Improving Plan Flexibility By Reasoning About Action Orderings and Instantiations

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

Max Victor Luke Waters

Bachelor of Arts, University of Western Australia, Australia
Master of Computing, RMIT University, Australia

School of Computing Technologies
College of Science, Technology, Engineering and Maths
RMIT University

April 2021

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Max Waters
April 27, 2021

Acknowledgement of Country

I acknowledge the Wurundjeri people of the Kulin Nations as the traditional owners of the land on which this research was undertaken, and respectfully recognise Elders both past and present.

For Erin.

Acknowledgements

Thanks to my supervisors, Lin Padgham and Sebastian Sardiña, for their encouragement and guidance, for pointing out the good ideas amongst the bad, and for not gloating when I didn't take good advice. Thanks to Lin for helping me to always keep the reader and the big picture in mind, and thanks to Sebastian for his constant enthusiasm and help in separating sense from nonsense.

Thanks to Bernhard Nebel for the discussions that helped me settle on the topic of this thesis.

Thanks to my family for their endless support for everything that I do, and for their generosity with wine, without which this thesis would not exist.

Thanks to my friends for not getting angry at me for turning up to their birthday/child's birthday/wedding/etc., and staring into space and thinking about treewidth [113] or something.

Thanks to Erin for her endless patience in the face of what was a seemingly interminable task. And thanks for getting me a cat when I needed it.

I'm sure that the irony of a PhD on intelligent planning taking twice as long as expected is not lost on any of you.

Contents

List of Figures

List of Tables

Abstract

To achieve its goals, an agent typically has a *plan*. However, in real-world domains, the environment can be dynamic, partially observable or contain other agents, and actions can fail or have unexpected side effects. Thus, this thesis introduces, theoretically analyses and empirically evaluates novel techniques for representing and generating *flexible* plans that allow the agent to modify its behaviour in response to unexpected events.

While most approaches to plan flexibility are concerned with flexible *action orderings*, often found by *relaxing* a classical plan into an optimally flexible partial-order plan (POP), the focus here is the flexibility that arises from allowing the agent to reason about the *domain objects* used by a plan, that is, the actions' *parameters*. This thesis shows that plans that allow for both domain objects and action orderings to be selected at execution time, and relaxation processes that optimise both domain object use and ordering, provide more execution-time flexibility than those that consider action orderings alone.

To reason about domain objects at *optimisation time*, the notions of *reinstantiated deordering* and *reinstantiated reordering* are introduced, relaxations of a plan under which both ordering constraints and action parameters can be changed. An empirical evaluation shows that these processes can find optimised POPs that are more flexible than is possible when the parameters remain unchanged.

To allow domain objects to be selected at *execution time*, the notion of a *partial plan* is introduced, a generalised plan comprising schematised actions and a *constraint formula* that specifies the allowable combinations of action orderings and parameters. A parameterised complexity analysis shows that finding a classical plan that satisfies these constraints is NP-hard, but polynomial for constraint formulae of bounded *treewidth*. An empirical evaluation shows that MkTR, a novel plan relaxation algorithm, can find partial plans of quadratic "complexity" that are significantly more flexible than POPs found by order optimisation techniques.

Symmetry breaking techniques are introduced to ameliorate the combinatorial explosion that results from optimising both action orderings and parameters. It is shown that a plan can have an exponential number of equally flexible alternatives with symmetrical *causal structures*. As causal structure symmetries are of a form that occurs in many CSP benchmark problems, but is too general for existing symmetry breaking approaches, *Multi Lex* is introduced, a novel and compact symmetry breaking constraint that removes all but a constant factor of generalised symmetries.

CHAPTER 1

Introduction

Since the early days of artificial intelligence research, the ability to achieve goals in complex, unpredictable environments has been considered an essential element of intelligent behaviour. To achieve its goals, an agent typically has a *plan*, that is, a strategy that guides its future actions. Unfortunately, real-world environments make plan execution particularly hard. Actions can fail or have unexpected, nondeterministic side effects, the environment can be dynamic, partially observable or contain other active agents, and it may not be feasible to consider all possibilities ahead of time or to perform complex planning at execution time. It is therefore preferable for a plan to be *flexible*, that is, for it to allow the agent to modify its behaviour in response to unexpected events. Thus, this thesis will introduce, theoretically analyse and empirically evaluate novel techniques for representing and generating plans that provide execution-time flexibility.

While most work on plan flexibility focuses on allowing the agent to decide the *order* in which it executes actions, of particular interest here is the flexibility that arises when the agent is allowed to reason about the *domain objects* that are used in the course of executing a plan. In the context of automated planning and scheduling [48, 51], a domain object is an element of the environment that is relevant to the processes of plan generation and execution. For example, in a factory domain this could include machines, engineers or purchase orders, or consumable items such as raw materials or energy. The central claim of this thesis is that plans and plan generation processes that allow for reasoning about both action orderings and domain object use provide more execution-time flexibility than those that reason about action orderings alone.

The simplest planning model, the well-studied *classical planning* [39, 48, 51], is particularly unsuited to uncertain environments. Classical planners assume that the environment is static and fully observable, and that actions are deterministic, and thus produce *classical plans*, that is, fixed sequences of actions. Nevertheless, there are techniques for generating flexible plans under these assumptions. For example, a *partial-order planner* [135] instead synthesises *partial-order plans* (POPs), a more flexible form of plan that specifies which actions must be executed, but without completely specifying their order. Other approaches repair plans or replan at execution time, or interleave planning, sensing and acting (e.g., [19, 46, 128]). More

complex approaches drop the classical assumptions. If the agent can gather sensory data, then a *contingent planner* (e.g., [64, 92, 102]) or *strong cyclic planner* (e.g., [26, 49, 90, 114]) can generate a conditional or iterative plan, respectively, that selects actions based on the observed environmental state. If not, a *conformant planner* (e.g., [27, 65, 124]) can generate a linear plan that succeeds no matter the effect of nondeterministic actions, or the true state of the unknown aspects of the environment. Unfortunately, this increased flexibility comes at a far greater computational cost than classical planning [60, 82, 111].

Another commonly-used technique, which will be used in this thesis, is to *relax* a classical plan, perhaps one produced by a classical planner, and thereby transform it into a more flexible form (e.g., [8, 33, 71, 91, 116, 120]). There are two benefits to this optimisation approach. Firstly, it can take advantage of recent developments in heuristic state-space planning, one of the most successful current approaches to plan synthesis[1], and secondly, it separates the processes of plan generation and optimisation, and so allows for the optimisation of plans that do not originate from an automated planner, but instead from, for example, a human subject-matter expert.

To effectively optimise a plan's flexibility, a number of interrelated computational challenges must be met. Firstly, a *plan representation* must be found that provides execution-time flexibility. Flexibility is often achieved through the use of a POP, a partially-specified plan comprising a set of actions and a set of required orderings. While representing plans in this way provides ordering flexibility, the domain objects used by the actions, that is, the actions' *parameters*, are fixed, which can limit the agent's execution-time options. For example, in a factory domain, a POP can recover from a late arrival of raw materials by constructing items that do not require that material while awaiting the delivery. However, it cannot recover from the permanent failure of a machine even if a suitable alternative is available.

It is thus preferable for a plan representation to *allow for variation in both a plan's order and the domain objects it uses*. As this requires a more expressive representation than a simple POP, and the design of formal languages typically involves a trade-off between expressiveness and complexity, an important consideration for plan representations that provide this kind of flexibility is *plan tractability*, that is, the computational cost of using the plan. For example, for a plan to be useful at execution time, the agent should be able to quickly determine which of its available actions are compatible with it, and determining a plan's validity, which will be typically be done offline before execution, must be a decidable problem.

The second challenge is the development of a theoretical framework for formally analysing plans expressed in a particular representation. Since the goal of this research is to provide an agent with as flexible a plan as possible, the framework should provide *optimality criteria* that define an ideally flexible plan, and allow for a complexity analysis of the problem of finding plans that satisfy those criteria. From this, algorithms that *generate* optimally flexible plans must be designed and implemented.

Plan representations that allow domain objects to be selected at execution time will require specialised frameworks. However, reasoning about

[1] As evidenced by the International Planning Competition (ipc2018.bitbucket.io/)

domain objects can also improve existing approaches to optimising POP flexibility. Bäckström's [8] seminal work on POP optimisation considers ways to maximise flexibility by minimising ordering constraints. However, these optimisations only modify the actions' orderings, not their parameters, and as the original plan was likely not generated with flexibility in mind, but rather with the aim of minimising makespan or cost, this can limit the flexibility of the final optimised POP. For example, consider a factory with two day-long production tasks that cannot be executed concurrently. A schedule that assigns one task to an engineer who is only available on Monday, and the other to an engineer who is only available on Tuesday, cannot have its order relaxed. However, an optimisation that reassigns both tasks to a third engineer who is available on both days allows a schedule with the same makespan, but with tasks that can now be executed in any order.

It is thus preferable, even when the plan representation does not provide object flexibility, for *optimality criteria to account for different orderings and domain objects, and for both of these to be modified by the optimisation process*. However, this will come at significant computational cost. Finding an optimally flexible POP is well-known to be intractable [8], and considering all combinations of action parameters produces an exponentially larger search space. The final challenge is thus finding algorithms that effectively optimises plans despite this combinatorial explosion.

The following example will illustrate how the intelligent use of domain objects can provide an agent with additional flexibility, informally introduce some of the concepts and terminology that will be used throughout this thesis, and show some computational challenges that arise when reasoning about flexibility.

1.1 Example

Consider a planning task from (an abbreviated version of) the well-known IPC *barman* domain[2], in which a barman must use shot glasses and shakers to prepare cocktails out of various ingredients. In this task, the domain objects comprise two hands (LH and RH), one shot glass (SH), one shaker (SHK), two ingredients (I_1 and I_2) and one cocktail (CT) that is created by adding the two ingredients to the shaker before shaking it. Initially, the shot glass and shaker are both clean, empty and on the table, and both hands are empty. The goal is for the shot glass to contain the cocktail.

Actions have preconditions: shot glasses must be clean before use and hands must be empty to pick up glasses or shakers. Additionally, most actions require two hands: to clean a glass, add an ingredient to a glass, or shake a shaker requires one hand to be holding the glass or shaker, while a second, empty, hand "assists" in cleaning, pouring or shaking, respectively. Plan P_1 below is an optimal solution to this problem:

[2] http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential.html

1. $\mathit{grasp}(\mathsf{RH}, \mathsf{SH})$
2. $\mathit{fill}(\mathsf{SH}, \mathsf{I}_1, \mathsf{RH}, \mathsf{LH})$
3. $\mathit{pourToCleanShaker}(\mathsf{SH}, \mathsf{I}_1, \mathsf{SHK}, \mathsf{RH}, \mathsf{LH})$
4. $\mathit{clean}(\mathsf{SH}, \mathsf{I}_1, \mathsf{RH}, \mathsf{LH})$
5. $\mathit{fill}(\mathsf{SH}, \mathsf{I}_2, \mathsf{RH}, \mathsf{LH})$
6. $\mathit{pourToUsedShaker}(\mathsf{SH}, \mathsf{I}_2, \mathsf{SHK}, \mathsf{RH}, \mathsf{LH})$
7. $\mathit{clean}(\mathsf{SH}, \mathsf{I}_2, \mathsf{RH}, \mathsf{LH})$
8. $\mathit{drop}(\mathsf{RH}, \mathsf{SH})$
9. $\mathit{grasp}(\mathsf{RH}, \mathsf{SHK})$
10. $\mathit{shake}(\mathsf{CT}, \mathsf{I}_1, \mathsf{I}_2, \mathsf{SH}, \mathsf{SHK}, \mathsf{RH}, \mathsf{LH})$
11. $\mathit{pourShakerToShot}(\mathsf{CT}, \mathsf{SH}, \mathsf{RH}, \mathsf{SHK})$

Figure 1.1: Plan P_1.

The barman first picks up the shot glass with its right hand. Then, in steps 2–4, I_1 is poured into the glass, and then from the glass into the shaker, and finally the glass is cleaned. The same is done with ingredient I_2 over steps 5–7. The barman then puts down the shot glass, picks up the shaker, and shakes it to create the cocktail. Finally, the cocktail is poured back into the glass, achieving the goal.

Beyond the problem of generating such a plan, there is the problem of executing it in a dynamic environment. Glasses can break, bottles can be picked up and used by other agents, and the barman could even sprain its robotic wrist. It is therefore desirable for the agent to have alternative ways of achieving its goal.

Reinstantiating plans A common technique for providing flexibility at execution time is to *reorder* the plan. For example, relative to the domain, it does not matter whether ingredient I_1 or I_2 is poured first. Therefore, if I_1 is not available at execution time, the barman could pour I_2 first in the hope that I_1 will soon become available again. Unfortunately, P_1 does not allow for any such reordering of actions.

When reordering a plan, it is essential that the plan remains valid, that is, that it remain legally executable *w.r.t.* the actions' preconditions and achieves the goal. As indicated by the action name, step 6 requires that the shaker not be empty, a condition that is only produced by step 3. Therefore, step 3 *must* be executed before step 6, and as a result I_1 must be poured first. Indeed, the actions in P_1 are so tightly causally linked that P_1 *cannot be reordered at all.*

However, it is possible to pour I_2 first by instead modifying the parameters of some actions, that is, by *reinstantiating* the plan. As with reordering, reinstantiation must preserve plan validity. In this example, modifying the ingredient order while maintaining the plan's validity is a trivial matter of swapping occurrences of I_1 and I_2 in steps 2–7, as has been done in plan P_2 below:

1. *grasp*(RH, SH)
2. *fill*(SH, **I$_2$**, RH, LH)
3. *pourToCleanShaker*(SH, **I$_2$**, SHK, RH, LH)
4. *clean*(SH, **I$_2$**, RH, LH)
5. *fill*(SH, **I$_1$**, RH, LH)
6. *pourToUsedShaker*(SH, **I$_1$**, SHK, RH, LH)
7. *clean*(SH, **I$_1$**, RH, LH)
8. *drop*(LH, SH)
9. *grasp*(LH, SHK)
10. *shake*(CT, I_1, I_2, SH, SHK, RH, LH)
11. *pourShakerToShot*(CT, SH, RH, SHK)

Figure 1.2: Plan P_2: a reinstantiation of P_1.

Plan P_2 is *not* simply a reorder of P_1. It executes the action *types* in the same order as they are executed P_1, but the parameters are different, meaning that it contains different actions. Thus, *reinstantiation can produce valid alternative plans which cannot be found by reordering.*

Reinstantiation allows new reorderings In the example above, the original plan has been transformed into another *ground*, *totally ordered* plan, that is, one in which every action has been assigned a specific execution time and specific parameters. As a single plan cannot provide additional flexibility, and it is not typically feasible to reason about alternative orderings and parameters at execution time, it is preferable for an agent to have a generalised plan that represents a number of different ways to achieve its goal. As POPs represent a set of ground, totally ordered plans that all achieve the same goal by executing the same actions, but differ in their ordering, they are often used for precisely this purpose.

Partial-order plans are often generated by removing unnecessary ordering constraints from a totally ordered plan. While, as discussed above, plan P_1 cannot be generalised in this way, a simple examination of the causal links between P_1's actions reveals a reinstantiation that allows the plan's ordering to be relaxed. A precondition of step 9 is that RH be empty, which is brought about by step 8. However, at the point when step 9 is executed, LH is also empty, and has been since the initial state. Thus, swapping RH and LH over steps 9–11 produces a valid reinstantiation, as shown in plan P_3 below:

1. *grasp*(RH, SH)
2. *fill*(SH, I_1, RH, LH)
3. *pourToCleanShaker*(SH, I_1, SHK, RH, LH)
4. *clean*(SH, I_1, RH, LH)
5. *fill*(SH, I_2, RH, LH)
6. *pourToUsedShaker*(SH, I_2, SHK, RH, LH)
7. *clean*(SH, I_2, RH, LH)
8. *drop*(RH, SH)
9. *grasp*(**LH**, SHK)
10. *shake*(CT, I_1, I_2, SH, SHK, **LH**, **RH**)
11. *pourShakerToShot*(CT, SH, **LH**, SHK)

Figure 1.3: Plan P_3: a reinstantiation of P_1 that can be reordered.

This change in parameters allows P_3 to admit a new reordering. Since step 9 no longer depends on step 8, step 8 need not precede 9. The two steps can be executed in any order so long as they are both executed between steps 7 and 10, and thus P_3 can be relaxed into a POP that represents both of those options. This shows that *an appropriate selection of domain objects can allow new ways of reordering a plan*, and can allow a plan that cannot otherwise be made more flexible to be relaxed into a POP.

Combinations of equivalent objects Providing an agent with the means to reason about, and modify, the domain objects it uses can provide it with more ways to achieve its goals, however, this can come at considerable computational cost. The problem of relaxing a plan into a POP with as few ordering constraints as possible is already known to be intractable [8], and reasoning about domain objects means considering an exponential number of possible action parameters. Plan P_4 below illustrates one way to reduce the effect of this combinatorial explosion.

Plan P_4 is a transformation of P_3 in which LH and RH have been swapped. The validity of this transformation derives from the observation that the domain objects LH and RH are identical in all name: they both represent hands, and are both initially empty. This means that swapping all occurrences of LH and RH in all steps in P_3 will produce a valid plan:

1. $\mathit{grasp}(\mathbf{LH}, \mathsf{SH})$
2. $\mathit{fill}(\mathsf{SH}, \mathsf{I}_1, \mathbf{LH}, \mathbf{RH})$
3. $\mathit{pourToCleanShaker}(\mathsf{SH}, \mathsf{I}_1, \mathsf{SHK}, \mathbf{LH}, \mathbf{RH})$
4. $\mathit{clean}(\mathsf{SH}, \mathsf{I}_1, \mathbf{LH}, \mathbf{RH})$
5. $\mathit{fill}(\mathsf{SH}, \mathsf{I}_2, \mathbf{LH}, \mathbf{RH})$
6. $\mathit{pourToUsedShaker}(\mathsf{SH}, \mathsf{I}_2, \mathsf{SHK}, \mathbf{LH}, \mathbf{RH})$
7. $\mathit{clean}(\mathsf{SH}, \mathsf{I}_2, \mathbf{LH}, \mathbf{RH})$
8. $\mathit{drop}(\mathbf{LH}, \mathsf{SH})$
9. $\mathit{grasp}(\mathbf{RH}, \mathsf{SHK})$
10. $\mathit{shake}(\mathsf{CT}, \mathsf{I}_1, \mathsf{I}_2, \mathsf{SH}, \mathsf{SHK}, \mathbf{RH}, \mathbf{LH})$
11. $\mathit{pourShakerToShot}(\mathsf{CT}, \mathsf{SH}, \mathbf{RH}, \mathsf{SHK})$

Figure 1.4: Plan P_4: a symmetric variation of plan P_3.

This equivalence extends to the plans' flexibility. As in P_3, steps 8 and 9 in P_4 can be executed in any order so long as they are both executed between steps 7 and 10. Thus, P_4 can also be relaxed into a POP representing two possible ways of ordering the actions. As there can, in general, be multiple combinations of equivalent objects, any permutation of which is *guaranteed* to yield an equally valid and flexible reinstantiation, a plan can have a possibly exponential number of variations that need not be considered by the relaxation process. Detecting and avoiding these equivalent set of objects, through the well-studied process of *symmetry breaking*, can make the problem of finding optimally flexible plans exponentially easier.

Relaxing action orderings and parameters Once the possibility of changing the domain objects used by a plan has been acknowledged, a natural next step is to provide further flexibility by *allowing the agent to select domain objects at execution time*. Of course, a POP cannot provide

this type of flexibility. While it allows decisions regarding the order of actions to be made at execution time, the actions' parameters are fixed.

However, plan P_1 can be relaxed into a form that provides this kind of flexibility by replacing its actions' parameters with variables, and supplying rules that describe the allowable combinations of variable bindings and orderings, that is, by transforming it into a *partial plan*. In this way, *a single partial plan can represent a set of ground plans that achieve the same goal, but do so by making use of different domain objects and action orderings.*

For example, a partial plan can be defined which encapsulates all of the transformations described above. In plan P_4 below, the actions' parameters have been selectively replaced by variables. The hands in steps 1–8 and 9–11 have been replaced by the variables h_1 and h_2, and h_3 and h_4, respectively, and all occurrences of I_1 and I_2 have been replaced by the variables i_1 and i_2, respectively:

1. $grasp(h_1, \mathsf{SH})$
2. $fill(\mathsf{SH}, i_1, h_1, h_2)$
3. $pourToCleanShaker(\mathsf{SH}, i_1, \mathsf{SHK}, h_1, h_2)$
4. $clean(\mathsf{SH}, i_1, h_1, h_2)$
5. $fill(\mathsf{SH}, i_2, h_1, h_2)$
6. $pourToUsedShaker(\mathsf{SH}, i_2, \mathsf{SHK}, h_1, h_2)$
7. $clean(\mathsf{SH}, i_2, h_1, h_2)$
8. $drop(h_1, \mathsf{SH})$
9. $grasp(h_3, \mathsf{SHK})$
10. $shake(\mathsf{CT}, \mathsf{I}_1, \mathsf{I}_2, \mathsf{SH}, \mathsf{SHK}, h_3, h_4)$
11. $pourShakerToShot(\mathsf{CT}, \mathsf{SH}, h_3, \mathsf{SHK})$

Figure 1.5: Plan P_5: allows domain objects to be chosen at execution time.

