at start (at ?r homebase))
  (at start (next_pitcher ?p))
  (at start (free_to_pitch)))
:effect (and
  (at start (ball_in_air))
  (at end (not (ball_in_air)))
  (at start (not (free_to_pitch)))
  (at end (free_to_pitch))))

```

A problem (taken from the Boston RedSox vs. Houston):

```
(define (problem RedSoxVsHouston)
  (:domain baseball)
  (:objects
    Martinez Lowe Wakefield Embree Foulke
    Schilling Leskanic Timlin Mendoza Arroyo - pitcher
    Biggio Vizcaino Ensberg Berkman Kent
    Bagwell Beltran Ausmus Palmeiro Chavez - runner)
  (:init
    (free homebase)
    (free base1)
    (free base2)
    (free base3)
    (connected homebase base1)
    (connected base1 base2)
    (connected base2 base3)
    (pitcher_order Martinez Lowe)
    (pitcher_order Lowe Wakefield)
    (pitcher_order Wakefield Embree)
    (pitcher_order Embree Foulke)
    (pitcher_order Foulke Schilling)
    (pitcher_order Schilling Leskanic)
    (pitcher_order Leskanic Timlin)
    (pitcher_order Timlin Mendoza)
    (pitcher_order Mendoza Arroyo)
    (pitcher_order Arroyo Martinez)
    (still_to_run Biggio)
    (still_to_run Vizcaino)
    (still_to_run Ensberg)
    (still_to_run Berkman)
    (still_to_run Kent)
    (still_to_run Bagwell)
    (still_to_run Beltran)
    (still_to_run Ausmus)
    (still_to_run Palmeiro)
    (still_to_run Chavez)
    (next_pitcher Arroyo)
    (free_to_pitch)
    (= (pitch_speed Martinez) 1.2)
    (= (pitch_speed Lowe) 1.3)
    (= (pitch_speed Wakefield) 1.0)
    (= (pitch_speed Embree) 0.9)
    (= (pitch_speed Foulke) 1.1)
    (= (pitch_speed Schilling) 0.8)
    (= (pitch_speed Leskanic) 1.4)
    (= (pitch_speed Timlin) 1.6)
    (= (pitch_speed Mendoza) 1.2)
    (= (pitch_speed Arroyo) 1.2)
    (= (hit_speed Biggio) 0.9)
    (= (hit_speed Vizcaino) 1.3)
    (= (hit_speed Ensberg) 0.7)
    (= (hit_speed Berkman) 2.1)
    (= (hit_speed Kent) 0.6)
    (= (hit_speed Bagwell) 1.0)
    (= (hit_speed Beltran) 0.9)
    (= (hit_speed Ausmus) 0.9)
    (= (hit_speed Palmeiro) 0.8)
    (= (hit_speed Chavez) 0.7)
    (= (run_speed Biggio) 1.5)
```



```
(= (run_speed Vizcaino) 1.3)
(= (run_speed Ensberg) 0.6)
(= (run_speed Berkman) 0.9)
(= (run_speed Kent) 1.0)
(= (run_speed Bagwell) 1.1)
(= (run_speed Beltran) 1.7)
(= (run_speed Ausmus) 0.8)
(= (run_speed Palmeiro) 0.8)
(= (run_speed Chavez) 0.5))
(:goal (and
  (completed Biggio)
  (completed Vizcaino)
  (completed Ensberg)
  (completed Berkman)
  (completed Kent)
  (completed Bagwell)
  (completed Beltran)
  (completed Ausmus)
  (completed Palmeiro)
  (completed Chavez)))
(:metric minimize (total-time)))
```

Bibliography

- [1] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
- [2] F. Bacchus, 2000. web site of the 2nd International Planning Competition 2000: <http://www.cs.toronto.edu/aips2000/>.
- [3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
- [4] C. Bäckström. Finding least constrained plans and optimal parallel executions is harder than we thought. In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, Vadstena, Sweden, Dec 1994. IOS Press.
- [5] R. Barták. Integrating planning into production scheduling: A formal view. In *Proceedings of the Workshop on Integrating Planning into Scheduling at ICAPS-04*, pages 1–8, June 2004.
- [6] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.
- [7] B. Bonet and H. Geffner. Heuristic search planer 2.0. *AI Magazine*, 22(3):77–80, 2001.