The ordering and binding rules can be informally defined as follows:

1. $i_1 = \mathsf{I}_1$ and $i_2 = \mathsf{I}_2$, or $i_1 = \mathsf{I}_2$ and $i_2 = \mathsf{I}_1$.
2. $h_1 \neq h_2$ and $h_3 \neq h_4$.
3. Steps 1–8 and 10–11 must be executed in order.
4. Steps 9 and 10 must be preceded by 8 and must precede 11.
5. If $h_1 = h_3$ then step 8 must precede step 9.

The first constraint ensures that the cocktail is mixed from the two required ingredients, but allows them to be poured in any order. The next constraint allows the barman to use any combination of different hands. The final three constraints specify the required orderings, and how the optional orderings are affected by the selected domain objects.

Partial plans come with their own computational difficulties. Generalised plans, such as partial-order or partial plans, compactly represent a set of different ways of achieving the same goal, and so for such a plan to be useful at execution time, an agent must be able to quickly extract an element from this set. For a POP, this is a simple matter of finding a sequence of actions that satisfies its ordering constraints, well-known to be a linear-time problem. However, one can imagine a partial plan with rules

encoding a complex combinatorial problem that cannot be solved in a time that is suitable for real-time decision-making. The challenge with partial plans is finding a set of rules that provide as much flexibility as possible while being simple enough to be useful at execution time.

1.2 Research Questions and Contributions

This thesis addresses the following questions regarding flexible plan representation, reasoning about action orderings and domain objects at both optimisation and execution time, and handling the resulting combinatorial explosion:

1. What form must a plan representation take in order to *(i)* provide execution-time flexibility by partially specifying action orderings and domain objects, and *(ii)* allow a ground plan that satisfies that partial specification to be found in a time that is suitable for real-time decision-making?

2. How can a theoretical framework for the formal analysis of plans *(i)* define criteria for an optimally flexible plan that accounts for both action and domain object flexibility, and *(ii)* allow a complexity analysis of the problem of finding a plan that satisfies those criteria?

3. To what degree does the tractability requirement of plans that partially specify action orderings and domain objects limit the flexibility they can provide?

4. What is the difference in flexibility between plans optimised by processes that modify both action orderings and domain objects, and those optimised by processes that modify orderings alone, given the same resource constraints?

5. How can a symmetry breaking constraint *(i)* detect both symmetrical action orderings and domain objects, and *(ii)* efficiently and effectively remove non-canonical plans from the search space?

The contributions of this thesis towards answering these questions can be summarised follows:

Chapter 3: Breaking generalised symmetries in matrix and graph models As the example above demonstrated, the plan optimisation problems studied in this thesis frequently display *symmetry*, that is, they have exponential numbers of (non) solutions that are equivalent up to some irrelevant permutation of domain objects and/or action orderings. Thus, before these problems are addressed directly, Chapter 3 addresses the important problem of identifying symmetries and removing them from the search space.

The plan optimisation techniques presented in Chapters 4 and 5 make a plan more flexible by modifying or relaxing its *causal structure*, that is, a matrix that maps action effects to preconditions, determines the plan's validity and variable bindings, and partially determines its order. Existing

techniques for breaking symmetries in matrices can handle the symmetries that occur when individual rows or individual columns are interchangeable. However, *symmetries over causal structures typically permute multiple rows and/or columns simultaneously.*

Thus, Chapter 3 formally defines a hierarchy of generalised symmetries over matrices and graphs, and presents a series of examples to show that they can occur in many CSP benchmark domains as well as the plan optimisation problems studied in later chapters. After a demonstration that these symmetries are too complex to be broken with standard techniques, *Multi Lex* is introduced: a lexicographic constraint that can break generalised symmetries in matrices and arbitrary graphs, including directed graphs with self-loops. An empirical evaluation shows Multi Lex to be a compact and efficient symmetry breaking constraint that removes all but a constant factor of symmetries from the search space.

Chapter 4: Optimising partial-order plans by modifying orderings and variable bindings One way to achieve execution-time flexibility is to post-process a ground, totally ordered plan into a minimally constrained POP through the well-studied processes of *plan deordering* and *plan reordering*. A drawback of these techniques is that they optimise the actions' orderings without modifying their parameters, and since the original parameters were likely not selected with flexibility in mind, remaining committed to them can unnecessarily restrict the ordering options.

Thus, Chapter 4 introduces, and formally defines, the notions of *reinstantiated deordering* and *reinstantiated reordering*, transformations of a plan under which both ordering constraints and parameters can be changed. The aim is to find an optimised POP with *fewer ordering constraints than would be possible had the parameters remained unchanged.*

A technique is presented for encoding the problem of finding a minimum reinstantiated de/reorder of a POP into an instance of the partial weighted MaxSAT problem. This encoding is an extension and optimisation of the approach introduced by Muise et al. [91].

Symmetry breaking techniques are used to counter the exponential increase in search space that results from allowing parameters to change. The notion of a *causal structure symmetry* is introduced, a symmetry resulting from equivalent combinations of (potential) causal links between action preconditions and effects. To detect these symmetries, the idea of a *plan description graph* is defined, an undirected coloured graph with automorphisms that correspond to the POP's symmetries. The symmetries are broken by extending the MaxSAT encoding with the Multi Lex constraint as introduced in Chapter 3.

An empirical evaluation over all previous IPC STRIPS domains shows that reinstantiation finds POPs that are significantly more flexible, and while symmetry breaking accelerates this search, the additional flexibility still comes at significant additional computational cost.

Chapter 5: Finding tractable and flexible partial plans While the techniques introduced in Chapter 4 allow the plan's domain objects to be modified at *optimisation time*, Chapter 5 is concerned with relaxing plans

into a form that allow domain objects to be selected at *execution time*. It thus introduces, and formally defines, the notion of a *partial plan*, a generalised plan that specifies which action *types* must be executed and provides a *constraint formula* that specifies the allowable combinations of orderings and action parameters.

A complexity analysis shows that finding a classical plan that satisfies a constraint formula is NP-hard. However, further analysis from a parameterised complexity perspective shows that when limited to constraint formulae of bounded *treewidth* (broadly, a measurement of the degree of cyclicity of the formula's underlying graph), the problem is polynomial.

Key to providing execution-time flexibility is the idea of a *minimal k-treewidth relaxation*, that is, a partial plan that cannot be made more flexible without its runtime cost exceeding some predetermined complexity. Further parameterised complexity analysis shows that finding such a plan is fixed-parameter tractable when the search is limited to a particular class of partial plans that represent constraints as sets of allowable causal links.

A practical technique is introduced for finding minimal k-treewidth relaxations within this class: MkTR is a greedy, policy-driven fpt-algorithm that relaxes a partial plan by iteratively expanding the set of allowable causal links while ensuring its treewidth stays below k. To prevent MkTR from exploring multiple symmetrical sets of causal links, policies are introduced that dynamically break causal structure symmetries using the Multi Lex constraint as introduced in Chapter 3. An empirical evaluation over all previous IPC STRIPS domains shows that overall, MkTR can find a partial plan of quadratic "complexity" that is significantly more flexible than POPs found by standard deordering and reordering techniques.

CHAPTER 2

Background and Literature Review

This chapter will first introduce the background concepts and notation that will be used throughout the rest of the thesis, and then present an overview of the existing literature in least-commitment planning, plan deordering and reordering, and symmetry breaking in matrix models, graph models and plan synthesis.

2.1 Background

2.1.1 Logical Preliminaries

The logical structures in this thesis are expressed in a standard first-order language $\mathcal{L}$ with finitely many variable, predicate and function symbols, and the usual logical connectives, including equality and precedence [18, Chapter 2]. This section will not give a full description of the syntax and semantics of first-order logic, but will give a brief reminder of some common notational conventions. The letters x, y and z will indicate variables, c, d and e will indicate constants and t terms (all possibly with annotations). For any logical structure η, the notation $\mathsf{vars}(\eta)$ and $\mathsf{consts}(\eta)$ denote the variables and constants, respectively, appearing in η.

Lists and matrices The notation $\vec{t}$ is used to denote ordered lists of possibly non-unique terms, with $\vec{t}[i]$ indicating the i-th element of the list. The notation $\vec{t_1} = \vec{t_2}$ is used as shorthand for $|\vec{t_1}| = |\vec{t_2}| \wedge \vec{t_1}[1] = \vec{t_2}[1] \wedge \cdots \wedge \vec{t_1}[|\vec{t_1}|] = \vec{t_2}[|\vec{t_2}|]$. Similarly, $\vec{t_1} \neq \vec{t_2}$ is shorthand for $|\vec{t_1}| \neq |\vec{t_2}| \vee \vec{t_1}[1] \neq \vec{t_2}[1] \vee \cdots \vee \vec{t_1}[|\vec{t_1}|] \neq \vec{t_2}[|\vec{t_2}|]$. The list resulting from the concatenation of two lists $\vec{t_1}$ and $\vec{t_2}$ is denoted $\vec{t_1}^\frown\vec{t_2}$.

The notation $\mathbf{M}$ indicates a matrix, i.e., rectangular array of possibly non-unique terms. The notation $\mathbf{M}[i]$, $\mathbf{M}[][j]$ and $\mathbf{M}[i][j]$ indicates the ith row, the jth column, and the jth element of the ith row of matrix $\mathbf{M}$, respectively.

Substitutions The concept of a *substitution* – a mapping from variables to terms – will be used throughout. The notation $\theta = \{x_1 \backslash t_1, \ldots, x_n \backslash t_n\}$

describes a substitution mapping each variable x_i to term t_i, for $1 \leq i \leq n$, and every other variable to itself. The set of variables explicitly mapped by substitution θ is denoted $\mathsf{domain}(\theta)$. The notation $\theta(x)$ is used to denote the term corresponding to variable x under substitution θ, and $\theta(\vec{x})$ is its generalisation to lists of variables. The result of applying a substitution θ to a formula ϕ is written $\phi\theta$, and means to simultaneously replace every variable x in ϕ with $\theta(x)$. A substitution θ is *ground* if every variable in its domain is mapped to a ground term, and is *complete w.r.t.* formula ϕ if $\mathsf{vars}(\phi) \subseteq \mathsf{domain}(\theta)$.

2.1.2 Classical Planning Formalism

Classical planning is the problem of synthesising a sequence of actions that achieves a desired goal from a given initial state. All classical planning tasks will be expressed in a standard first-order STRIPS formalism [80].

2.1.2.1 Classical Planning Tasks

A *classical planning task* is a tuple $\Pi = \langle \mathcal{C}, \mathcal{O}, s_I, s_G \rangle$, where $\mathcal{C}$ is a set of constants, $\mathcal{O}$ is a set of operators, and s_I and s_G are sets of ground literals describing the initial state and goal, respectively. An *operator* is a tuple $o = \langle \mathsf{name}(o), \mathsf{vars}(o), \mathsf{pre}(o), \mathsf{post}(o) \rangle$, where $\mathsf{name}(o)$ is the name, or *type* of the operator, $\mathsf{vars}(o)$ is a list of variables and $\mathsf{pre}(o)$ and $\mathsf{post}(o)$ are finite sets of (ground or non-ground) literals with variables taken from $\mathsf{vars}(o)$. It will be assumed that for all operators, $\mathsf{name}(o) = o$, and so operators will be referred to by name. When distinguishing between different operators of the same name, the notation $o(\vec{x})$ is used, where $\mathsf{vars}(o) = \vec{x}$. It is also required that operators with the same name be structurally equivalent, that is, if $\mathsf{name}(o) = \mathsf{name}(o')$, then there exists some substitution θ such that $\theta(o) = \theta(o')$. An *action* is a ground operator $a = \langle \mathsf{pre}(a), \mathsf{post}(a) \rangle$, where $\mathsf{pre}(a)$ and $\mathsf{post}(a)$ are finite sets of ground literals. If o is an operator and substitution θ is ground and complete *w.r.t.* $\mathsf{vars}(o)$, then $\langle \mathsf{pre}(o)\theta, \mathsf{post}(o)\theta \rangle$ is the action resulting from *instantiating* o with θ.

2.1.2.2 Classical and Partial-Order Plans

A *classical plan* $\vec{a}$ is a finite sequence of actions. With the above definitions of actions and operators in mind, and assuming that no variable appears in more than one operator, a plan can also be represented as $\vec{o}\theta$, where $\vec{o}$ is a list of operators and θ is a ground substitution that is complete *w.r.t.* every operator in $\vec{o}$. Representing a plan in this way separates the abstract definition of the plan's operators from the variable bindings used to create a particular instantiation.

As plans are typically discussed with reference to some planning task, it will be assumed every classical plan is book-ended by the distinguished actions a_I, which has no parameters, no preconditions and postconditions that are the plan's initial state, and a_G, which has no parameters, no postconditions and preconditions that are the plan's goal. In this form, a classical plan is valid *iff* it is *executable* (i.e., the actions can be applied in sequence without violating any precondition requirements).

A *partial-order plan* (POP) is a generalisation of a classical plan that defines which actions must be executed without (completely) defining their order. Instead, a set of ordering constraints is supplied in the form of a strict (i.e., irreflexive) partial order over the actions. Typically, a POP is defined as a tuple $P = \langle A, \prec \rangle$, where A is a set of actions and $\prec$ is a strict partial order over A. However, with the above definitions of actions, operators and substitutions in mind, and with the assumption that no variable appears in more than one operator, a POP can be equivalently represented as follows:

Definition 2.1. *A* **partial-order plan** *(POP) is a tuple* $P = \langle O, \theta, \prec \rangle$ *where* O *is a set of operators,* θ *is a ground substitution that is complete* w.r.t. *to* O*, and* $\prec$ *is a strict, transitively closed partial order over* O*.*

The advantage of this representation is that it separates the operator types from the bindings that create any particular instantiation, and allows for reasoning about the structure and validity of the POP to explicitly refer to a POP's variable bindings. As with classical plans, a planning task is "embedded" into a POP via the distinguished parameter-free operators o_I (which yields the initial state) and o_G (which checks the goals).

While the actions in a classical plan must be executed in sequence, unordered actions in a POP can be executed in any order. A POP is therefore a compact representation of a set of classical plans, known as its *linearisations*:

Definition 2.2. *A classical plan* $\vec{a} = \langle a_1, a_2, \ldots, a_n \rangle$ *is a* **linearisation** *of POP* $P = \langle A, \prec \rangle$ iff $A = \{a_1, a_2, \ldots, a_n\}$ *and* $\prec \subseteq \{a_i \prec a_j : 1 \leq i < j \leq n\}$.

A POP is *valid iff* all of its linearisations are valid:

Definition 2.3. *A POP is* **valid** iff *all of its linearisations are executable.*

Classical plans are special cases of POPs, and any classical plan $\vec{a} = \langle o_1, \ldots, o_n \rangle \theta$ can be expressed as an equivalent partial-order plan $\langle \{o_1, \ldots, o_n\}, \theta, \{o_i \prec o_j : 1 \leq i < j \leq n\} \rangle$.

2.1.2.3 The Producer-Consumer-Threat Formalism

The *producer-consumer-threat formalism* (PCT) [8] is typically used to describe a POP's causal structure by identifying which actions produce, consume or threaten which ground literals. Here, it describes how operators produce, consume or threaten (possibly non-ground) literals:

Definition 2.4. *Let* o *be an operator and* $q(\vec{s})$ *a literal. Then:*

$$\mathsf{prods}(o, q(\vec{s})) \stackrel{\text{def}}{=} q(\vec{s}) \in \mathsf{post}(o).$$

$$\mathsf{cons}(o, q(\vec{s})) \stackrel{\text{def}}{=} q(\vec{s}) \in \mathsf{pre}(o).$$

$$\mathsf{thrtns}(o, q(\vec{s})) \stackrel{\text{def}}{=} \neg q(\vec{s}) \in \mathsf{post}(o).$$

In the field of *partial-order causal link planning* (POCL), a PCT-based notion of POP validity is used that derives from the implied causal dependencies between a POP's operators [135], not the validity of its linearisations. A *causal link* associates a consumer, (i.e., an operator precondition), with a producer (i.e., an operator postcondition):

Definition 2.5. *A* **causal link** *is a 4-tuple* $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle$ *such that* $\mathsf{prods}(o_p, q(\vec{u}))$ *and* $\mathsf{cons}(o_c, q(\vec{s}))$.

The consumer is *supported* by the causal link *iff* it is preceded by, and codesignated with, the producer (i.e., $\theta(\vec{s}) = \theta(\vec{u})$). The causal link is *threatened iff* a codesignated threat (i.e., some $o_t, q(\vec{v})$ *s.t.* $\mathsf{thrtns}(o_t, q(\vec{v}))$ and $\theta(\vec{v}) = \theta(\vec{s})$) can be ordered between the producer and consumer. A POP P's *causal structure* is denoted L_P, and is a set of implicit unthreatened causal links defined as follows:

Definition 2.6. *The* **causal structure** *of a POP* $P = \langle O, \theta, \prec\rangle$ *is the set of unthreatened causal links* L_P s.t. $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle \in L_P$ iff*:*

- $o_p \prec o_c$,
- $\theta(\vec{s}) = \theta(\vec{u})$, *and*
- *for all* o_t, $q(\vec{v})$ s.t. $\mathsf{thrtns}(o_t, q(\vec{v}))$, *either* $\theta(\vec{v}) \neq \theta(\vec{s})$, $o_t \prec o_p$ *or* $o_c \prec o_t$.

A POP P is *POCL-valid iff* every consumer is supported by an unthreatened causal link:

Definition 2.7. *A POP* $P = \langle O, \theta, \prec\rangle$ *is* **POCL-valid** iff *for all* $o_c \in O$ *and* $q(\vec{s})$ s.t. $\mathsf{cons}(o_c, q(\vec{s}))$, *there is an* $o_p, q(\vec{u})$ s.t. $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle \in L_P$.

This notion of validity is stronger than that in Definition 2.3, meaning that a POP might not be POCL-valid despite its linearisations all being executable [72]. However, any such POP can always be made POCL-valid by adding ordering constraints [11].

2.1.3 Parameterised Complexity

Rather than measuring a problem's time complexity solely as a function of the size of the input instance, the *parameterised complexity* [36] approach measures time complexity in terms of both the instance size and one or more additional parameters. The parameter typically measures some structural property of a problem instance. Parameterised complexity can thus provide a more fine-grained analysis than classical complexity analysis, and can better identify the structural properties of problem instances which are responsible for a problem's complexity.

The development of the field was motivated by the existence of problems that are intractable (assuming that $\mathsf{P} \neq \mathsf{NP}$) when their complexity is measured only in terms of the instance size, but are computable in a time which grows polynomially *w.r.t.* the instance size and exponentially *w.r.t.* a small parameter. For example, given a graph $G = \langle V, E\rangle$ and an integer k, the Vertex Cover problem asks whether there exists a V' such that $|V'| \leq k$ and if $(u, v) \in E$ then either $u \in V'$ or $v \in V'$. While Vertex Cover is NP-complete [73], it is also solvable in time $\mathcal{O}(1.2738^k + (k \times |G|))$ [23], meaning that even large problem instances can be solved efficiently when k is fixed to some small value.

An algorithm is an *fpt-algorithm* if it has time complexity of $\mathcal{O}(f(k) \times n^{\mathcal{O}(1)})$, where n is the size of the input and f is a computable function

depending solely on parameter k. Similarly, a decision problem is in complexity class FPT *iff* it can be solved by an fpt-algorithm. When f is exponential, exponential time is required. However, if k is *fixed*, then the time required scales polynomially with n, meaning that problems in FPT remain feasible so long as k stays small.

The complexity of parameterised problems can be compared by way of *fpt-reduction*, an fpt-algorithm that reduces one parameterised problem to another while preserving the solution set, instances sizes and parameters:

Definition 2.8. *If Q and Q' are parameterised decision problems, then Q is* **fpt-reducible** *to Q'* iff *every instance (I, k) of Q can be transformed, in fpt-time with parameter k, into an instance (I', k') of Q' such that $k' \leq f(k)$, where f is a computable function, and $(I, k) \in Q$* iff *$(I', k') \in Q'$.*

Complexity class FPT is the first level of the parameterised hierarchy, which comprises the classes $\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \cdots \subseteq \mathsf{W[P]} \subseteq \mathsf{XP}$. A parameterised problem with parameter k is in class W[1] *iff* it can be fpt-reduced to the problem of determining whether a 3-CNF Boolean formula has a satisfying assignment with weight at most k (i.e., that sets at most k variables to *true*). Problems in class W[t] for $t > 1$ can be fpt-reduced to the problem of determining whether a Boolean formula with a nesting depth of at most t has a satisfying assignment with weight at most k [21]. Complexity class W[P] contains all problems solvable by a non-deterministic Turing machine in $f(k).n^c$ steps, of which at most $h(k).\log(n)$ are non-deterministic (where f and h are computable functions, c is a constant and k is the parameter). Class XP contains all parameterised problems that can be solved in a running time of $\mathcal{O}(n^{f(k)})$. As with FPT, if k is fixed, then XP problems scale polynomially with the size of the input. However, as k is in the exponent, large problems may become infeasible.

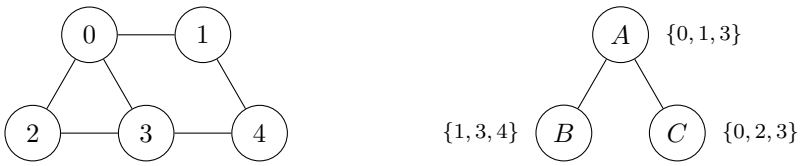
2.1.4 Treewidth

It has been observed that many computationally hard graph problems are tractable when parameterised with the *treewidth* [113] of the graph. Consequently, a common approach in parameterised complexity analysis is to demonstrate that an otherwise intractable problem is solvable in polynomial time when limited to instances with an underlying structure that can be described as a graph with bounded treewidth.

A graph's treewidth is a positive integer that, intuitively, measures its "cyclicity": a tree has a treewidth of 1, indicating no cycles, and a complete graph (or clique) of n vertices has a treewidth of $n-1$, indicating the presence of every possible cycle. More formally, treewidth is defined with reference to *tree decompositions*:

Definition 2.9. *Let $G = \langle V, E \rangle$ be a graph, $T = \langle W, F \rangle$ be a tree and $X : W \to 2^V$ be a function which maps every node in W to a subset of V. Then, $D = \langle T, X \rangle$ is a* **tree decomposition** *of graph G, denoted* $\mathsf{tree\text{-}decomposition}(G, D)$, iff*:*

- *for every $v \in V$ there is a $w \in W$ such that $v \in X(w)$,*
- *for every $(v, v') \in E$ there is a $w \in W$ such that $v, v', \in X(w)$, and*
- *if $v \in X(w)$, $v \in X(w')$ and w'' is on a path from w to w', then $v \in X(w'')$.*



(a) A graph G with treewidth of two. (b) A tree decomposition D of graph G in Figure 2.1a.

Figure 2.1: Treewidth example.

For example, the tree D in Figure 2.1b is a tree decomposition of graph G in Figure 2.1a. The nodes in D are annotated with their associated vertices in G. The decomposition D satisfies Definition 2.9, that is, *(i)* every vertex in G is associated with some node in D, *(ii)* for every pair of connected vertices in G, there is a node in D associated with them both, and *(ii)* if a vertex in G is associated with two nodes in D, it is also associated with all nodes on any path between them: in this case vertex 3 is associated with nodes B and C, and so is also associated with A.

The *width* of a tree decomposition is one less than the largest number of vertices associated with any node, and the *treewidth* of a graph is equal to the smallest width of all of its tree decompositions. The decomposition in Figure 2.1b has a width of two, and as it is a minimal decomposition of graph G, it follows that G has a treewidth of two. The problem of deciding whether a graph's treewidth is bounded by $k > 0$ is NP-complete [7] and in FPT when parameterised with k [15].

2.1.5 Partial Weighted MaxSAT

The *propositional satisfiability* problem (SAT) [13] asks whether there exists an interpretation of a given propositional formula (i.e., an assignment of truth values to its propositional variables) such that the formula evaluates to *true*. Typically, the formulae are expressed in conjunctive normal form (CNF), that is, as a conjunction of clauses where each clause is a disjunction of literals. An interpretation will satisfy a CNF formula *iff* it satisfies each of the formula's clauses.

The problem of *partial weighted maximum satisfiability* (partial weighted MaxSAT) is a generalised optimisation variant of SAT that distinguishes between *soft* clauses, which are assigned a numeric weight, and *hard* clauses, which are unweighted. The aim of the partial weighted MaxSAT problem is to find an interpretation that satisfies all hard clauses and maximises the total weight of the satisfied soft clauses. Here, the syntax $\overset{k}{\phi}$ is used to indicate that clause ϕ has weight k, and no weight marking indicates a hard clause. For example, the formula below is trivially unsatisfiable:

$$(\mathrm{p}_1 \overset{3}{\vee} \mathrm{p}_2) \wedge \overset{2}{\neg} \mathrm{p}_1 \wedge \neg \mathrm{p}_2 \,.$$

However, the interpretation $\{\mathrm{p}_1 \setminus \top, \mathrm{p}_2 \setminus \bot\}$ maximises the soft clause weights (3) while satisfying the hard clause, and is thus an optimal solution to the partial weighted MaxSAT problem.

2.1.6 Constraint Satisfaction Problems

A *constraint satisfaction problem* (CSP) [126] asks whether a set of variables can be assigned values that satisfy a given set of constraints. A CSP typically comprises a set of variables, their domains, and a set of constraints that specify the allowable combinations of variable bindings:

Definition 2.10. *A* **Constraint Satisfaction Problem** *(*CSP*)* $S = \langle X, D, C\rangle$ *where:*

- $X = \{x_1, \ldots, x_n\}$ *is a set of variables,*
- $D = \{D_1, \ldots, D_n\}$ *is the domains for the variables in* X*, and*
- $C = \{C_1, \ldots, C_m\}$ *is the constraints* s.t. *for* $1 \leq i \leq m$*,* $C_i = \langle t_i, R_i\rangle$ *where* $t_i = \langle x^i_1, \ldots, x^i_k\rangle$ *is the constraint scope, and* $R_i : D_{x^i_1} \times \cdots \times D_{x^i_k}$ *is a* k*-ary relation over the corresponding domains.*

A solution is a ground substitution θ *that is complete* w.r.t. X s.t.*:*

- *for all* $x \in X$*,* $\theta(x) \in D_x$*, and*
- *for all* $\langle\langle x^i_1, \ldots, x^i_k\rangle, R_i\rangle \in C$*,* $\langle\theta(x^i_1), \ldots, \theta(x^i_k)\rangle \in R_i$*.*

While Definition 2.10 represents constraints *extensionally*, that is, as sets of allowable combinations of bindings, they are frequently expressed *intentionally*, that is, as a compact formula. For example, if $X = \{x_1, x_2\}$ and $D_{x_1} = D_{x_2} = \{0, 1\}$, then the extensional constraint $\langle\langle x_1, x_2\rangle, \{\langle 0, 0\rangle, \langle 1, 1\rangle\}\rangle$ could be represented intentionally as $x_1 = x_2$.

2.1.7 Groups, Permutations and Automorphisms

Much of the later work on symmetry breaking will be presented in the context of groups, permutations and automorphisms [66]. The purpose of this section is not to give a full description of the field, but rather provide the background concepts that will be relied upon later.

2.1.7.1 Groups

A *group* is an algebraic structure comprising a set and a binary operator that satisfy four conditions:

Definition 2.11. *A* **group** $G = \langle S, \cdot\rangle$ *is a set of items* S *and a binary operator* $\cdot : S \times S$ s.t.*:*

- *if* $s_1, s_2 \in S$*, then* $s_1 \cdot s_2 \in S$ *(closure),*
- $s_1 \cdot (s_2 \cdot s_3) = (s_1 \cdot s_2) \cdot s_3$ *(associativity),*
- *there exists an* s_{id} s.t. $s_{id} \cdot s = s \cdot s_{id} = s$ *for all* s *(identity), and*
- *if* $s \in S$ *there is an* s^{-1} s.t. $s \cdot s^{-1} = s^{-1} \cdot s = s_{id}$ *(invertibility).*

For example, the set of integers $\mathbb{Z}$ and the addition function $+$ satisfy the above requirements: the sum of two integers is an integer (closure), $x + (y + z) = (x + y) + z$ (associativity), there is an identity element, 0, as $x + 0 = x$ (identity), and each integer x has an inverse, $-x$, as $x + -x = 0$ (invertibility). Thus, $\langle \mathbb{Z}, + \rangle$ is group.

A *generating set* is a compact representation of a group. If $G = \langle S, \cdot \rangle$ is a group, then $\Sigma \subseteq S$ is a generating set of G *iff* S is equal to the closure of Σ under $\cdot$:

Definition 2.12. *A* **generating set** Σ *of a group* $G = \langle S, \cdot \rangle$ *is a subset of* S *such that* $s \in S$ iff *there exists an* $s_1, s_2, \ldots, s_n$ *such that* $s = s_1 \cdot s_2 \cdot, \cdots, \cdot s_n$ *and for* $1 \leq i \leq n$ *either* $s_i \in \Sigma$ *or* $s_i^{-1} \in \Sigma$.

Continuing the example above, $\Sigma = \{1\}$ is a generating set for $G = \langle \mathbb{Z}, + \rangle$, as every integer is expressible as a sequence of additions of 1 or -1.

The group-theoretic analogue of the Cartesian product of sets is known as the *direct product*, and is defined as follows:

Definition 2.13. *If* $G = \langle S, \cdot \rangle$ *and* $H = \langle T, \star \rangle$ *are groups then the* **direct product** $G \times H$ *is the group* $\langle U, \triangle \rangle$, *where*

- $U = S \times T$, *i.e.,* U *contains all* (s, t) s.t. $s \in S$ *and* $t \in T$, *and*
- $\triangle$ *is a binary operation over* U s.t. $(s_1, t_1) \triangle (s_2, t_2) = (s_1 \cdot s_2, t_1 \star t_2)$

A *subgroup* of a group is a subset that is itself a group:

Definition 2.14. *If* $G = \langle S, \cdot \rangle$ *and* $H = \langle T, \star \rangle$ *are groups then* G *is a* **subgroup** *of* H, *denoted* $G \leq H$, iff $\cdot = \star$ *and* $S \subseteq T$.

2.1.7.2 Permutations and Symmetry Groups

A *permutation* is a bijective function from a set to itself. A permutation can be concisely defined in *cyclic notation*, which represents a permutation as a series of circular transformations of groups of elements. For example, the notation $\sigma = (c_1\ c_2\ c_3)(c_4\ c_5)$ describes a permutation with two such cycles. The first cycle maps c_1 to c_2, c_2 to c_3 and c_3 back to c_1, and the second cycle swaps c_4 and c_5. All remaining elements from $\mathsf{domain}(\sigma)$ not explicitly contained in a cycle are fixed.

The set mapped by permutation σ is denoted $\mathsf{domain}(\sigma)$, and $\sigma(t)$ denotes the item that t is mapped to under permutation σ. If η is a logical structure, then, when clear from context, the notation $\sigma(\eta)$ is used to denote the result of simultaneously replacing every term t in η with $\sigma(t)$, while preserving the structure (or type) of η. For example, if $\vec{t} = \langle t_1, \ldots, t_n \rangle$ is a list, then $\sigma(\vec{t})$ denotes the list $\langle \sigma(t_1), \ldots, \sigma(t_n) \rangle$, if $R : S \times S$ is a relation then $\sigma(R)$ is the relation such that $(t, t') \in R$ *iff* $(\sigma(t), \sigma(t')) \in \sigma(R)$, and if $\theta = \{x_1 \backslash t_1, \ldots, x_n \backslash t_n\}$ is a substitution, then $\sigma(\theta)$ denotes the substitution $\{\sigma(x_1) \backslash \sigma(t_1), \ldots, \sigma(x_1) \backslash \sigma(t_n)\}$.

The *identity permutation* σ_{id} maps every item to itself, and σ^{-1} is the *inverse* of σ such that $\sigma^{-1}(s) = s'$ *iff* $\sigma(s') = s$. The *composition* of two permutations is denoted $\sigma_1 \circ \sigma_2$, and results in a permutation that maps every item s to $\sigma_1(\sigma_2(s))$. A set of permutations along with their

inversions and the identity permutation, all closed under composition, and the composition operator form a group, known as a *symmetry group*.

A useful class of symmetry groups are the *symmetric groups*, which define all possible permutations over a set:

Definition 2.15. *The* **symmetric group** *over a set X is the group $S_X = \langle P, \circ \rangle$ where P contains all permutations over X and $\circ$ is the composition operator. The symmetric group over $\{1, 2, \ldots, n\}$ is denoted S_n.*

A symmetric group over a set of size n contains $n!$ permutations. For example, S_3 contains the permutations $(1\ 2), (2\ 3), (3\ 1), (1\ 2\ 3), (1\ 3\ 2)$ and the identity permutation.

2.1.7.3 Automorphisms

Broadly speaking, an *automorphism* of a logical structure η is a permutation σ such that $\sigma(\eta) = \eta$. For example, if $R : S \times S$ is a relation then σ is an automorphism of R *iff* $R(s, s')$ implies $R(\sigma(s), \sigma(s'))$ and *vice versa*. The problem of finding the automorphisms of a logical structure is typically reduced to the problem of finding of the automorphisms of a coloured graph representation of that structure. This particular type of automorphism is formally defined as permutation over the coloured graph's vertices which preserves its edge set and colours:

Definition 2.16. *Let $G = \langle V, E, C \rangle$ be a coloured graph where V is a set of vertices, $E : V \times V$ is a set of edges and $C : V \to \mathbb{N}$ maps every vertex to a colour, represented here as a positive integer. An* **automorphism** *of G is a permutation σ such that $(v_1, v_2) \in E$ iff $(\sigma(v_1), \sigma(v_2)) \in E$, and for all $v \in V$, $C(v) = C(\sigma(v))$. All automorphisms of graph, along with the composition operator, form a group, denoted $\mathsf{Aut}(G)$.*

The problem of determining whether a graph has a non-identity automorphism is in NP, but has no known polynomial time algorithms [83]. Nevertheless, in practice there exist algorithms that can efficiently find a generating set for the automorphisms of a graph of hundreds of thousands of vertices [85].

2.2 Least Commitment Planning

In order to achieve its goals in a dynamic environment, an agent must plan ahead while remaining flexible enough to handle unexpected events. The aim of the *least commitment* approach to planning is to improve the agent's flexibility by only making decisions regarding the ordering of actions, or the domain objects used by the plan, when absolutely necessary.

Under this strategy, plans are typically represented as a set of actions and a set of ordering constraints over those actions, that is, as a partial-order plan (POP) [8]. A POP is a compact representation of a set of plans – the POP's linearisations – all of which contain the same actions, and achieve the same goals, but differ in the order in which the actions are

executed. Other approaches go further, and instead of completely specifying the POP's actions, instead supply a set of constraints that define the allowable combinations of variable bindings (e.g., [71]). These plan representations reduce the agent's commitment to any particular execution order or set of domain objects.

The benefit of least-committed plans is that the scheduler (or executing agent) has more plans to choose from at execution time. If the agent is situated in an environment which is dynamic, partially observable, or contains other active agents, it may find that its intended plan becomes infeasible due to unforeseen external events. A least-committed plan may allow a feasible alternative to be found without resorting to replanning. Thus, the use of least-committed plans can provide an agent with both a robust guarantee of success if the plan is followed, as well as a degree of flexibility that is beneficial for real-time execution.

The least-commitment approach is reflected in the field of *partial-order planning* [135]. A partial-order planner does not commit to a sequence of actions, instead it begins with an empty plan, and only commits to the timing of actions, or variable bindings (e.g., [9, 139]) as actions are added and constraints emerge.

Another common approach to minimising commitment is to post-process the output of a state-space planner to remove over-commitments to action ordering (e.g., [8, 33, 91, 116, 120]) and/or domain objects (e.g., [71]), or to even optimise the plan size by removing unnecessary actions (e.g., [91]). There are two benefits to the post-processing approach. Firstly, it takes advantage of recent developments in state-space planning, which, due to the effectiveness of heuristic-based planners, is currently receiving more attention from the planning community than partial-order planning. Secondly, by separating the processes of plan generation and optimisation, it allows for the optimisation of plans that do not originate from an automated planner, but instead from, for example, a human subject-matter expert.

This section will next discuss various approaches to partial-order planning, and then examine some different notions of optimal or least-committed plans, and some practical approaches to the intractable problem of computing such optimums.

2.2.1 Partial-Order Planning

2.2.1.1 Partial-Order Causal Link Planning

Standard partial-order planners perform a search through *plan space* in a process known as *partial-order causal link planning* (POCL planning) [135]. Nodes and edges in the POCL search space represent (possibly partially specified) POPs, plan refinement operations, respectively. The search process begins with an empty POP, and is complete when POCL-valid node is found. Each non-goal node represents a POP with at least one *flaw*, in that it has either an open precondition, (i.e., a consumer not causally linked to any producer), or a threatened causal link. Open preconditions can be resolved by adding in actions and/or causal links, and threats can be resolved by either promotion, demotion or separation (i.e., ordering the threat before the producer or after the consumer, or rebinding the variables). The two

steps in POCL planning are therefore *plan selection*: deciding which POP to refine, and *flaw selection*: choosing a flaw to resolve, and generating successor POPs that resolve it.

Plan selection strategies are typically based on A^* search. The early, well-studied planners SNLP by Mcallester and Rosenblitt [84] and UCPOP by Penberthy and Weld [98] estimate the remaining search cost as the number of open preconditions. Nguyen and Kambhampati [95] use heuristics adapted from state-space planning (e.g., planning graphs [14]) to estimate the number of actions required to resolve open preconditions, and prune plans with inconsistent constraints from the search space, and Vidal and Geffner [130] prune plans by combining heuristic lower bounds with a constraint programming technique that detects whether a plan will always lead to a dead-end. More recently, Younes and Simmons [141] use a variation of the additive heuristic [17] that can account for subgoal independence and action reuse, and an estimate of number of actions needed to achieve particular preconditions [140].

The flaw selection strategy of SNLP and UCPOP is to resolve threats before open preconditions. A wide range of other strategies have been studied. Peot and Smith [99] propose delaying separable or unforced threats (i.e., those with more than one way to be resolved), and Gerevini and Schubert [50] take a "least commitment" approach and resolve non-separable threats and forced open conditions first. In contrast, Joslin and Pollack [67] do not treat threats and open preconditions differently, and order all flaws by how many ways they can be resolved. Younes and Simmons [141] explore the use of heuristic-based flaw selection, and conflict-driven flaw selection, that aims to detect inconsistent plans early in the search.

While partial-order planning is an intuitively attractive approach to plan synthesis, in practice it has been far less successful than planning techniques based on early-commitment, heuristic-guided state-space search. The next two sections will outline planning techniques which represent a middle ground between least-commitment partial-order planning and early-commitment state-space planning.

2.2.1.2 Forward-Chaining Partial-Order Planning

The POPF partial-order planner developed by Coles et al. [30] combines the least-commitment strategy of partial-order planning with the powerful advances in heuristic-based forward-chaining planning. As in forward-chaining planners, nodes in the POPF search space represent both a plan and the state that results from its execution, and edges represent the addition of actions to the plan. However, rather than simply append new actions to a linear plan, POPF maintains a POP and adds only the ordering constraints needed to ensure that preconditions are met, threats are avoided, and importantly, the state representing the result of executing the POP can be computed.

Maintaining the post-execution state allows POPF to make use of the powerful heuristic approach of forward-chaining planners, and maintaining a POP allows it to avoid exploring multiple linear plans which differ only by an irrelevant permutation of actions.

2.2.1.3 Petri Net Unfolding

Hickmott et al. [63] introduce a novel approach for synthesising parallel partial-order plans by transforming a planning instance into a *Petri net reachability* problem [93]. Petri nets are compact representations of state transition systems that expressly model parallelism and causal connections between possible state changes, and are frequently used for modelling and analysing distributed systems. A Petri net is (broadly speaking) a directed bipartite graph of *transition* nodes, that represent state changes, and *place* nodes, that represent transition preconditions and contain tokens when those preconditions are satisfied. The *firing* of a transition moves tokens from a transition node's immediate predecessors to its immediate successors. The Petri net *reachability problem* is a search for a sequence of transition firings that results in a desired token placement, and is solved by *unfolding* the Petri net. The unfolding process, like POPF, reasons about partially-ordered actions while still having access to a fully-specified state, meaning that state-based heuristics can be used to direct the search.

Analysis by Hickmott and Sardiña [62] gives a theoretical foundation to the clear parallels between Petri net unfolding and partial-order planning. Their results reveal that Petri net unfolding produces POPs that are minimally deordered under the requirement of *strong independence* (actions with conflicting pre/postconditions cannot be unordered, and there is no ambiguity as to which producer supports a consumer), and, with the appropriate unfolding heuristic, have a minimal parallel plan length.