- [8] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-ff. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [9] J. Carlier and É. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annal of Operation Research*, 26:269–287, 1990.
- [10] Y. Chen, C.-W. Hsu, and B. W. Wah. SGPlan: Subgoal Partitioning and Resolution in Planning. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.

- [11] A. Coddington, M. Fox, and D. Long. Handling Durative Actions in Classical Planning Frameworks. In J. Levine, editor, *Proceedings of the 20th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 44–58. University of Edinburgh, December 2001.
- [12] A. Coles and A. Smith. Marvin: Macro actions from reduced versions of the instance. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [13] S. Cresswell and A. Coddington. Planning with Timed Literals and Deadlines. In J. Porteous, editor, *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 22–35. University of Strathclyde, December 2003. ISSN 1368-5708.
- [14] S. Cresswell and A. Coddington. Compilation of LTL goal formulas into PDDL. In *European Conference on Artificial Intelligence (ECAI 2004)*, 2004.
- [15] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. In *Proceedings from Principles of Knowledge Representation and Reasoning*, pages 83–93. Toronto, Canada, 1989.
- [16] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.
- [17] M. B. Do and S. Kambhampati. Sapa: a Domain-Independent Heuristic Metric Temporal Planner. In *Proceedings from the 6th European Conference of Planning (ECP)*, 2001.
- [18] B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, Chicago, USA, June 1994. AAAI Press.
- [19] S. Edelkamp. Taming numbers and duration in the model checking integrated planning system. In *Journal of Artificial Research (JAIR)*, 2002.
- [20] S. Edelkamp and M. Helmert. On the Implementation of Mips. In *Proceedings from the 4th Artificial Intelligence Planning and Scheduling (AIPS), Workshop on Decision-Theoretic Planning*, pages 18–25. Breckenridge, Colorado:AAAI-Press, 2000.
- [21] S. Edelkamp and J. Hoffmann. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical report, Fachbereich Informatik, Germany and Institut für Informatik, Germany, 2003.
- [22] K. Erol, D. Nau, and V. S. Subrahmanian. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artificial Intelligence Journal*, 76(1-2):75–88, July 1995.

- [23] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. of the 2nd IJCAI*, pages 608–620, London, UK, 1971.
- [24] M. Fox and D. Long. The Automatic Inference of State Invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.
- [25] M. Fox and D. Long. HybridSTAN: Identifying and Managing Combinatorial Optimisation Sub- problems in Planning. In *Proceedings of the 12th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 445–452, 2001.
- [26] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Technical report, University of Durham, UK, 2001.
- [27] M. Fox and D. Long. The third International Planning Competition: Temporal and Metric Planning. In *Proceedings from the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, pages 115–117, 2002.
- [28] M. Fox and D. Long. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.
- [29] M. Fox and D. Long. Time in Planning. to be published, Jan 2004.
- [30] M. Fox, D. Long, and K. Halsey. An Investigation into the Expressive Power of PDDL2.1. In *Proceedings of the 16th European Conference of Artificial Intelligence (ECAI)*, 2004.
- [31] M. Fox, D. Long, and M. Hamdi. Handling Multiple Sub-problems within a Planning Domain. In *Proceedings of the 20th UK Planning and Scheduling Special Interest Group (PlanSIG)*, 2001.
- [32] G. Gallo and S. Pallottino. Shortest path algorithms. *Annals of Operations Research*, 13:38–64, 1988.
- [33] A. Garrido, M. Fox, and D. Long. A Temporal Planning System to Manage Level 3 Durative Actions of PDDL2.1. In *Proceedings of the 20th UK Planning and Scheduling Special Interest Group (PlanSIG)*, pages 127–138, 2001.
- [34] H. Geffner. PDDL 2.1: Representation vs. Computation. *Journal of Artificial Intelligence Research*, 20:139–144, 2003.
- [35] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the Limitations of the Situation Calculus? In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Blotsoe*, pages 167–180. Kluwer, London, 1991.

- [36] A. Gerevini and I. Serina. LPG: A Planner based on Local Search for Planning Graphs. In *Proceedings of the 6th International Conference of Artificial Intelligence Planning and Scheduling (AIPS'02)*, Menlo Park, CA, 2002. AAAI Press.