2.2.2 Partial-Order Plan Optimisation

2.2.2.1 Optimality Definitions for Partial-Order Plans

The seminal work on action ordering for plan flexibility is that of Bäckström [8], which studies the complexity of the problems of deordering and reordering POPs in order to meet various optimality criteria. Two types of modifications are considered: *deordering*, in which constraints can be removed but not added, and *reordering*, in which any modification can be made. In both cases, the modifications must result in a *valid* POP:[1]

Definition 2.17. *Let* $P = \langle O, \theta, \prec \rangle$ *and* $Q = \langle O, \theta, \prec' \rangle$ *be POPs. Then:*

- Q *is a* **reordering** *of* P iff *they are both valid,*
- Q *is a* **deordering** *of* P iff *it is a reordering and* $\prec' \subseteq \prec$*, and*
- Q *is a* **strict deordering** *of* P iff *it is a deordering and* $\prec' \subset \prec$*.*

A number of optimality criteria are introduced. These criteria define the relative optimality of two POPs by comparing their ordering relations based on set inclusion and cardinality. A *minimal deorder* of a POP cannot be relaxed any further (i.e., removing any ordering constraint will render the POP invalid), and a *minimum deorder* is the smallest of all valid deorderings of a POP. A *minimum reorder* places the fewest possible constraints over the POP's actions while remaining valid:

[1] From Definition 2.1, these optimality criteria always compare the transitive closure of the POPs' ordering constraints.

Definition 2.18. *Let* $P = \langle O, \theta, \prec \rangle$ *and* $Q = \langle O, \theta, \prec' \rangle$ *be POPs. Then:*

- Q *is a* **minimal deordering** *of* P iff Q *is a deordering of* P *and there is no POP* R *such that* R *is a strict deordering of* Q*,*
- Q *is a* **minimum deordering** *of* P iff Q *is a deordering of* P *and there is no POP* $R = \langle O, \theta, \prec'' \rangle$ *such that* R *is a deordering of* P *and* $| \prec'' | < | \prec' |$*, and*
- Q *is a* **minimum reordering** *of* P iff Q *is a reordering of* P *and there is no POP* $R = \langle O, \theta, \prec'' \rangle$ *such that* R *is a reordering of* P *and* $| \prec'' | < | \prec' |$.

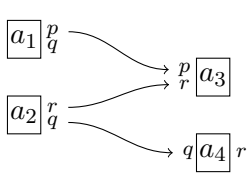
A minimal deorder of a given POP can be found in polynomial time. However, deciding whether there exists a deorder or reorder with fewer than k ordering constraints is NP-complete, and finding a minimum deorder or reorder is NP-hard and cannot be approximated within a constant factor. Aghighi and Bäckström [3] extend this analysis from a parameterised complexity perspective: deciding whether there exists a minimum deorder or reorder with fewer than k ordering constraints are both W[2]-hard and in W[P] when parameterised with k, and when parameterised with the size of the original order relation, deciding the existence of a minimum deorder or reorder of any size are in FPT and W[P], respectively.

While the optimality definitions above have become the *de facto* standard in the plan relaxation literature, the approach is not without its problems. Firstly, these definitions cannot compare POPs with different actions, which is required to compare the performance of different partial-order planners, or a reordering algorithm over different problem sets. Secondly, while Bäckström correctly states that these optimality definitions are "intuitively reasonable", the further claim that "it is questionable whether considerably better or more natural definitions [exist]", is less certain. An obvious alternative flexibility measure is the number of linearisations represented by a POP: a definition which does not always coincide with the number of ordering constraints.

Muise et al. [91] demonstrate that two partial orders of the same cardinality may not have the same number of linearisations. Their example can be extended to show that two minimum reorderings of the same plan can represent different sized sets of classical plans. Figure 2.2 depicts two minimum reorders of the plan $\langle a_1, a_2, a_3, a_4 \rangle$.[2] Despite having the same number of (transitive) ordering constraints, the POP in Figure 2.2a has five linearisations, while that in Figure 2.2b has only four. Thus, Bäckström's definitions would classify both POPs as equally optimal despite Figure 2.2a being intuitively less constrained.

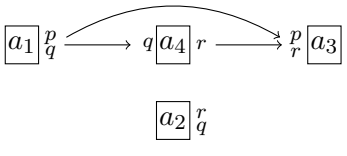
As a result, many authors use richer definitions of optimality. Originally introduced by Nguyen and Kambhampati [95], *flex* measures the proportion of actions in a POP which are not (transitively) ordered, and so avoids the problem of comparing the optimality of plans with different

[2] In all plan diagrams in this section, symbols to the left and right of actions indicate preconditions and postconditions, respectively, and arrows indicate ordering constraints. Initial and goal states are omitted for clarity.



$\langle a_1, a_2, a_3, a_4 \rangle$
$\langle a_1, a_4, a_3, a_4 \rangle$
$\langle a_2, a_1, a_3, a_4 \rangle$
$\langle a_2, a_1, a_4, a_3 \rangle$
$\langle a_2, a_4, a_1, a_3 \rangle$

(a) A minimum reorder with five linearisations.



$\langle a_2, a_1, a_4, a_3 \rangle$
$\langle a_1, a_2, a_4, a_3 \rangle$
$\langle a_1, a_4, a_2, a_3 \rangle$
$\langle a_1, a_4, a_3, a_2 \rangle$

(b) A minimum reorder with four linearisations.

Figure 2.2: Two minimum reorders of the plan $\langle a_1, a_2, a_3, a_4 \rangle$ with different numbers of linearisations.

actions. Say et al. [116] and Muise et al. [91] explicitly define optimality in terms of linearisations, although as counting linearisations is #P-complete [20], Muise et al. also introduce a modified *flex* measure that correlates with a POP's linearisation count.

However, implemented POP relaxation algorithms typically do not directly optimise either the *flex* value or the number of linearisations. Instead, Bäckström's criteria serve as a computable "proxy" for the more complex optimality definitions, that is, algorithms are developed which minimise the number of ordering constraints, and the success of these algorithms is evaluated based on, for example, the *flex* or (approximate) linearisation count of the generated POPs.

Other authors have studied the problem of reordering POPs in order to optimise other aspects of the POP. For example, as modifying a POPs ordering constraints can modify its (implicit) causal links, and actions without dependent consumers need not be executed, reordering can allow the POP size to be minimised. Olz and Bercher [96] study the theoretical aspects of plan size minimisation, and show that deciding whether a POP admits a de/reordering that allows actions to be removed is NP-complete. Similarly, modifying orderings can allow a POP's makespan to be minimised, and Bercher and Olz [11] show that finding a de/reorder that allows a reduction in parallel plan length is also NP-complete.

The rest of this section will examine some practical approaches to post-processing POPs.

2.2.2.2 Deordering and Debinding via Explanation Based Generalisation

Kambhampati and Kedar [71] study the problem of relaxing both action orderings *and* variable bindings in the context of *explanation based generalisation*. The notion of a *partially ordered, partially instantiated plan* (POPI) is introduced. A POPI is a generalisation of a POP that specifies

which action *types*(i.e., operators) must be executed, without (completely) specifying their order or their variable bindings. Like POPs, POPIs are compact representations of a set of classical plans. However, a POPI's instantiations can differ in their variable bindings (i.e., the domain objects used by actions) as well as the order in which actions are executed.

A POPI is a tuple $P = \langle O, \Phi, \prec \rangle$ where O is a set of operators, $\prec$ is a strict partial order over O, and Φ is a set of literals of the form $\vec{t} = \vec{u}$ or $\vec{t} \neq \vec{u}$, where $\vec{t}$ and $\vec{u}$ are lists of terms (i.e., variables or constants). There is no syntax for representing domain constraints, and many of the paper's complexity claims are under the assumption of infinite domains.

Two techniques are presented, *explanation-based order generalisation* (EOG) and *explanation-based precondition generalisation* (EPG), that relax a POPI's ordering and variable binding constraints, respectively. Both algorithms use a *validation structure* – a subset of L_P (Definition 2.6) that "explains" the plan's validity – as a guide for this relaxation. A validation structure links every consumer in a POPI with the *supporting* producer (Section 2.1.2.3) that appears *first* in an arbitrary linearisation.

The EOG algorithm removes all ordering constraints that are not required by the validation structure, that is, consumers must be preceded by their associated consumers, and any threats to the link that can be codesignated with the consumer must either precede the producer, or be preceded by the consumer, depending on how there were ordered in the input POPI. Similarly, EPG removes all binding constraints except those required by the validation structure, that is, consumers and producers must be codesignated, and any threats that can be ordered between them must remain non-codesignated.

The authors claim that EOG and EPG produce relaxations that are optimal *w.r.t.* a validation structure, but provide no formal definitions of POPI optimality. Additionally, as EOG relies on a choice of validation structure, it does not guarantee even a minimal deordering of its input Bäckström [8].

To build a validation structure, or compute an EPG, it must be determined whether the POPI's binding constraints entail a given literal, which is in P when variables domains are infinite. Similarly, under the infinite domain assumption, finding a classical plan which satisfies the POPI's binding constraints is trivial. However, if this assumption is dropped then no complexity guarantees are given for either POPI relaxation or instantiation.

2.2.2.3 Finding Optimally Ordered POPs with MaxSAT

Muise et al. [91] introduce a practical technique for the problem of finding optimal de/reorderings of a POP by encoding it as a partially weighted MaxSAT instance (Section 2.1.5). Two MaxSAT encodings are defined: MD and MR. In both encodings, plan validity is preserved by expressing POCL validity as hard clauses that must be satisfied by any solution. Further clauses and clause weights are constructed to ensure that an optimal solution to the MaxSAT instance corresponds to a minimum deordering (in the case of MD) or reordering (in the case of MR) of the input plan.

The quality of the POPs produced by MD and MR are compared experimentally with those produced by the polynomial, non-optimal deorder-

ing algorithm EOG (Section 2.2.2.2) over test instances taken from 15 IPC STRIPS domains. Results show that the MaxSAT approach is an effective optimisation technique, despite at times being unable to encode the acyclicity constraints: closing the precedence relation for an n-step plan requires n^3 clauses, which is infeasible for $n > 200$. The comparative results show that *(i)* despite having no optimality guarantees, EOG algorithm always produces a minimum deorder, and *(ii)* searching for a minimum reorder with MR, rather than a minimum deorder with MD, provides very little increase in plan quality (less than 0.01 average *flex* increase in all but three domains, and a maximum average increase of 0.09).

Muise et al. [91] also present an extension to MR that minimises both ordering constraints *and plan size*. The notion of a *minimum cost least commitment POP* (MCLCP) is introduced, an extension of Bäckström's optimality definitions that can account for differences in POP size. Assuming uniform action cost, an MCLCP can be defined as follows:

Definition 2.19. *Let $P = \langle O, \theta, \prec \rangle$ and $Q = \langle O', \theta, \prec' \rangle$ be two POPs and c be a cost function over POPs. Then, P is a* **minimum cost least commitment POP** *of Q* iff *it is a minimum reordering of itself, and there is no POP $R = \langle O'', \theta, \prec'' \rangle$* s.t. *$O'' \subset O$.*

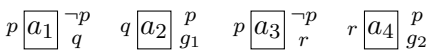
As a POP with both minimal cost and orderings may not exist, MCLCP optimisation prioritises plan cost. It is therefore possible for an MCLCP to have lower cost but more ordering constraints than a minimum reorder.

A generalisation of MR is introduced that allows action removal, and includes additional soft clauses ensuring that an optimal solution corresponds to an MCLCP of a given POP. Experimental results [89] show that an MCLCP typically has both lower cost and fewer ordering constraints than a minimum reorder. However, as MCLCP removes actions, the difference in ordering constraints is a not an indicator of increased flexibility. When MCLCP does not remove actions, the number of ordering constraints coincides with a minimum reorder.

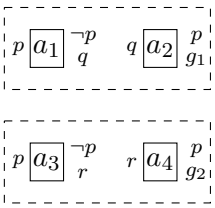
2.2.2.4 Finding Optimal POPs with MILP models

Do and Kambhampati [33] introduce a number of MILP models for converting *parallel position-constrained plans*, that is, plans that specify each action's execution time, into optimal POPs. Beyond minimising orderings, three other optimality definitions are considered: minimising the makespan, maximising the *summation of slacks* (the distances between a_G and each action that achieves a goal) and maximising the *average flexibility* (the average range of possible action start times). However, of these only makespan optimisation is evaluated.

As discussed above, Bäckström [8]'s optimality definitions are often used as a computable proxy for more complex flexibility measures based on, for example, linearisation count. Say et al. [116] extend the work of Do and Kambhampati [33] and Muise et al. [91], and introduce MILP models that implement three such proxy functions. The $\mathcal{O}_C^{MILP}$ model finds an MCLCP (Definition 2.19), and so optimises the plan size, then the number of (transitively closed) ordering constraints. The $\mathcal{O}_O^{MILP}$ and $\mathcal{T}^{MILP}$ models optimise plan size and then *open ordering constraints* (a minimal



(a) A plan that achieves the goal $s_g = \{g_1, g_2\}$. It cannot be deordered.



(b) A block decomposition of Figure 2.3a. The two blocks can be executed in any order.

Figure 2.3: Block deordering example.

set of constraints derived from POCL validity) and *temporal flexibility* (the sum of the time between each action's earliest possible start time and latest possible finish time), respectively.

While theoretically none of these proxies always produces more linearisations than the others, in practice $\mathcal{T}^{MILP}$ produces (on average) more linearisations than $\mathcal{O}_C^{MILP}$ and MR. Unlike $\mathcal{O}_C^{MILP}$, MR, MD and MCLCP, where ordering constraints are encoded explicitly with a propositional variable for each pair of actions, $\mathcal{O}_O^{MILP}$ and $\mathcal{T}^{MILP}$ encode them implicitly with variables representing action start and end times. This avoids the cubic number of clauses required to enforce transitivity in the explicit encodings, allowing $\mathcal{O}_O^{MILP}$ and $\mathcal{T}^{MILP}$ to find optimal solutions significantly more often than $\mathcal{O}_C^{MILP}$ and MR.

2.2.2.5 Block Deordering

Siddiqui and Haslum [120] propose a generalised notion of plan deordering termed *block deordering*. A *block* is a set of partially ordered actions that behaves like a macro-operator, that is, it produces and consumes propositions, and, when ordering constraints exist between actions in different blocks, blocks themselves can be (implicitly) partially ordered. A *block decomposition* of a POP is a division of the actions into a possibly recursive set of blocks, with the requirement that no action outside of a block can be interleaved with actions within a block.

Because causal dependencies between actions in the same block can be ignored when deordering blocks, block deordering is sometimes able to deorder plans that cannot be deordered by standard methods. For example, Figure 2.3a depicts a plan which achieves the goal $s_G = \{g_1, g_2\}$ from an initial state $s_I = \{p\}$. The plan is totally ordered and is a minimum reordering, and cannot be further optimised by standard de/reordering methods. Figure 2.3b shows a block decomposition of Figure 2.3a. While $a_1 \prec a_2$ and $a_3 \prec a_4$, are still required, no ordering constraints exist between actions in different blocks, meaning that Figure 2.3b allows two linearisations: $\langle a_1, a_2, a_3, a_4 \rangle$ and $\langle a_3, a_4, a_1, a_2 \rangle$.

A polynomial anytime block decomposition algorithm is compared experimentally with EOG (Section 2.2.2.2) over a number of IPC domains. The quality of the resulting POPs are compared by their *flex* values, that is, the proportion of actions which are not transitively ordered. The results show that block deordering never achieves a lower average *flex* value than EOG, and can achieve an average increase in *flex* of up to 0.15 (e.g., the *parcprinter* domain), and in several domains is able to relax plans that EOG cannot deorder at all.

2.3 Symmetry Breaking

In the context of planning problems, and indeed search and optimisation problems of many kinds, *symmetry* refers to the presence of multiple states or branches in a search space that are "equivalent" in the sense that an algorithm can explore only one of them and remain sound and complete. Symmetries can thus be exploited to reduce the amount of search needed to solve a problem. This exploitation, known as *symmetry breaking*, typically takes one of two approaches: *static symmetry breaking* extends the problem description with additional constraints that rule out redundant areas of the search space, and *dynamic symmetry breaking* modifies the search algorithm to avoid such areas.

A *syntactic symmetry* is an *automorphism* (Section 2.1.7.3) of a problem description, that is, a permutation of symbols that maps the description back to itself. Arguably the simplest form of symmetry in planning problems is *object symmetry*, which occurs when the problem contains interchangeable (combinations of) domain objects [44, 68, 109]. More complex symmetries result when variables and operator pre/postconditions can also be permuted [100, 118], or occur in structures implied by the problem description, such the automorphisms of the transition graph [110], and permutations of interchangeable ground actions [29].

As many POP optimisation tasks, including that of maximising flexibility (Section 2.2.2), can be naturally represented as constraint optimisation problems, this section will also discuss symmetries in the context of CSPs. And since a plan's *causal structure* (Definition 2.6), which is central to many POP optimisation approaches, can be expressed as a matrix of propositional variables, particular attention will be paid to lexicographic symmetry breaking techniques in graph [28, 88] and matrix models [42, 56].

This section will first describe a commonly-used, general-purpose technique for detecting and breaking syntactic symmetries, before describing some specialised approaches to symmetry breaking in matrix models, graph models, and classical planning.

2.3.1 Detecting and Breaking Syntactic Symmetries

A *syntactic symmetry*, broadly speaking, is an automorphism of a problem instance that preserves the validity and optimality of its solutions. For example, consider the SAT instance $\phi = (\mathrm{p}_1 \vee \mathrm{p}_2) \wedge (\neg\,\mathrm{p}_1 \vee \neg\,\mathrm{p}_2)$ and the permutation $\sigma = (\mathrm{p}_1\ \ \mathrm{p}_2)$. As swapping p_1 and p_2 in ϕ produces an identical formula, σ is an automorphism of ϕ. Additionally, σ preserves validity. A solution to ϕ is a substitution $\theta : \{\mathrm{p}_1, \mathrm{p}_2\} \to \{0, 1\}$, and a valid solution sets

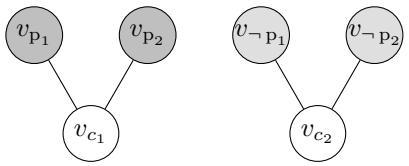


Figure 2.4: A coloured graph representation of the SAT formula $(\mathrm{p}_1 \vee \mathrm{p}_2) \wedge (\neg\,\mathrm{p}_1 \vee \neg\,\mathrm{p}_2)$.

exactly one variable to 1. As swapping p_1 and p_2 in any solution simply swaps their bindings, it follows that σ preserves validity. It is therefore a symmetry of ϕ.

In this example, σ is a *variable symmetry*, that is, it permutes variables but fixes constants. Common variations include *value symmetries*, which permute constants but fix values, and *variable/value symmetries*, which permute both.

A problem's symmetries form a *symmetry group* (Definition 2.1.7.2) that partitions the solution space into sets of equivalent, or *symmetrical* solutions: two solutions are symmetrical, that is, they belong to the same *symmetry class*, if one is the image of the other under one of the problem's symmetries. In the above example, the solutions $\theta_1 = \{\mathrm{p}_1 \backslash 0, \mathrm{p}_2 \backslash 1\}$ and $\theta_2 = \{\mathrm{p}_1 \backslash 1, \mathrm{p}_2 \backslash 0\}$ are symmetrical.

Detecting syntactic symmetries Crawford et al. [32] introduced a now standard (e.g., [68, 86, 100, 105]) technique that reduces symmetry detection to the problem of finding the automorphisms (Definition 2.16) of a coloured graph that captures the relevant aspects of the problem instance.

The colours and edges of the graph must be carefully constructed to ensure that its automorphisms correspond to the instance's symmetries. For example, a SAT instance can be transformed into coloured graph with three differently-coloured sets of variables representing the clauses, positive literals and negative literals, respectively, and edges connecting literals to any clauses in which they appear. Figure 2.4 depicts such a transformation of ϕ. The graph has a single automorphism, $(v_{\mathrm{p}_1}\ v_{\mathrm{p}_2})(v_{\neg\,\mathrm{p}_1}\ v_{\neg\,\mathrm{p}_2})$, which corresponds to the symmetry above $\sigma = (\mathrm{p}_1\ \mathrm{p}_2)$.

Lex-leader constraints Puget [103] showed that symmetries can be statically *broken* by selecting a single member of each symmetry class – the *canonical* solution – and adding *symmetry breaking constraints* to the problem description to rule out some or all non-canonical solutions. These constraints are *correct iff* they admit at least one element from each symmetry class, and *total* if they admit exactly one. A correct set of constraints that is not total is *partial*.

Crawford et al. [32] present the *lex-leader* method, a general technique for generating symmetry breaking constraints for variable symmetries under which the *lexicographic least* element of each symmetry class is considered canonical. Lexicographic ordering is a generalisation of alphabetic ordering that derives an ordering over lists of symbols from an ordering over their individual elements:

Definition 2.20. *If $\vec{x}$ and $\vec{y}$ are lists of symbols of length n, and $\prec$ is a total order over all elements appearing $\vec{x}$ and $\vec{y}$, then:*

- $\vec{x} =_L \vec{y}$ iff *for all* $1 \leq i \leq n$, $\vec{x}[i] = \vec{y}[i]$, *and*
- $\vec{x} \preceq_L \vec{y}$ iff $\vec{x} =_L \vec{y}$, *or there exists a* $1 \leq k < n$ *such that* $\vec{x}[k] \prec \vec{y}[k]$, *and* $\langle x_1, \ldots x_{k-1} \rangle =_L \langle y_1, \ldots y_{k-1} \rangle$.

Under the lex-leader approach, lexicographic comparison of solutions derives from a selected ordering of variables. If $\vec{x}$ is the selected variable order, then $\theta \preceq_{\mathrm{L}} \theta'$ *iff* $\theta(\vec{x}) \preceq_{\mathrm{L}} \theta'(\vec{x})$. For example, let $\langle \mathrm{p}_1, \mathrm{p}_2 \rangle$ be an ordering of the variables in SAT instance ϕ above. The two (symmetrical) solutions $\theta_1 = \{\mathrm{p}_1 \backslash 0, \mathrm{p}_2 \backslash 1\}$ and $\theta_2 = \{\mathrm{p}_1 \backslash 1, \mathrm{p}_2 \backslash 0\}$ can be lexicographically ordered by comparing the lists $\theta_1(\langle \mathrm{p}_1, \mathrm{p}_2 \rangle)$ and $\theta_2(\langle \mathrm{p}_1, \mathrm{p}_2 \rangle)$. As these resolve to 01 and 10, respectively, it follows that $\theta_1 \preceq_{\mathrm{L}} \theta_2$.

A symmetry σ is broken by ruling out any solution that is lexicographically greater than its symmetric equivalent, that is, all solutions θ must satisfy the constraint $\theta \preceq_{\mathrm{L}} \sigma(\theta)$. In the case of variable symmetries, this can be equivalently expressed as $\theta(\vec{x}) \preceq_{\mathrm{L}} \theta(\sigma(\vec{x}))$ (i.e., the bindings of $\vec{x}$ are on the left, and the bindings of the permuted variables are on the right). For example, the variable symmetry σ in the SAT example above can be broken with the following constraint:

$$\theta(\langle \mathrm{p}_1, \mathrm{p}_2 \rangle) \preceq_{\mathrm{L}} \theta(\langle \sigma(\mathrm{p}_1), \sigma(\mathrm{p}_2) \rangle).$$

For θ_1, this resolves to $01 \preceq_{\mathrm{L}} 10$, which holds, but for θ_2 it resolves to $10 \preceq_{\mathrm{L}} 01$, which does not.

While Crawford et al.'s original approach applied to variable symmetries, it has since been extended other symmetries such as value symmetries [42, 106, 131], problems with variable *and* value symmetries [107], variable/value symmetries, and symmetries acting on sets [131].

As the lex-leader approach places a constraint per symmetry, and symmetry groups can be exponential in size, total symmetry breaking with lex-leaders is, in general, infeasible. While in some cases all symmetries can be broken with a polynomial number of constraints (e.g., *all-different* problems [104] and some special cases of matrices [42]), most lex-leader implementations instead break only a polynomial subset of symmetries (typically the generating set). Shlyakhter [119] notes that the challenge with partial lex-leader methods is to find a set of constraints that is both *effective* and *compact*, that is, that removes as many non-canonical solutions as possible without being so large as to slow the search process. This is often achieved by exploiting the structure of the problem, and effective, specialised partial lex-leaders have been developed for structures such as relations, functions, directed acyclic graphs [119], undirected graphs [28, 88], matrices [42, 56], and SAT formulae [6]. The use of lex-leaders in matrices and graphs will now be examined in more detail.

2.3.2 Symmetry Breaking in Matrix Models

A *matrix model* is a CSP (Definition 2.10) with one or more matrices (Section 2.1.1) of variables. Such models occur frequently in CSPs [41, 43]. As rows and columns typically correspond to objects in the CSP problem

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{matrix}$$

(a) Matrix $\mathbf{M}_1$

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix}$$

(b) Matrix $\mathbf{M}_2$

Figure 2.5: Double Lex failure example. $\mathbf{M}_2$ is the result of transposing rows 1 and 2, and columns 2 and 3, and both satisfy Double Lex.

description, indistinguishable objects result in forms of variable symmetry known as *row symmetry* and *column symmetry*, in which entire rows or columns, respectively, of variables are pairwise interchangeable. More concretely, a symmetry between rows and i and j in a $n \times m$ matrix $\mathbf{M}$ takes the form $\sigma = (\mathbf{M}[i][1]\ \mathbf{M}[j][1]) \cdots (\mathbf{M}[i][m]\ \mathbf{M}[j][m])$. Column symmetries are defined similarly.

For example, the goal of the *Balanced Incomplete Block Design* problem [87] (BIBD, CSPLib problem 26[3]) is to divide n objects into m (possibly overlapping) sets of size k, called blocks, *s.t.* all objects appear in r blocks and all pairs of objects appear together in s blocks. A BIBD instance can be modelled as an $n \times m$ Boolean (0/1) matrix $\mathbf{M}$ *s.t.* $\mathbf{M}[i][j] = 1$ if object i is in block j. As objects and blocks are interchangeable, a BIBD instance has *total row and column symmetry* (i.e., all rows and all columns are interchangeable).

In group theoretic terms, the symmetries of a matrix with total row and column symmetry can be generated (Definition 2.12) by the $n+m-2$ symmetries that swap adjacent rows and columns. Equivalently, this group is the composition of symmetric groups (Definition 2.15) $S_R \times S_C$, where R and C contain the matrix rows and columns, respectively. As this group is of size $n!m!$, total symmetry breaking is infeasible. This section will outline two approaches to partially breaking symmetries in matrix models.

2.3.2.1 Double Lex

Flener et al. [42] present Double Lex, a lex-leader technique for partially breaking row and column symmetries in matrix models. Under this approach, matrices are lexicographically ordered by placing their rows end-to-end, that is, if $\mathbf{M}$ and $\mathbf{N}$ are equal-sized n-row matrices, then $\mathbf{M} \preceq_{\mathrm{L}} \mathbf{N}$ *iff* $\mathbf{M}[1]^\frown \cdots ^\frown \mathbf{M}[n] \preceq_{\mathrm{L}} \mathbf{N}[1]^\frown \cdots ^\frown \mathbf{N}[n]$.[4]

Under Double Lex, a matrix $\mathbf{M}$ is canonical *iff* it is the lexicographic least in its symmetry class under this ordering, and non-canonical matrices are ruled out with lex-leader constraints between certain pairs of interchangeable rows and columns. If an $n \times m$ matrix $\mathbf{M}$ has total row and column symmetry, then Double Lex orders the rows and columns in non-decreasing lexicographic order with the $m+n-2$ ordering constraints $\mathbf{M}[1] \preceq_{\mathrm{L}} \mathbf{M}[2], \ldots, \mathbf{M}[n-1] \preceq_{\mathrm{L}} \mathbf{M}[n]$, and $\mathbf{M}[][1] \preceq_{\mathrm{L}} \mathbf{M}[][2], \ldots \mathbf{M}[][m-1] \preceq_{\mathrm{L}} \mathbf{M}[][m]$. This can be naturally generalised to matrices with subsets of interchangeable rows and columns.

[3] http://www.csplib.org/

[4] Where $^\frown$ is the concatenation operator.

$$\begin{matrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

(a) Matrix $\mathbf{M}_3$

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{matrix}$$

(b) Matrix $\mathbf{M}_4$

Figure 2.6: Directed graphs $\mathbf{M}_3$ and $\mathbf{M}_4$ comprise an entire symmetry class with neither lexicographically ordered rows or columns.

Double Lex is a *partial* symmetry breaking constraint. While ordering rows and columns breaks all row symmetries and column symmetries [40], it is not guaranteed to break those that are a composition of a row and column symmetry. For example, consider an unconstrained 3×3 matrix model $\mathbf{M}$ with complete row and column symmetry. As solution $\mathbf{M}_2$ in Figure 2.5 is the result of swapping the first and second rows, and the second and third columns in $\mathbf{M}_1$, they are symmetrical. However, both satisfy Double Lex, meaning that at least one non-canonical solution has not been removed.

Katsirelos et al. [74] demonstrate that in the worst case, Double Lex leaves an exponential number of solutions in each symmetry class. Nevertheless, experimental results [42, 56, 74] show that Double Lex, in practice, breaks most row and column symmetry.

2.3.2.2 Snake Lex

Grayland et al. [56]'s Snake Lex is an alternative to Double Lex that uses a different ordering of variables to determine canonicity. It comprises a pair of constraints, *Row-Wise Snake Lex* and *Column-Wise Snake Lex*, that partially break total row and column symmetries, respectively.

Under Column-Wise Snake Lex, canonicity is defined *w.r.t.* to a *column-wise snake ordering*, an ordering of variables that "snakes" up and down columns. For an n-column matrix $\mathbf{M}$, this ordering is the vector $\vec{m}_1 \frown \cdots \frown \vec{m}_n$, where for $1 \leq i \leq n$, $\vec{m}_i = \mathbf{M}[][i]$ if i is odd, or is the reverse of $\mathbf{M}[][i]$ otherwise. Column symmetries are broken by requiring that the first column be lexicographically less than or equal to the second and third columns, that the reverse of the second column be lexicographically less than or equal to the reverse of the third and fourth columns, and so on.

Row-Wise Snake Lex is defined similarly: canonicity is determined by a *row-wise snake ordering*, which "snakes" back and forth along columns, and row symmetries are broken with similarly rotated lexicographic constraints.

The different definitions of canonicity that underpin Row-wise and Column-wise Snake Lex (i.e., row-wise and column-wise snake ordering, respectively) render the two constraints incompatible. Additionally, like Double Lex, Snake Lex is a partial symmetry breaking constraint that can leave an exponential number of solutions in each symmetry class [74]. Nevertheless, empirical evaluation [56, 74] over test instances with both row and column symmetry show that Snake Lex can, with a judicious choice of Row-wise or Snake-wise, improve on Double Lex by decreasing search time and pruning more non-canonical solutions.

2.3.3 Symmetry Breaking in Graph Models

A *graph model* is a CSP in which the variables form the adjacency matrix of a graph. For example, the problem of finding a minimum deorder or reorder of an n-step plan $\vec{a}$ (Definition 2.18) can be modelled as an n^2 graph model $\mathbf{M}$ *s.t.* $\mathbf{M}[i][j] = 1$ *iff* $\vec{a}[i] \prec \vec{a}[j]$ in the optimised POP, with optimality found by minimising $\sum \mathbf{M}$ and validity preserved by placing constraints over $\mathbf{M}$ derived from the POCL requirements (Definition 2.7).

Unlike matrix models, in which domain objects are associated with a row or column but not both, domain objects in graph models correspond to graph vertices, and are therefore associated with both a row *and* column of the adjacency matrix. This introduces subtleties not present in matrix models: interchangeable objects create symmetries that permute rows and columns simultaneously, and rows cannot be reordered without reordering the equivalent columns (nor *vice versa*).

This means that symmetries in graph models cannot, in general, be broken by simply ordering rows and/or columns, as doing so could rule out entire symmetry classes. For example, consider the directed (cyclic) graphs $\mathbf{M}_3$ and $\mathbf{M}_4$ in Figure 2.6. Neither have ordered rows or columns, and yet they comprise an entire of symmetry class: swapping any pair of vertices in $\mathbf{M}_3$ results in $\mathbf{M}_4$ and *vice versa*, and all other symmetries fix them both.

The special case of simple graphs (i.e., undirected with no self-loops) is an exception to this. As the rows in such graphs are identical to the columns, and the diagonal is zero, each symmetry class will contain at least one element with ordered rows and columns [28]. Thus, breaking symmetries by ordering rows or columns is applicable in this case.

2.3.3.1 Simple Graphs

Miller and Prosser [88] introduce a lex-leader constraint for simple graphs in the context of finding *diamond-free degree sequences* (i.e. graphs $\mathbf{M}$ with no diamond subgraphs where for $1 \leq i < |\mathbf{M}|$, $\sum \mathbf{M}[i] \geq \sum \mathbf{M}[i+1]$). Their approach is essentially the row-wise component of Double Lex: canonicity is defined *w.r.t.* to the same row-wise variable ordering, and non-canonical graphs are removed with lex-leaders between adjacent rows, (i.e., for an n^2 graph $\mathbf{M}$, $1 \leq i < n$, $\mathbf{M}[i] \preceq_{\mathrm{L}} \mathbf{M}[i+1]$).

Codish et al. [28] observe that swapping row and column i with row and column j in a simple graph $\mathbf{M}$ fixes the elements at $\mathbf{M}[i][i]$, $\mathbf{M}[i][j]$, $\mathbf{M}[j][i]$ and $\mathbf{M}[j][j]$, and so optimise Miller and Prosser's approach by excluding these fixed points from the lexicographic constraints. While these minimised lex-leaders are more effective (i.e., more non-canonical graphs are removed), they are no longer transitive (i.e., a lex-leader holding between rows i and j, and rows j and k does not imply that it holds between rows i and k). Thus, transitivity must be explicitly enforced with additional constraints. Symmetries are broken by requiring that for all $1 \leq i < j \leq n$, $\mathbf{M}[i] \restriction \{i,j\} \preceq_{\mathrm{L}} \mathbf{M}[j] \restriction \{i,j\}$ (where $\vec{t} \restriction S$ denotes the result of removing all elements from $\vec{t}$ at indexes in S). The effectiveness of this approach is confirmed by an experimental evaluation that closes several highly symmetric open problems in extremal graph theory.

2.3.3.2 Directed Acyclic Graphs

Shlyakhter [119] analyses a symmetry breaking technique for directed acyclic graphs (DAGs) that simply disallows edges from "higher" to "lower" vertices: if $\mathbf{M}$ is an n^2 adjacency matrix, then for $1 \leq i < j \leq n$, $\mathbf{M}[j][i] = 0$ must hold. Unlike other techniques presented here, it is not based on lexicographic ordering; indeed it is unclear how canonicity is defined in this context (if at all). Nevertheless, the observation that all symmetry classes contain at least one DAG that is topologically sorted *w.r.t.* any given vertex ordering demonstrates its correctness. As there can be more than one such DAG per class it is a partial symmetry breaking constraint, nevertheless experimental analysis shows that it is very effective in practice.

Interestingly, this symmetry breaking constraint also enforces acyclicity. Since acyclicity constraints typically require a cubic number of constraints, this demonstrates an alternative role for symmetry breaking: to reduce the size of the original problem constraints.

2.3.4 Symmetry Breaking in Classical Planning

Exploiting symmetries to increase the efficiency of domain-independent automated planners is a popular area of research. As in other fields, symmetry breaking is typically employed to shrink the search space by (either statically or dynamically) removing as many non-canonical solutions as possible. Beyond search space reduction, symmetry breaking in classical planning has been applied to a wide range of areas such as reachability analysis [115], grounding [121], task decomposition [2] and pruning redundant operators [134], however these will not be considered here.

This section will outline some techniques for search space reduction by breaking some (related) types of symmetry, namely *object symmetries*, the more general *structural symmetries*, *state transition symmetries* and *order symmetries*.

2.3.4.1 Object Symmetries

Interchangeable objects in a problem instance result in so-called *functional*, or *object symmetries*. This section will examine two commonly occurring types object symmetry, here termed *simple* and *complex*.

Simple object symmetries occur when pairs of individual objects are interchangeable, and can be detected by identifying objects with indistinguishable configurations in the initial state and goal. For example, in the *barman* instance in Section 1.1, hands LH and RH are functionally equivalent: both are initially empty, and neither appear in the goal or an operator definition. Swapping LH with RH thus results in an identical planning instance, and similarly modifying a plan will not alter its validity.

Complex object symmetries occur when *combinations* of objects are interchangeable, but individual objects are not. Consider a small *blocksworld* instance where initially block A is on B and C is on D, and the goal is to unstack all blocks. There are no simple object symmetries, however, A and B *as a pair* are interchangeable with C and D. While the exploitation of complex object symmetries can result in more effective symmetry breaking than simple object symmetries, their detection is a more complex

task that typically requires specialised graph automorphism software such as NAUTY [85].

Fox and Long [44] detect simple object symmetries with a straightforward analysis of the initial state and goal, and use the resulting sets of indistinguishable objects to define symmetries over actions. For example, in the *barman* domain, the actions $grasp(\mathsf{LH}, \mathsf{SH}_1)$ and $grasp(\mathsf{RH}, \mathsf{SH}_2)$ are interchangeable *iff* LH and RH, and SH_1 and SH_2, are interchangeable. Symmetries are broken by preventing the Graphplan-style planner from selecting symmetric alternative actions when backtracking. A limitation of this approach is that selecting an action referring to an object breaks the object symmetry. However, follow-up work [45] resolves this by identifying new symmetries that emerge during the search process.

Riddle et al. [109] eliminate simple object symmetries by reformulating the PDDL into a "bagged" representation, where sets of interchangeable objects are replaced with counter variables, and actions are updated to modify the counters rather than specific object instances. For example, interchangeable parcels in a logistics domain are replaced by counters for each location, and the action of loading a parcel into a truck is modified to decrement the depot counter and increment the truck counter. This replacement requires a stronger form of interchangeability than simple object symmetry: for replacements to be valid, actions must never need to discriminate between object instances. It is thus only applied when various criteria are met, for example, the objects cannot appear in negated literals, and predicates referring to the object type must be *single-valued* (e.g., a parcel cannot be in more than one truck).

Joslin and Roy [68] use Crawford et al. [32]'s approach to detect and break complex object symmetries in a CSP representation of a logistics planning problem. The initial state and goal are transformed into a coloured graph with a vertex for each object and literal, coloured by type and predicate symbol, respectively, and edges connecting objects to any literals in which they appear. The automorphisms of this graph are computed with NAUTY, translated back into symmetries of the planning instance, and encoded as lex-leaders. For example, if locations L_1 and L_2 are symmetrical, then a constraint such as $at(L_2, 1) \rightarrow at(L_1, 1)$ breaks the symmetry by preventing the solver from trying both L_1 and L_2 at step 1. While this technique only breaks symmetries in the first step of the plan, experiments show that it still results in quicker solve times with little overhead.

2.3.4.2 Structural Symmetries

Introduced by Pochter et al. [100] and later formalised by Shleyfman et al. [118], *structural symmetry* is a generalisation of object symmetry that also considers interchangeable variables, operator pre/postconditions and in some cases costs (e.g., [34]). Structural symmetries are detected by finding the automorphisms of the *problem description graph* (PDG): a coloured graph that encodes the structure of the planning instance's variables, values and operators. The power of structural symmetries lies in the fact that each of the PDG's automorphisms induces a symmetry of the instance's state transition graph, that is, symmetries of an exponentially large search space can be (partially) derived from a compact representation.

Pochter et al. [100] use structural symmetries to guide the progression of a state-space planner. Symmetries are dynamically broken by preventing the planner from visiting any two states with the same canonical form. However, as generating a canonical form is NP-hard [83], an approximate form is generated by applying various permutations and keeping the lowest state found. This speeds up the algorithm at the expense of less reduction in the search space.

This exploitation of structural symmetry results in an impressive reduction in execution time and/or node expansion, with little overhead from symmetry detection. Domshlak et al. [34] extend the approach by detecting further symmetries in the state transition graph as the search progresses, and Gnad et al. [54] successfully combine it with *star-topology decoupled search* [52] (a form of factored planning with a star-shaped sub-problem dependency graph).

Shleyfman et al. [118] study the interaction of symmetry breaking and heuristic search, and demonstrate that many heuristics, such as *optimal delete relaxation* and some of its approximations, *critical path*, and various *landmark*-based heuristics, are invariant under structural symmetry (i.e., they compute the same estimate for symmetric states). However, those that rely on arbitrary tie-breaking policies, such as *fast-forward* and *landmark-cut*, are not. Interestingly, Domshlak et al. [35] show *landmark-cut*'s variance can improve its informativeness: as symmetric states have symmetric landmarks, their heuristic values can be aggregated as search progresses.

2.3.4.3 State Transition Symmetries

A *state transition symmetry* is an automorphism of the state transition graph implied by a planning instance. More concretely, a *transition system* is a set of *transition relations* (i.e., partial functions over states), and a state transition symmetry is an automorphism that permutes both states and transitions but does map goal states to non-goal states, or *vice versa*.

Rintanen [110] addresses the problem of breaking state transition symmetries in the context of planning as satisfiability [75]. Typical approaches to breaking state transition symmetries cannot be directly applied to SAT planning: state-space search algorithms that generate a single successor state from each symmetry class (e.g., Emerson and Sistla [38] and Pochter et al. [100]) are too complex to be practically expressed in propositional logic, and approaches that place lexicographic constraints on states (e.g., Joslin and Roy [68]), do not generalise to sequences as successors of canonical states are not necessarily canonical.

Thus, lexicographic constraints are placed on sequences of transitions, rather than individual states at given time points. Two transition relations t_1 and t_2 are interchangeable in state s *iff* there is a state transition symmetry σ *s.t.* $\sigma(t_1) = t_2$ and $\sigma(s) = s$. A transition sequence $t_1, \ldots, t_n$ producing states $s_0, \ldots, s_n$ is canonical *iff* each t_i is the lexicographic least of all transitions with which it is interchangeable in s_{i-1}, and symmetries are broken with constraints limiting the applicability of non-canonical transitions. However, as detecting all transition symmetries is infeasible, the symmetries that are broken at each step are only simple object symmetries.