- [37] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998. Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The Planning Domain Definition Language. AIPS-98 Planning Committee.
- [38] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pages 61–67, Menlo Park, CA, 1994. AAAI Press.
- [39] P. Haslum. TP4'04 and HSP*_a. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [40] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning*, 2001.
- [41] M. Helmert and S. Richter. Fast downward – making use of causal dependencies in the problem representation. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [42] J. Hoffmann. Extending FF to Numerical State Variables. In F. V. Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 571–575, Lyon, France, July 2002.
- [43] J. Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, Toulouse, France, April 2002. 379-387.
- [44] J. Hoffmann, S. Edelkamp, R. Englert, F. Liporace, S. Thiébaux, and S. Trüg. Towards realistic Benchmarks for Planning: the Domains used in the Classical Part of IPC-4. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [45] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
- [46] J. Hoffmann and B. Nebel. What makes the difference between HSP and FF? In *Proceedings IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, 2001.

- [47] H. Kautz and B. 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.
- [48] J. Koehler. Solving Complex Planning Tasks Through Extraction of Subproblems. In *Artificial Intelligence Planning Systems*, pages 62–69, 1998.
- [49] T. S. Kumar. Incremental Computation of Resource-Envelopes in Producer-Consumer Models. In *Proceedings of The Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, 2003.
- [50] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003.
- [51] A. L. Lansky. A Representation of Parallel Activity Based on Events, Structure, and Causality. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning about Actions and Plans*, pages 123–159. Kaufmann, Los Altos, CA, 1987.
- [52] D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *Proceedings of AIPS 2000*, pages 196–205, 2000.
- [53] D. Long and M. Fox. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 52–61, 2003.
- [54] D. Long, M. Fox, L. Sebastia, and A. Coddington. An examination of resources in planning. In *Proceedings of the 19th workshop of the U.K. Planning and Scheduling Special Interest Group (PLANSIG)*, 2000.
- [55] J. McCarthy and P. J. Hayes. *Some philosophical problems from the standpoint of artificial intelligence*, pages 26–45. Morgan Kaufmann Publishers Inc., 1987.
- [56] D. McDermott. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35–55, 2000.
- [57] D. McDermott. PDDL2.1 – The Art of the Possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research*, 20:145–148, 2003.
- [58] E. P. D. Pednault. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *KR'89: Proc. of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. Kaufmann, San Mateo, CA, 1989.
- [59] P. Regnier and B. Fade. Complete determination of parallel actions and temporal optimization in linear plans of action. In *Proceedings of the European Workshop on Planning*, pages 100–111. Springer-Verlag, 1991.

- [60] D. Smith. The Case for Durative Actions: A Commentary on PDDL2.1. *Journal of Artificial Intelligence Research*, 20:149–154, 2003.
- [61] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15:61–94, 2000.
- [62] D. Smith and D. S. Weld. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 326–337, 1999.
- [63] B. Srivastava. RealPlan: Decoupling Causal and Resource Reasoning in Planning. In *Proceedings from the Twelfth Innovative Applications of Artificial Intelligence Conference on Artificial Intelligence (IAAI)*, pages 812–818, 2000.
- [64] B. Srivastava and S. Kambhampati. Scaling up planning by teasing out resource scheduling. In *Proceedings of the 5th European Conference on Planning (ECAI-00)*, pages 172–186. Springer-Verlag, 2000.
- [65] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1-3):297–326, 1992.
- [66] P. van Beek and X. Chen. CPlan: A Constraint Programming Approach to Planning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590, 1999.
- [67] M. M. Veloso, M. A. Pérez, and J. G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 207–212, San Diego, CA, November 1990. Morgan Kaufmann.
- [68] V. Vidal. The yahsp planning system: Forward heuristic search with lookahead plans analysis. International Planning Competition 4 Booklet, ICAPS 2004, June 2004. Extended Abstract.
- [69] M. Vilain, H. Kautz, and P. van Beek. *Constraint propagation algorithms for temporal reasoning: a revised report*, pages 373–381. Morgan Kaufmann Publishers Inc., 1990.
- [70] D. S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 4, 1994.
- [71] S. A. Wolfman and D. S. Weld. The LPSAT Engine & Its Application to Resource Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 310–317, 1999.