2.3.4.4 Order Symmetries

Order symmetries occur when the search space contains plans that differ only by a permutation of equivalent actions. This occurs frequently in the problem of plan de/reordering: if a plan contains multiple identical steps, then each of its reorders will have an exponential number symmetric variations that differ only by a swapping of these actions. Say et al. [116] (Section 2.2.2.4) break these symmetries with the DAG symmetry breaking constraint studied by Shlyakhter [119] (Section 2.3.3.2), and require that for all pairs of identical actions a_1, a_2 *s.t.* a_1 precedes a_2 in the original plan, $a_2 \not\prec a_1$ in the optimised POP.

As discussed in Section 2.2.1, partial-order planners such as Coles et al. [30]'s POPF explore a search space with nodes representing (possibly partially specified) POPs and edges representing plan refinement operations. Coles and Coles [29] present an extension to POPF that prevents it from exploring nodes with symmetrical ordering constraints. At each step, the node is encoded into a coloured graph with a node for each action, coloured by type, and an edge for each ordering constraint. The node is not expanded if the graph's canonical form has been previously seen.

More complex order symmetries occur when a plan comprises independent sub-sequences. As any interleaving of non-interfering sequences of actions will lead to the same state, a search space can contain an exponential number of symmetrical plans that differ only by a permutation of this interleaving. Valmari [127] eliminated these equivalent interleavings (in the context of software verification) by limiting search to specific subsets of the transition system termed *(strong and weak) stubborn sets*. Strong stubborn sets and similar partial-order reduction techniques have since been used extensively in heuristic planning [5, 24, 25, 53, 55, 59, 132, 133, 138], planning with resources [136], non-deterministic planning [137] and goal recognition [76]. While rarely referred to as such, the fact that these approaches exploit permutations of orderings means that they are certainly symmetry breaking techniques.

In the context of planning, a strong stubborn set contains, intuitively, actions needed to achieve (some part of) the goal from a given state, and any other actions that might conflict with an element of the set. Actions outside the set need not be executed right away and can be ignored in that state. While several formal definitions of strong stubborn sets exist, a typical example based on necessary enabling sets and action interference is presented below.

A *necessary enabling set* for action a in state s is a set of actions such that all plans $\langle a_1, \ldots, a_G \rangle$ that are executable in s and contain a also contain an element of the set before the first occurrence of a. When $a = a_G$, the set is known as a *disjunctive action landmark*. Two actions *interfere* if their postconditions conflict, or one threatens to the other's preconditions. An SSS for state s is a set of actions T_s *s.t. (i)* T_s is a necessary enabling set for o_G in s, *(ii)* if $a \in T_s$ is not executable in s, then T_s is a necessary enabling set for a in s, and *(iii)* if $a \in T_s$, a and a' interfere, and there is a plan $\langle a, \ldots, a', \ldots, a_G \rangle$ that is executable in s, then $a' \in T_s$.

Order symmetries are broken dynamically. At each step in the search, actions outside T_s are not considered, however, as every optimal solution to the planning instance will have a permutation that is not pruned, the completeness of the search is ensured.

This literature review introduced a number of fundamental background concepts, and then presented an overview of relevant research into partial-order planning, POP optimisation, and symmetry breaking.

The plan optimisation problems addressed in this thesis can be expressed as hybrid matrix/graph models. However, they frequently exhibit a form of symmetry that is more general than row/column symmetry (Section 2.3.2), and thus beyond the scope of standard approaches such as Double Lex [42] and Snake Lex [56] (Sections 2.3.2.1 and 2.3.2.2). Thus, Chapter 3 introduces a taxonomy of symmetry groups (Section 2.1.7) to define these more general symmetries, and introduces an extension of Double Lex that can (partially) break them. Additionally, unlike standard approaches to breaking symmetries in graph models (Section 2.3.3), Multi Lex can also lexicographically break symmetries in directed or undirected (single-edged) graphs, including those with self-loops.

Both Chapter 4 and Chapter 5 present generalised forms of Bäckström [8]'s POP optimality definitions (Section 2.2.2.1). Chapter 4 extends them to account for POPs with different variable bindings, and extends the MaxSAT-based technique (Section 2.1.5) of Muise et al. [91] (Section 2.2.2.3) to find bindings that minimise ordering constraints. Chapter 5 introduces a generalised form of plan that need not completely specify either the orderings or variable bindings, and extends Bäckström [8]'s definitions to compare the optimality of plans of this type. A parameterised complexity analysis (Section 2.1.3) resolves the central challenge of these plans, namely balancing flexibility and tractability, by limiting the search to plans of bounded treewidth (Section 2.1.4).

In both cases, the search for flexible plans is optimised by extending Pochter et al. [100]'s notion of structural symmetry (Section 2.3.4.2) to POPs, and then simultaneously breaking object symmetries (Section 2.3.4.1) and order symmetries (Section 2.3.4.4) by limiting the search to POPs with canonical causal structures (Section 2.1.2.2). Chapter 4 breaks symmetries statically with Crawford et al. [32]'s standard technique (Section 2.3.1), while Chapter 5 breaks them dynamically with an approach similar that of Pochter et al. [100] and Coles and Coles [29] (Section 2.3.4.4. Both methods use the novel symmetry breaking predicate defined in Chapter 3.

CHAPTER 3

Breaking Generalised Symmetries in Matrix and Graph Models

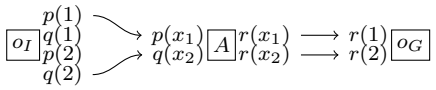
3.1 Introduction

This thesis is primarily concerned with the post-processing of plans in order to optimise various flexibility criteria. However, as in many optimisation problems, the search for an optimally flexible plan can be hindered by an exponential number of symmetrical (non) solutions that, in this case, differ only by an irrelevant permutation of functionally equivalent domain objects and operators, and represent an equal number of different ways to achieve the goal. Thus, before addressing the plan optimisation problems directly, this chapter focuses on the important problem of identifying and removing symmetrical plans from the search space.

The optimisation problems that will be addressed in Chapters 4 and 5 can be naturally expressed as hybrid *matrix/graph models* (Sections 2.3.2 and 2.3.3). Standard symmetry breaking techniques for such models, such as Double Lex [42] and Snake Lex [56], are only applicable to *row symmetry* and *column symmetry*, in which rows and columns are *individually* interchangeable. However, plan optimisation problems frequently exhibit more complex symmetries in which *combinations of rows and/or columns are interchangeable, but individual rows and columns are not*. These more general symmetries, termed here *multi-row-column symmetries*, cannot be broken with standard approaches. Thus, this chapter defines a taxonomy of generalised symmetries, and introduces Multi Lex, a novel, compact and effective extension of Double Lex that can partially break such symmetries in matrix models and arbitrary graph models, including directed graphs with self-loops. Empirical evaluation reveals that the factorial number of symmetrical solutions introduced by generalised symmetries are nearly entirely removed by Multi Lex.

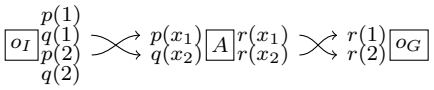
As multi-row-column symmetries are not confined to plan optimisation problems, but occur in many CSP[1] [1] benchmark domains, this chapter will discuss symmetry breaking in the broader context of matrix and graph models, rather than plan optimisation specifically. Chapters 4 and 5 will apply the findings to the problem of optimising plan flexibility.

[1] http://www.csplib.org



	$p(x_1)$	$q(x_2)$	$r(1)$	$r(2)$
$p(1)$	1	0	0	0
$q(1)$	0	0	0	0
$p(2)$	0	0	0	0
$q(2)$	0	1	0	0
$r(x_1)$	0	0	1	0
$r(x_2)$	0	0	0	1

(a) Plan P_1.



	$p(x_1)$	$q(x_2)$	$r(1)$	$r(2)$
$p(1)$	0	0	0	0
$q(1)$	0	1	0	0
$p(2)$	1	0	0	0
$q(2)$	0	0	0	0
$r(x_1)$	0	0	0	1
$r(x_2)$	0	0	1	0

(b) Plan P_2.

Figure 3.1: The causal structures of two symmetrical plans from a synthetic domain. Plan P_2 is obtained from P_1 by swapping rows $p(1)$ and $q(1)$ and column $r(1)$ with rows $p(2)$ and $q(2)$ and column $r(2)$, respectively.

3.1.1 Example

Much of the later work in this thesis is concerned with making plans more flexible by modifying their *causal structure*. A plan's causal structure is an implicit set of unthreatened *causal links*, that is, mappings from *producers* (i.e., initial state facts and operator postconditions) to causally dependent *consumers* (i.e., goals and operator preconditions), that determine the plan's validity (Definition 2.7). Chapter 4 studies the problem of finding a causal structure that minimises the number of ordering constraints, and in Chapter 5, plans are relaxed further by providing each consumer with a set of possible producers. While the details will be left for later chapters, the aspect common to both that is relevant here is that *a plan's causal structure can be represented as a matrix over which symmetries frequently occur*.

For example, Figure 3.1 depicts two plans from a synthetic domain. The plans contain a single step, A, and the initial state and goal are represented by o_I and o_G, respectively. Causal structures are represented by both arrows and Boolean (0/1) matrices in which producers and consumers are associated with rows and columns, respectively. The causal structures have a symmetry deriving from an *object symmetry* (Section 2.3.4): as objects 1 and 2 are interchangeable (swapping them leaves the initial state and goal unchanged), rows $p(1)$ and $q(1)$ and column $r(1)$ can be swapped with rows $p(2)$ and $q(2)$ and column $r(2)$, respectively. Plan P_2 is the result of permuting plan P_1 in this way.

This symmetry permutes multiple rows and columns simultaneously, that is, it is a *multi-row-column symmetry*, and *cannot be broken with constraints between individual rows*. For example, requiring lexicographic ordering between rows $p(1)$ and $p(2)$, $q(1)$ and $q(2)$, and columns $r(1)$ and $r(2)$ would rule out both plans.

Symmetries in casual structures can also derive from multiple instances of the same operator type. As the roles that such operators play in a plan's causal structure can be swapped, interchangeable operators with at least one precondition and postcondition create symmetries in which rows and columns are swapped simultaneously.

This chapter will continue as follows. Section 3.2 defines a hierarchy of generalised symmetries, Section 3.3 formally defines Multi Lex, and Section 3.4 examines the number of symmetrical solutions introduced by multi-row-column symmetries, and evaluates the ability of Multi Lex to remove them.

3.2 Generalised Symmetries in Matrix and Graph Models

In the context of *matrix models* (i.e., CSPs containing one or more matrices of variables), a *variable symmetry* is a permutation of variables that leaves the model's constraints unchanged, and preserves the validity and optimality of its solutions (Section 2.3.1). Arguably the most common forms of symmetry in matrix models are *row symmetries* and *column symmetries*, special cases of variable symmetry that result from individually interchangeable domain objects, and swap a pair of rows or columns of variables, respectively. These can be generalised into *multi-row symmetries* and *multi-column symmetries*, in which multiple rows or columns are simultaneously swapped, and *multi-row-column symmetries*, which, as shown in Section 3.1.1 above, swap multiple pairs of rows and columns.

As *graph models* (i.e., CSPs that model the adjacency matrix of a graph) associate vertices with both a row and column of an adjacency matrix, vertices v_i and v_j being interchangeable results in a special case of multi-row-column symmetry that swaps rows i and j, and columns i and j. These symmetries, termed here *vertex symmetries*, can be naturally generalised to *multi-vertex symmetries*. This section will formally define this symmetry hierarchy, and for each type of symmetry, a corresponding category of symmetry group (Section 2.1.7.2).

As these symmetries are not confined to plan optimisation problems– a brief survey of CSPLib reveals that they can occur in many matrix and graph-based CSP domains–illustrative examples will be taken from combinatorial and graph theoretic problems. In all examples, a matrix of variables exhibits symmetries in which combinations of rows and/or columns are interchangeable, but individual rows and columns are not.

3.2.1 Multi-Row Symmetries and Multi-Column Symmetries

All of the symmetry types outlined above swap entire rows and/or columns of variables. To make their formal definitions more concise, the notation $\sigma_R^{i,j}$ and $\sigma_C^{i,j}$ is used to denote a permutation that exchanges the variables in a pair of rows or columns, respectively, and fixes all others:

Definition 3.1. *Let* $\mathbf{M}$ *be an* $r \times c$ *matrix. Then:*

- *for* $1 \leq i, j \leq r$, $\sigma_R^{\mathbf{M},i,j} = (\mathbf{M}[i][1] \ \ \mathbf{M}[j][1]) \cdots (\mathbf{M}[i][c] \ \ \mathbf{M}[j][c])$, *and*
- *for* $1 \leq i, j \leq c$, $\sigma_C^{\mathbf{M},i,j} = (\mathbf{M}[1][i] \ \ \mathbf{M}[1][j]) \cdots (\mathbf{M}[r][i] \ \ \mathbf{M}[r][j])$.

When clear from context, $\mathbf{M}$ is excluded from the notation. For example, if $\mathbf{M} = \langle\langle x_1, x_2, x_3\rangle, \langle x_4, x_5, x_6\rangle, \langle x_7, x_8, x_9\rangle\rangle$ is a 3^2 matrix, then

0 0 0 1	0 1 0 0	1 0 0 0
0 1 0 0	0 0 0 1	0 0 1 0
(a) Deck $\mathbf{D}_1$.	(b) Deck $\mathbf{D}_2$.	(c) Deck $\mathbf{D}_3$.

Figure 3.2: Three symmetrical solutions to a *Vessel Loading* example in which two crates must be placed in non-adjacent locations.

$\sigma_R^{1,2} = (x_1\ x_4)(x_2\ x_5)(x_3\ x_6)$ is a permutation that swaps each variable in the first row with the corresponding variable in the second. Note that in all cases, $\sigma_R^{i,j} = \sigma_R^{j,i}$ and $\sigma_C^{i,j} = \sigma_C^{j,i}$, and when $i = j$, $\sigma_R^{i,j} = \sigma_C^{i,j} = \sigma_{id}$.

Row symmetries and *column symmetries* are special cases of variable symmetry that swap pairs of rows or columns, respectively:

Definition 3.2. *Let* $\mathbf{M}$ *be a matrix with symmetry* σ*. Then:*

- σ *is a* **row symmetry** iff *there exists an* i, j s.t. $\sigma = \sigma_R^{i,j}$*, and*
- σ *is a* **column symmetry** iff *there exists an* i, j s.t. $\sigma = \sigma_C^{i,j}$*.*

A *multi-row symmetry* is a generalised row symmetry that swaps multiple pairs of rows simultaneously. It is expressible as the composition of multiple row symmetries, with the condition that any given row is swapped at most once. Column symmetries can be similarly generalised into *multi-column symmetries*:

Definition 3.3. *Let* $\mathbf{M}$ *be a matrix with symmetry* σ*. Then:*

- σ *is a* **multi-row symmetry** iff *there exist two disjoint sets of row indices* $i_1, \ldots, i_n$ *and* $j_1, \ldots, j_n$ s.t. $\sigma = \sigma_R^{i_1,j_1} \circ \cdots \circ \sigma_R^{i_n,j_n}$*, and*
- σ *is a* **multi-column symmetry** iff *there exist two disjoint sets of column indices* $i_1, \ldots, i_n$ *and* $j_1, \ldots, j_n$ s.t. $\sigma = \sigma_C^{i_1,j_1} \circ \cdots \circ \sigma_C^{i_n,j_n}$*.*

For example, the goal of the *Vessel Loading* problem (CSPLib problem 8), is to find an arrangement of crates on a rectangular deck that satisfies distance constraints. An instance can be modelled as a matrix $\mathbf{D}$ containing the crate type at each location. Figure 3.2 shows three symmetrical solutions to a simple example in which two crates of the same type must be placed in non-adjacent locations on a 2×4 deck. As reflecting the deck about its horizontal or vertical axis will produce an equally valid solution, this instance has a row symmetry $\sigma_R^{1,2}$ that swaps the rows, and a multi-column symmetry $\sigma_C^{1,4} \circ \sigma_C^{2,3}$ that simultaneously swaps columns one and two with columns four and three, respectively. Solutions $\mathbf{D}_2$ and $\mathbf{D}_3$ are the result of applying these two symmetries, respectively, to $\mathbf{D}_1$.

Other CSP domains with multi-row and/or multi-column symmetry include *Balanced Academic Curriculum*, *Wagner-Whitin Distribution*, *N-Queens* (and its variations), *Travelling Tournament* and *Layout* (CSPLib problems 30, 40, 54, 68 and 132).

$b_1 = \langle 2, \{t_1, t_2\}\rangle$
$b_2 = \langle 2, \{t_3, t_4\}\rangle$
$b_3 = \langle 1, \{t_1, t_3\}\rangle$

(a) Bids b_1–b_3.

	t_1	t_2	t_3	t_4
b_1	1	1	0	0
b_2	0	0	0	0
b_3	0	0	0	0

(b) Solution $\mathbf{W}_1$.

	t_1	t_2	t_3	t_4
b_1	0	0	0	0
b_2	0	0	1	1
b_3	0	0	0	0

(c) Solution $\mathbf{W}_2$.

Figure 3.3: Two optimal solutions to a *Winner Determination* problem comprising three bids over four items.

3.2.2 Muli-Row-Column Symmetries

A *multi-row-column symmetry* is a further generalisation that simultaneously swaps multiple rows and columns. These symmetries can be expressed as the composition of a multi-row symmetry and a multi-column symmetry:

Definition 3.4. *If* $\mathbf{M}$ *is a matrix with symmetry* σ*, then* σ *is a* **multi-row-column symmetry** iff *it is expressible as the composition of a multi-row symmetry and a multi-column symmetry.*

For example, the *Winner Determination* problem (CSPLib problem 53) simulates an auction in which numeric bids are placed for (possibly overlapping) sets of items, and the aim is to find a set of disjoint offers that maximises the sale price. An instance with r bids over c items can be modelled as an $r \times c$ matrix where $\mathbf{W}[i][j] = 1$ *iff* bid b_i has been accepted for item t_j. Figure 3.3 depicts a small example comprising two bids over four items, and two optimal solutions. As swapping symbols b_1, t_1 and t_2 with b_2, t_3 and t_4, respectively, in Figure 3.3a leaves the instance unchanged, this instance has a multi-row-column symmetry $\sigma = \sigma_R^{1,2} \circ \sigma_C^{1,3} \circ \sigma_C^{2,4}$. Applying σ to $\mathbf{W}_1$ produces the equally valid and optimal $\mathbf{W}_2$.

Other matrix models with multi-row-column symmetry include *Warehouse Allocation*, *Tank Allocation*, *Interview Assignment*, *Balanced Nursing Workload* and *Target Tracking* (CSPLib problems 34, 51, 62, 69 and 72).

3.2.3 Multi-Vertex Symmetries

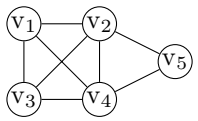
As vertices in graph models are associated with both a row and column in the adjacency matrix, interchangeable vertices result in special cases of multi-row-column symmetry, termed here *vertex symmetries*, in which rows i and j are swapped *iff* columns i and j are also:

Definition 3.5. *Let* $\mathbf{M}$ *be an* n*-vertex graph model with symmetry* σ*. Then,* σ *is a* **vertex symmetry** iff *there exists an* i, j s.t. $\sigma = \sigma_R^{i,j} \circ \sigma_C^{i,j}$.

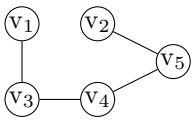
A *multi-vertex symmetry* is a generalised vertex symmetry in which multiple pairs of vertices are simultaneously swapped:

Definition 3.6. *Let* $\mathbf{M}$ *be an* n*-vertex graph model with symmetry* σ*. Then,* σ *is a* **multi-vertex symmetry** iff *there exist two disjoint sets of indices* $i_1, \ldots, i_n$ *and* $j_1, \ldots, j_n$ s.t. $\sigma = \sigma_R^{i_1,j_1} \circ \cdots \circ \sigma_R^{i_n,j_n} \circ \sigma_C^{i_1,j_1} \circ \cdots \circ \sigma_C^{i_n,j_n}$.

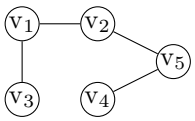
For example, Figure 3.4 depicts two solutions to an instance of the *Hamiltonian Path* problem, which asks whether a graph has a path that



(a) Graph $\mathbf{G}$.



(b) Hamiltonian path $\mathbf{P}_1$.



(c) Hamiltonian path $\mathbf{P}_2$.

0	0	1	0	0
0	0	0	0	1
1	0	0	1	0
0	0	1	0	1
0	1	0	1	0

(d) Adjacency matrix of path $\mathbf{P}_1$.

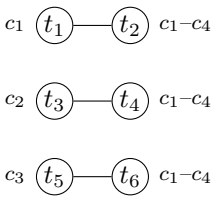
1	0	0	1	0
0	0	1	0	1
0	0	1	0	0
0	0	0	0	1
0	1	0	1	0

(e) Intermediate matrix $\mathbf{P}'$.

0	1	1	0	0
1	0	0	0	1
1	0	0	0	0
0	0	0	0	1
0	1	0	1	0

(f) Adjacency matrix of path $\mathbf{P}_2$.

Figure 3.4: Two solutions to a *Hamiltonian Path* problem.



(a) Tank configuration.

	c_1	c_2	c_3	c_4
t_1	1	0	0	0
t_2	0	0	0	0
t_3	0	1	0	0
t_4	0	0	0	0
t_5	0	0	1	0
t_6	0	0	0	1

(b) Solution $\mathbf{A}_1$.

	c_1	c_2	c_3	c_4
t_1	1	0	0	0
t_2	0	0	0	0
t_3	0	1	0	0
t_4	0	0	0	1
t_5	0	0	1	0
t_6	0	0	0	0

(c) Solution $\mathbf{A}_2$.

Figure 3.5: Two optimal solutions to a *Tank Allocation* instance comprising six tanks and four chemicals.

visits each vertex exactly once. As swapping v_1 and v_2 with v_3 and v_4, respectively, leaves $\mathbf{G}$ unchanged, this instance has a multi-vertex symmetry $\sigma = \sigma_R^{1,3} \circ \sigma_R^{2,4} \circ \sigma_C^{1,3} \circ \sigma_C^{2,4}$ that swaps rows and columns one and two with rows and columns three and four. Path $\mathbf{P}_2$ is the image of $\mathbf{P}_1$ under σ. To make this transformation clearer, adjacency matrix $\mathbf{P}'$ in Figure 3.4e depicts an "intermediate" state, in which rows one and two have been swapped with rows three and four (as shaded). Swapping the columns in $\mathbf{P}'$ (as outlined) completes the transformation into $\mathbf{P}_2$.

More complex graph models with multi-vertex symmetry include *Synchronous Optical Networking*, *Diameter and Degree Bounded Network Design Problem* and *Transshipment* (CSPLib problems 56, 71 and 83).

3.2.4 Generalised Symmetry Groups

In the examples above, multi-row-column symmetries occur when a matrix or graph model contains interchangeable combinations of domain objects. The resulting symmetries that swap the rows and/or columns associated with a pair of such combinations can be closed under composition (Definition 2.11) to form the symmetric group (Definition 2.15) over the combinations. If there are n combinations, this group can (as with all symmetric groups) be generated by $n-1$ symmetries, where each σ_i swaps all rows and columns in combination i with their counterparts in combination $i+1$.

For example, Figure 3.5 (ignoring highlighted cells for now) shows two symmetrical solutions to a simple instance of the *Tank Allocation* problem (CSPLib problem 51), in which an assignment of chemicals to tanks must be found that satisfies adjacency and tolerance constraints. The instance comprises six tanks $t_1, \ldots, t_6$, and four chemicals $c_1, \ldots, c_4$. The tank configuration is in Figure 3.5a, with tanks annotated with their tolerances, and chemicals c_1–c_3 cannot be stored in adjacent tanks.

This instance contains three combinations of interchangeable objects: $\langle t_1, t_2, c_1 \rangle$, $\langle t_3, t_4, c_2 \rangle$, and $\langle t_5, t_6, c_3 \rangle$. Swapping the elements of any one with their counterparts in any other leaves Figure 3.5 unchanged. This instance's symmetry group Γ is thus, essentially, the symmetric group over the three combinations of objects, and can be generated from $\Sigma = \{\sigma_1, \sigma_2\}$, where $\sigma_1 = \sigma_R^{1,3} \circ \sigma_R^{2,4} \circ \sigma_C^{1,2}$ swaps the first and second combinations, and $\sigma_2 = \sigma_R^{3,5} \circ \sigma_R^{4,6} \circ \sigma_C^{2,3}$ swaps the second and third.

More formally, symmetry groups of this type are termed *multi-row-column symmetry groups*, and occur when a subset of a matrix or graph model's rows and/or columns can be partitioned into equal-sized, interchangeable combinations:

Definition 3.7. *Let* $\mathbf{M}$ *be a matrix or graph model with symmetry group* Γ*. Then,* Γ *is a* **multi-row-column symmetry group** *over* $\mathbf{M}$ iff *there exist* n *equal-length, disjoint lists of row indices* $\vec{r}_1, \ldots, \vec{r}_n$*, and* n *equal-length, disjoint lists of column indices* $\vec{c}_1, \ldots, \vec{c}_n$*, and* Γ *has a generator* $\Sigma = \{\sigma_1, \ldots, \sigma_{n-1}\}$ *where for* $1 \leq i < n$*,* σ_i *is a multi-row-column symmetry as follows:*

$$\sigma_i = \circ \{\sigma_R^{j,k} : j = \vec{r}_i[n], k = \vec{r}_{i+1}[n], 1 \leq n \leq |\vec{r}_i|\} \circ \\ \circ \{\sigma_C^{j,k} : j = \vec{c}_i[n], k = \vec{c}_{i+1}[n], 1 \leq n \leq |\vec{c}_i|\}.$$

When the result of n interchangeable combinations of rows and/or columns, multi-row-column symmetry groups are isomorphic to S_n and are thus of size $n!$.

Special cases arise when Σ contains more specific symmetry types. For example, when Σ comprises multi-row symmetries (i.e., all $|\vec{c}_i| = 0$), Γ is termed a *multi-row symmetry group*, and when $\mathbf{M}$ is a graph model and Σ comprises multi-vertex symmetries (i.e., all $\vec{r}_i = \vec{c}_i$), Γ is a termed a *multi-vertex symmetry group*, respectively.

A natural addition to this taxonomy are *multi-row-and-multi-column symmetry groups*, in which multiple rows and columns are swapped, but rows and columns need not be swapped simultaneously (as in the *Vessel Loading* example in Figure 3.2). Such groups are expressible as the direct product (Definition 2.13) $\Gamma = \Gamma_r \times \Gamma_c$ of a multi-row symmetry group Γ_r and a multi-column symmetry group Γ_c.

3.2.5 Symmetry Hierarchy

Figure 3.6 depicts the hierarchy of the symmetry groups presented above, with arrows leading from less to more general groups. Those shaded in grey have been previously studied (Sections 2.3.2 and 2.3.3), and the remainder are defined above. Generalisations from single to multiple interchangeable rows, columns, or vertices are marked with dotted lines.

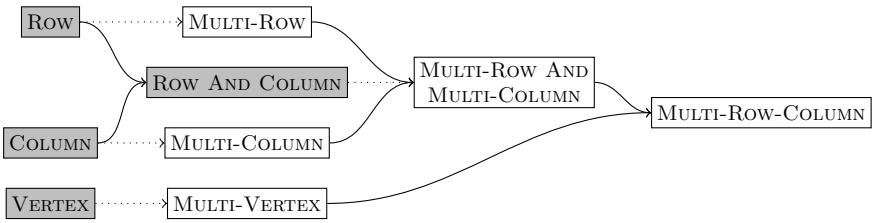


Figure 3.6: Hierarchy of generalised symmetry groups. Arrows lead from less to more general symmetries, with dotted lines indicating a generalisation from single to multiple interchangeable rows and/or columns.

The more general symmetry groups can be considered "weaker" in that they result in fewer symmetrical solutions. Intuitively, this is because more general symmetries are "implied" by less general ones: if all rows are interchangeable, then any equal-sized combinations will be also, but of course the opposite does not hold.

Less intuitively, as generalised symmetries are the composition of row and/or column symmetries (Definitions 3.2 and 3.2), and groups are closed under composition, any symmetry group resulting from interchangeable combinations of rows and/or columns must be a subgroup (Definition 2.14) of the one created when those same rows and columns are individually interchangeable. Since two solutions are symmetrical *w.r.t.* to a symmetry group *iff* one is the image of the other under an element of the group, if $\Gamma_1 \leq \Gamma_2$, then Γ_1 cannot introduce more symmetry than Γ_2: any two solutions symmetrical under Γ_1 must also be symmetrical under Γ_2, but the opposite need not hold.

3.3 Breaking Generalised Symmetries with Multi Lex

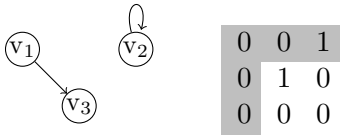
Standard approaches to symmetry breaking in matrix and graph models are too strong for the generalised symmetries introduced above. Thus, this section introduces Multi Lex, an extension of Double Lex that breaks multi-row-column symmetries (and by extension, multi-row, multi-column, and multi-vertex symmetries) by extending the model with lexicographic constraints (Definition 2.20).

3.3.1 Multi-Row-Column Symmetry Example

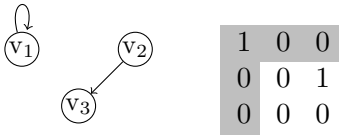
For example, the *Tank Allocation* example in Figure 3.5, has a multi-row-column symmetry group generated by $\sigma_1 = \sigma_R^{1,3} \circ \sigma_R^{2,4} \circ \sigma_C^{1,2}$ and $\sigma_2 = \sigma_R^{3,5} \circ \sigma_R^{4,6} \circ \sigma_C^{2,3}$. Solution $\mathbf{A}_1$ is in canonical form, however, attempting to break σ_2 with Double Lex by requiring that, for example, $\mathbf{A}[3] \preceq_L \mathbf{A}[5]$, $\mathbf{A}[4] \preceq_L \mathbf{A}[6]$ and $\mathbf{A}[][2] \preceq_L \mathbf{A}[][3]$[2], would rule out both $\mathbf{A}_1$ and $\mathbf{A}_2$. Indeed, Double Lex would rule out all solutions from this symmetry class, indicating that it is *too strong to apply to this form of symmetry.*

However, Double Lex can be generalised to break multi-row-column symmetries. To explain the generalisation, it is helpful to consider an al-

[2] As $\mathbf{A}$ is a 0/1 Boolean matrix, lexicographic comparison amounts to comparison of binary numbers.



(a) Graph $\mathbf{G}_1$.



(b) Graph $\mathbf{G}_2$.

Figure 3.7: Symmetrical solutions to DAG model, with self-loops, in which v_1 and v_2 must have an out-degree of at least one.

ternative definition of Double Lex. Under the standard definition, if $\mathbf{M}$ is a matrix with row symmetry $\sigma_R^{i,j}$ where $i < j$, Double Lex requires that $\mathbf{M}[i] \preceq_L \mathbf{M}[j]$. Alternatively, the left-hand side of the constraint, $\mathbf{M}[i]$, can be defined as the list of all variables in $\mathbf{M}$, in their original order, that are mapped into a higher row by $\sigma_R^{i,j}$. As $\sigma_R^{i,j}(\mathbf{M}[i]) = \mathbf{M}[j]$, the right-hand side is simply the image of the left. Multi Lex breaks multi-row-column symmetries with lex-leader constraints constructed in the same way. Each constraint is of the form $\vec{v} \preceq_L \sigma(\vec{v})$, where $\vec{v}$ contains all variables that are mapped into a higher row or column by σ, ordered as they would be if the matrix's rows were placed end-to-end.

Returning to Figure 3.5, as σ_2 above swaps rows 3 and 5, rows 4 and 6 and columns 2 and 3, $\vec{v}$ in this case contains the variables in the matrix's third and fourth rows, and second column (as highlighted in Figure 3.5b). The variables in $\vec{v}$ are ordered as they would be if the matrix's rows were placed end to end, that is, $\vec{v}$ contains the second element of the first and second rows, then the entire third and fourth rows, and then the second element of the fifth and sixth rows. When breaking σ_2 in $\mathbf{A}_1$, Multi Lex compares the highlighted cells in Figure 3.5b with the highlighted cells in Figure 3.5c (since $\sigma_2(\mathbf{A}_1) = \mathbf{A}_2$). As 000100000000 $\prec_L$ 000100010000, Multi Lex holds for $\mathbf{A}_1$, but not $\mathbf{A}_2$.

Due to the factorial size of multi-row-column symmetry groups, Multi Lex only explicitly breaks the symmetries found in the symmetry group's generating set. Thus, like Double Lex, Multi Lex trades tractability for completeness, meaning that while it holds for canonical solutions (i.e., it is *correct*), there may be non-canonical solutions for which it also holds.

3.3.2 DAG with Self-Loops Example

Unlike the standard graph symmetry breaking techniques discussed in Section 2.3.3, the approach described above can also break symmetries in directed graphs with self-loops. Graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ in Figure 3.7 are solutions to a three-vertex DAG model with the constraint that vertices v_1 and v_2 have an out-degree of at least one. As v_1 and v_2 are interchangeable, $\mathbf{G}_1$ and $\mathbf{G}_2$ are symmetrical under the vertex symmetry $\sigma = \sigma_R^{1,2} \circ \sigma_C^{1,2}$.

Multi Lex breaks this symmetry with a constraint of the form $\vec{v} \preceq_L \sigma(\vec{v})$, where $\vec{v}$ contains all variables in the first row, and then the first elements of the second and third rows. When applied to $\mathbf{G}_1$, Multi Lex compares the highlighted cells in Figure 3.7a with the highlighted cells in Figure 3.7b (since $\sigma(\mathbf{G}_1) = \mathbf{G}_2$). As 00100 $\prec_L$ 10000, Multi Lex holds for $\mathbf{G}_1$, but not $\mathbf{G}_2$. Multi Lex will now be defined more formally.

3.3.3 Canonical Matrices

As Multi Lex aims to remove as many non-canonical solutions as possible, canonicity in this context must be defined. As is standard, the lexicographic least solution in each symmetry class is selected as the canonical form, with the lexicographic ordering deriving from a *row-wise ordering* of variables:

Definition 3.8. *The* **row-wise ordering** *of a matrix* $\mathbf{M}$ *is the vector* $\mathbf{M}_R = \mathbf{M}[1]^\frown \cdots ^\frown \mathbf{M}[n]$.

A solution is a substitution θ (Section 2.1.1) that maps variables to constants, and is *canonical iff* the resulting matrix is lexicographically less than or equal to its image under all symmetries:

Definition 3.9. *If* $\mathbf{M}$ *is a matrix of variables with symmetry group* Γ*, and solution* θ *is a ground substitution that is complete* w.r.t. *to the variables in* $\mathbf{M}$*, then:*

$$\mathsf{can}(\mathbf{M}, \Gamma, \theta) \stackrel{\text{def}}{=} \bigwedge_{\sigma \in \Gamma} \theta(\mathbf{M}_R) \preceq_L \theta(\sigma(\mathbf{M}_R)).$$

The canonicity definition above compares lists of constants. On the left-hand side, the variables in $\mathbf{M}$ have been placed in a row-wise ordering and then replaced by their bindings under θ, and on the right-hand side the variables have been permuted by σ before being ordered and replaced.

3.3.4 Breaking Multi-Row-Column Symmetries

Multi Lex posts a lex-leader constraint for each element of the symmetry group's generating set. For each generator σ, a vector is constructed by limiting the matrix's row-wise ordering to variables in a row or column that σ maps to a higher row or column. The lex-leader requires that this vector be lexicographically less than or equal to its image under σ.

If σ is a multi-row-column symmetry, then the sets R_σ and C_σ contain the indices of rows or columns, respectively, that are mapped to a higher row or column by σ:

Definition 3.10. *Let* $\mathbf{M}$ *be a matrix with multi-row-column symmetry* $\sigma = \sigma_r \circ \sigma_c$*, where* σ_r *and* σ_c *are a multi-row and a multi-column symmetry, respectively. Then,* $R_\sigma = \{i : \sigma_r(\mathbf{M}[i]) = \mathbf{M}[j], i < j\}$*, and* $C_\sigma = \{i : \sigma_c(\mathbf{M}[][i]) = \mathbf{M}[][j], i < j\}$.

For example, if $\sigma = \sigma_R^{1,4} \circ \sigma_R^{3,5} \circ \sigma_C^{4,6}$, that is, rows 1 and 3 column 4 are swapped with rows 4 and 5 and column 6, then $R_\sigma = \{1, 3\}$ and $C_\sigma = \{4\}$.

If $\mathbf{M}$ is a matrix and R, C are sets of integers, then $\mathbf{M} \upharpoonright \langle R, C\rangle$ denotes the restriction of the row-wise ordering of $\mathbf{M}$ to variables that have either a row index in R or column index in C:

Definition 3.11. *If* $\mathbf{M}$ *is an* $r \times c$ *matrix,* $R \subseteq \{1, \ldots, r\}$ *and* $C \subseteq \{1, \ldots, c\}$ *are sets of integers, and* $\vec{c}$ *is the natural total order over* C *of length* n*, then* $\mathbf{M} \upharpoonright \langle R, C\rangle$ *denotes the vector* $\vec{x_1}^\frown \cdots ^\frown \vec{x_r}$*, where for* $1 \leq i \leq r$*:*

$$\vec{x}_i = \begin{cases} \mathbf{M}[i] & \text{if } i \in R \\ \langle \mathbf{M}[i][\vec{c}[1]], \ldots, \mathbf{M}[i][\vec{c}[n]]\rangle & \text{if } i \notin R \end{cases}$$

For example, if $\mathbf{M} = \langle\langle x_1, x_2, x_3\rangle, \langle x_4, x_5, x_6\rangle, \langle x_7, x_8, x_9\rangle\rangle$, then $\mathbf{M} \restriction \langle\{1,2\},\{2\}\rangle$ contains first and second rows of $\mathbf{M}$, and the second element of the third row: $\langle x_1, x_2, x_3, x_4, x_5, x_6, x_8\rangle$.

Using the above definitions, Multi Lex is defined as follows:

Definition 3.12. *If* $\mathbf{M}$ *is a matrix with multi-row-column symmetry group* Γ*, and* Σ *is a generating set of* Γ *as in Definition 3.7, then the* **Multi Lex** *constraint for* $\mathbf{M}$ *and* Σ *is defined as follows:*

$$\textsc{MultiLex}(\mathbf{M}, \Sigma) \stackrel{\text{def}}{=} \bigwedge_{\sigma \in \Sigma} \mathbf{M} \restriction \langle R_\sigma, C_\sigma\rangle \preceq_L \sigma(\mathbf{M}) \restriction \langle R_\sigma, C_\sigma\rangle.$$

A constraint is posted for each element of the generating set. On the left-hand side, the variables in $\mathbf{M}$ have been placed in row-wise order, and then limited to those that are mapped to a higher row or column under σ (i.e., with a row or column number in R_σ or C_σ). On right-hand side they have been permuted by σ before being ordered and limited.

The following theorem establishes that Multi Lex holds for canonical matrices, meaning that it is a correct symmetry breaking constraint:[3]

Theorem 3.1. Multi Lex Correctness. *Let* $\mathbf{M}$ *be a matrix with multi-row-column symmetry group* Γ*,* Σ *be a generating set of* Γ *as in Definition 3.7 and* θ *be a ground substitution that is complete* w.r.t. *to the variables in* $\mathbf{M}$*. If* $\mathsf{can}(\mathbf{M}, \Gamma, \theta)$ *and* $\sigma \in \Sigma$*, then* $\theta(\mathbf{M}) \restriction \langle R_\sigma, C_\sigma\rangle) \preceq_L \sigma(\theta(\mathbf{M})) \restriction \langle R_\sigma, C_\sigma\rangle$ *holds.*

Multi Lex posts a polynomial-sized set of constraints, that in the worst case is of size $4rc$:

Theorem 3.2. Multi Lex Size Complexity. *Let* $\mathbf{M}$ *be an* $r \times c$ *matrix with multi-row-column symmetry group* Γ*, and* Σ *be the generating set of* Γ *as in Definition 3.7. Then, Multi Lex posts constraints of size* $\mathcal{O}(rc)$*.*

Comparison with Double Lex It is clear from the above that when applied to row or column symmetry groups, Multi Lex reduces to Double Lex. For example, if all rows are interchangeable, then from Definition 3.7, $\Sigma = \{\sigma_1, \ldots, \sigma_{r-1}\}$ where each σ_i swaps rows i and $i+1$. In this case, $\mathbf{M} \restriction \langle R_{\sigma_i}, C_{\sigma_i}\rangle$ resolves to $\mathbf{M}[i]$, meaning that Multi Lex, like Double Lex, requires the rows to lexicographically ordered, that is, for $1 \leq i \leq r-1$, $\mathbf{M}[i] \preceq_{\text{L}} \mathbf{M}[i+1]$.

However, unlike Double Lex, Multi Lex can be applied to a broader range of symmetries, including those found in arbitrary graph models (i.e., directed with cycles and self-loops allowed). This flexibility comes with no additional size cost, as both methods produce constraints smaller than $4rc$.

[3] Proofs for all theorems in this chapter are in Appendix B.

3.4 Experimental Evaluation

This evaluation addresses two questions. As discussed in Section 3.2.5, more general symmetry groups can express "weaker" symmetries. The first question is thus whether multi-row-column symmetries introduce a significant number of non-canonical solutions into the search space, as compared with stronger symmetries that swap the same rows and columns individually. Secondly, Multi Lex is a partial symmetry breaking constraint, raising the question of its effectiveness in practice.

Symmetry breaking techniques are typically assessed by measuring the degree to which they improve search times, however, this approach is not ideal. Constraint solvers rely on variable ordering and branching heuristics that can clash with symmetry breaking constraints [61, 123], meaning that any difference in search time cannot be attributed solely to symmetry reduction. Additionally, isolating the degree of symmetry introduced by multi-row-column symmetries is complicated by the fact that benchmark problems may contain a combination of generalised and simple symmetries, or no generalised symmetries at all. Thus, this section takes the same approach as Shlyakhter [119], and answers the above questions by counting the number of non-canonical solutions that are introduced by generalised symmetries, and removed by Multi Lex, over a range of synthetic test cases.

3.4.1 Experimental Setup

Tests were performed over unconstrained matrix models, simple graph models and DAG models with a range of sizes and symmetries. Each test case comprises a variable matrix $\mathbf{M}$, a set of constraints $\mathbf{C}$, and a symmetry group Γ with generator Σ. For all matrix models $\mathbf{C} = \varnothing$, and for graph models $\mathbf{C}$ contains only those constraints required to ensure that a solution represents a simple graph or DAG, as appropriate. For each model, a range of different symmetry groups were created by dividing the rows and columns into varying numbers of equal-sized divisions as in Definition 3.7. Four sets of tests were performed, as described below:

A: Matrices with multi-row-column symmetry Test cases comprise an unconstrained $r \times c$ matrix model $\mathbf{M}$ and a multi-row-column symmetry group Γ with generator Σ defined by partitioning the rows and columns into n interchangeable divisions. Tests cover all $2 \leq c \leq r \leq 6$ and all n *s.t.* $n|r$ and $n|c$.[4]

B: Matrices with multi-row and multi-column symmetry Test cases comprise an unconstrained $r \times c$ matrix model $\mathbf{M}$ and a multi-row and multi-column symmetry group $\Gamma = \Gamma_r \times \Gamma_c$ with generator $\Sigma = \Sigma_r \cup \Sigma_c$. Multi-row symmetry group Γ_r and multi-column symmetry group Γ_c were produced by partitioning the rows into n_r interchangeable lists, the columns into n_c interchangeable lists, respectively. Tests cover all $2 \leq c \leq r \leq 6$ and all n_r, n_c *s.t.* $n_r|r$ and $n_c|c$.

[4] $x|y$ indicates that x and y are positive integers and y is a multiple of x.

C: Simple graph models Test cases comprise a v-vertex simple graph model $\mathbf{M}$ and a multi-vertex symmetry group produced by dividing the vertices into n interchangeable lists. Constraint set $\mathbf{C}$ ensures that solutions represent simple graphs:

$$\bigwedge_{1\leq i\leq v} \neg\mathbf{M}[i][i]. \tag{3.1}$$

$$\bigwedge_{1\leq i<j\leq v} \mathbf{M}[i][j] \leftrightarrow \mathbf{M}[j][i]. \tag{3.2}$$

Tests cover all graph models with all $2 \leq v \leq 12$, and all n *s.t.* $n|v$.

D: DAG models Test cases comprise a v-vertex DAG model and a multi-vertex symmetry group defined by dividing the vertices into n interchangeable lists. The DAG model contains a v^2 adjacency matrix $\mathbf{M}$, and additional variables and constraints to enforce acyclicity. The v^2 matrix $\mathbf{T}$ contains the transitive closure of $\mathbf{M}$, with $\mathbf{T}[i][j] = 1$ indicating that v_j is reachable from v_i, and $\mathbf{A}$ is a v^3 matrix with $\mathbf{A}[i][j][k]$ indicating that v_k is reachable from v_i via v_j. Formulae 3.3 and 3.4 define the values of $\mathbf{T}$ and $\mathbf{A}$, and Formula 3.5 enforces acyclicity:

$$\bigwedge_{1\leq i,j,k\leq v} \mathbf{A}[i][j][k] \leftrightarrow \mathbf{M}[i][j] \wedge \mathbf{T}[j][k]. \tag{3.3}$$

$$\bigwedge_{1\leq i,j\leq v} \mathbf{T}[i][j] \leftrightarrow (\mathbf{M}[i][j] \vee \bigvee_{1\leq k\leq n} \mathbf{A}[i][k][j]). \tag{3.4}$$

$$\bigwedge_{1\leq i\leq v} \neg\mathbf{T}[i][i]. \tag{3.5}$$

Tests cover all DAG models with $2 \leq v \leq 8$, and all n *s.t.* $n|v$.

For each test case, the total number of solutions (sol, i.e., the total number of $r\times c$ matrices, or simple or directed acyclic graphs with v vertices), the number of canonical solutions (can) and the number of solutions allowed by Multi Lex (mlx) are reported. All values of sol are well-known [57], and some values of can [57, 58, 122]. Unknown values of can were computed by counting the solutions that satisfy $\mathbf{C}\wedge\bigwedge_{\sigma\in\Gamma}\mathbf{M}_R \preceq_{\mathrm{L}} \sigma(\mathbf{M}_R)$, and the value of mlx was computed by counting the solutions that satisfy $\mathbf{C}\wedge\textsc{MultiLex}(\mathbf{M},\Sigma)$. For this, the exact model counters Relsat [69] and GANAK [117] were used over appropriately constructed SAT instances.

The degree of symmetry created by a symmetry group is measured by the proportion of non-canonical solutions, that is, $\mathsf{sym} = (\mathsf{sol}-\mathsf{can})/\mathsf{sol}$, and the success of Multi Lex is measured by its efficiency in ruling out non-canonical solutions, that is $\mathsf{eff} = (\mathsf{sol}-\mathsf{mlx})/(\mathsf{sol}-\mathsf{can})$.

3.4.2 Results

Results are displayed in Tables 3.1–3.4. Canonical solution counts indicate that while generalised symmetries result in fewer redundant solutions than row, column or vertex symmetries, they still result in a factorial size increase in the search space. Efficiency results show that Multi Lex removes nearly

$r \times c$	n	Solns	Canonical	Sym. (%)	Allowed	Eff. (%)
2×2	2	2^4	10	37.50	10	100.00
3×3	3	2^9	104	79.68	139	91.42
4×2	2	2^8	136	46.87	136	100.00
4×4	2	2^{16}	32,896	49.80	32,896	100.00
4×4	4	2^{16}	3,044	95.35	6,192	94.96
5×5	5	2^{25}	291,968	99.12	897,275	98.18
6×2	2	2^{12}	2,080	49.21	2,080	100.00
6×3	3	2^{18}	44,224	83.12	66,988	89.55
6×4	2	2^{24}	8,390,656	49.98	8,390,656	100.00
6×6	2	2^{36}	34,359,869,440	49.99	34,359,869,440	100.00
6×6	3	2^{36}	11,453,771,776	83.33	18,042,740,416	88.49
6×6	6	2^{36}	96,928,992	99.85	432,351,138	99.51

Table 3.1: Test set A results: matrix models with multi-row-column symmetry. *Solns* and *Canonical* indicate the number of solutions and canonical solutions, resp. *Sym.*, *Allowed* and *Eff.* indicate the % of solutions that are not canonical, the number of solutions allowed by Multi Lex, and the % of non-canonical solutions removed by Multi Lex, resp.

all non-canonical solutions, and that as the model size increases, the number of solutions that remain in each symmetry class tends towards a constant factor of the input.

3.4.2.1 Symmetry Rates for Generalised Symmetries

As expected, generalised symmetries introduce fewer symmetrical solutions than their simple counterparts. For example, Table 3.2 shows that while a 4×4 matrix with total row and column symmetry ($n_r = n_c = 4$) has 317 canonical solutions (99.51% symmetry), splitting the row and columns into two interchangeable groups of two ($n_r = n_c = 2$) increases this to $16,576$ (74.7% symmetry). More extreme differences occur in larger models. For example, Table 3.3 shows that a 12-vertex simple graph with total vertex symmetry ($n = 12$) has 1.6×10^{11} canonical solutions (99.99% symmetry), but splitting the vertices into two interchangeable groups of six ($n = 2$) increases this to 3.6×10^{19} (49.99% symmetry). Nevertheless, generalised symmetries render a significant proportion of the search space redundant, with $\mathsf{sym} \geq 95\%$ for most non-trivial matrices.

Closer examination reveals that the proportion of non-canonical solutions increases factorially with the number of interchangeable divisions of rows and/or columns, or vertices. This should come as no surprise. Pólya [101] observed that the proportion of v-vertex graphs that admit only trivial automorphisms (i.e., $\sigma(G) = G$ *iff* $\sigma = \sigma_{id}$) approaches 1 as v increases. Thus, for any given Γ, the number of symmetric alternatives for a graph will tend towards $|\Gamma| - 1$, and the proportion of graphs that are canonical *w.r.t.* Γ tends towards $1/|\Gamma|$.

To determine how quickly this theoretical limit is reached in the context of simple graphs, DAGs and matrices (i.e., undirected bipartite graphs) with generalised symmetries, the value $\Delta_c = (\mathsf{can}/\mathsf{sol}) - (1/|\Gamma|)$ was computed for each test case, that is, the difference between the observed proportion of canonical solutions and the predicted asymptotic limit. As multi-vertex and multi-row-column symmetry groups are isomorphic to S_n and multi-row and

$r \times c$	n_r	n_c	Solns	Canonical	Sym. (%)	Allowed	Eff. (%)
2×2	2	2	2^4	7	56.25	7	100.00
3×2	3	2	2^6	13	79.68	14	98.03
3×3	3	3	2^9	36	92.96	45	98.10
4×2	2	2	2^8	76	70.31	82	96.66
4×2	4	2	2^8	22	91.40	25	98.71
4×3	2	3	2^{12}	430	89.50	529	97.29
4×3	4	3	2^{12}	87	97.87	130	98.92
4×4	2	2	2^{16}	16,576	74.70	20,280	92.43
4×4	4	2	2^{16}	1,996	96.95	2,861	98.63
4×4	4	4	2^{16}	317	99.51	650	99.48
5×2	5	2	2^{10}	34	96.67	41	99.29
5×3	5	3	2^{15}	190	99.42	336	99.55
5×4	5	2	2^{20}	7,882	99.24	12,068	99.59
5×4	5	4	2^{20}	1,053	99.89	2,942	99.81
5×5	5	5	2^{25}	5,624	99.98	24,520	99.94
6×2	2	2	2^{12}	1,072	73.82	1,204	95.63
6×2	3	2	2^{12}	430	89.50	517	97.62
6×2	6	2	2^{12}	50	98.77	63	99.67
6×3	2	3	2^{18}	23,052	91.20	29,638	97.24
6×3	3	3	2^{18}	8,240	96.85	12,716	98.23
6×3	6	3	2^{18}	386	99.85	785	99.84
6×4	2	2	2^{24}	4,197,376	74.98	5,186,272	92.13
6×4	2	4	2^{24}	384,112	97.71	548,460	98.99
6×4	3	2	2^{24}	1,415,896	91.56	1,930,604	96.64
6×4	3	4	2^{24}	131,505	99.21	251,834	99.27
6×4	6	2	2^{24}	27,412	99.83	43,947	99.90
6×4	6	4	2^{24}	3,250	99.98	11,819	99.94
6×5	2	5	2^{30}	5,215,764	99.51	8,065,688	99.73
6×5	3	5	2^{30}	1,757,384	99.83	3,980,567	99.79
6×5	6	5	2^{30}	28,576	99.99	183,010	99.98
6×6	2	2	2^{36}	17,180,065,792	74.99	22,114,697,984	90.42
6×6	3	2	2^{36}	5,730,905,440	91.66	8,211,426,224	96.06
6×6	3	3	2^{36}	1,911,933,696	97.21	3,329,563,864	97.87
6×6	6	2	2^{36}	59,953,552	99.91	100,449,400	99.94
6×6	6	3	2^{36}	20,075,154	99.97	53,817,407	99.95
6×6	6	6	2^{36}	251,610	99.99	2,625,117	99.99

Table 3.2: Test set B results: matrix models with multi-row and multi-column symmetry. *Solns* and *Canonical* indicate the number of solutions and canonical solutions, resp. *Sym.*, *Allowed* and *Eff.* indicate the % of solutions that are not canonical, the number of solutions allowed by Multi Lex, and the % of non-canonical solutions removed by Multi Lex, resp.

multi-column symmetry groups are isomorphic to $S_{n_r} \times S_{n_c}$ (Section 3.2.4), this limit is $1/n!$ and $1/(n_r!n_c!)$, respectively.

Results show that Δ_c approaches 0 as either the model size $|\mathbf{M}|$ increases or n decreases. Figure 3.8 depicts this trend by plotting $|\mathbf{M}|$ against Δ_c for all test cases with values of $n, n_r, n_c \leq 4$ for which multiple data points are available. (For all test cases not included in the graph, $\Delta_c \leq 0.016$.) The graph indicates the proportion of canonical solutions quickly approaches $1/|\Gamma|$. Indeed, for all matrix models with $r \times c \geq 15$, and graph models with $v \geq 6$, $\Delta_c \leq 0.01$.

3.4.2.2 Effectiveness of Multi Lex

The results in Tables 3.1–3.4 show that despite being a partial symmetry breaking constraint, Multi Lex consistently removes nearly all non-canonical solutions. Its efficiency drops below 95% in just nine of 94 test cases, and all symmetries are broken in 19.

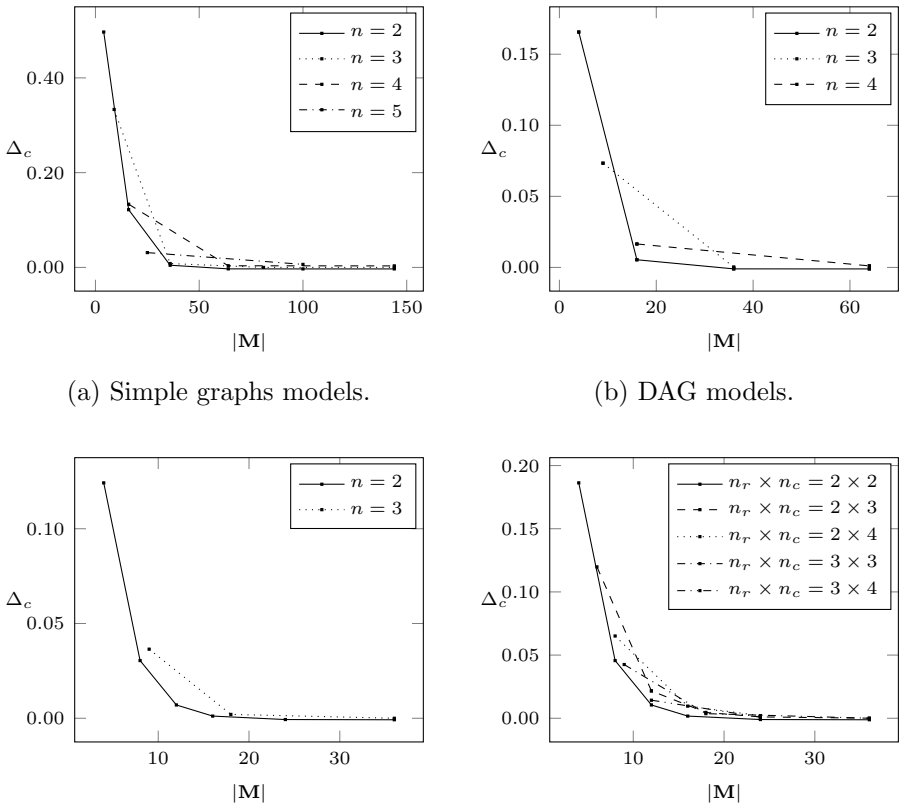
v	n	Solns	Canonical	Sym. (%)	Allowed	Eff. (%)
2	2	2^1	2	0.00	2	100.00
3	3	2^3	4	50.00	4	100.00
4	2	2^6	40	37.50	40	100.00
4	4	2^6	11	82.81	11	100.00
5	5	2^{10}	34	96.67	46	98.78
6	2	2^{15}	16,640	49.21	16,640	100.00
6	3	2^{15}	5,728	82.51	7,964	91.73
6	6	2^{15}	156	99.52	325	99.48
7	7	2^{21}	1,044	99.95	4,045	99.85
8	2	2^{28}	134,250,496	49.98	134,250,496	100.00
8	4	2^{28}	11,258,944	95.80	22,511,480	95.62
8	8	2^{28}	12,346	99.99	89,812	99.97
9	3	2^{36}	11,454,296,064	83.33	17,363,353,728	89.68
9	9	2^{36}	274,668	99.99	3,583,903	99.99
10	2	2^{45}	17,592,202,821,632	49.99	17,592,202,821,632	100.00
10	5	2^{45}	293,386,342,912	99.16	788,243,248,694	98.58
10	10	2^{45}	12,005,168	99.99	258,518,959	99.99
11	11	2^{55}	1,018,997,864	99.99	33,860,186,124	99.99
12	2	2^{66}	36,893,488,181,778,841,600	49.99	36,893,488,181,778,841,600	100.00
12	3	2^{66}	12,297,829,519,913,385,984	83.33	18,694,383,605,104,918,528	89.59
12	4	2^{66}	3,074,458,453,725,478,912	95.83	6,412,674,136,868,017,152	95.27
12	12	2^{66}	165,091,172,592	99.99	8,085,067,416,957	99.99

Table 3.3: Test set C results: simple graph models. *Solns* and *Canonical* indicate the number of solutions and canonical solutions, resp. *Sym.*, *Allowed* and *Eff.* indicate the % of solutions that are not canonical, the number of solutions allowed by Multi Lex, and the % of non-canonical solutions removed by Multi Lex, resp.

v	n	Solns	Canonical	Sym. (%)	Allowed	Eff. (%)
2	2	3	2	33.33	2	100.00
3	3	25	6	76.00	9	84.21
4	2	543	275	49.35	275	100.00
4	4	543	31	94.29	71	92.18
5	5	29,281	302	98.96	1,166	97.01
6	2	3,781,503	1,890,896	49.99	1,890,896	100.00
6	3	3,781,503	630,863	83.31	1,083,027	85.64
6	6	3,781,503	5,984	99.84	39,143	99.12
7	7	1,138,779,265	243,668	99.97	2,656,761	99.78
8	2	783,702,329,343	391,851,196,415	49.99	391,851,196,415	100.00
8	4	783,702,329,343	32,656,331,783	95.83	81,515,025,162	93.49
8	8	783,702,329,343	20,286,025	99.99	361,414,631	99.95

Table 3.4: Test set D results: DAG models. *Solns* and *Canonical* indicate the number of solutions and canonical solutions, resp. *Sym.*, *Allowed* and *Eff.* indicate the % of solutions that are not canonical, the number of solutions allowed by Multi Lex, and the % of non-canonical solutions removed by Multi Lex, resp.

Closer examination reveals that for multi-vertex and multi-row-column symmetry groups, the proportion of solutions allowed by Multi Lex, mlx/sol, approaches $1/2(n-1)!$ as the model size increases. (Any pattern in matrix models with multi-row and multi-column symmetry was not immediately clear). Unlike the trend shown in Figure 3.8, this value is not justified theoretically, but rather by an empirical analysis that shows it to be an accurate predictor of Multi Lex's efficiency.



(a) Simple graphs models.

(b) DAG models.

(c) Matrix models with multi-row-column symmetry.

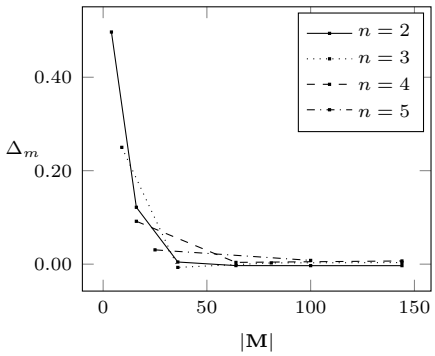
(d) Matrix models with multi-row and multi-column symmetry.

Figure 3.8: Asymptotic limit for canonical solutions under generalised symmetries. Model size $|\mathbf{M}|$ is plotted against $\Delta_c = (\mathsf{can}/\mathsf{sol}) - (1/|\Gamma|)$ for selected values of n, and n_r and n_c.

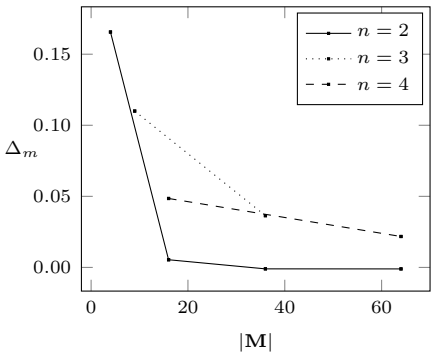
For all test cases over multi-vertex and multi-row-column symmetries, Figure 3.9d plots $\mathsf{mlx}/\mathsf{sol}$, the proportion of solutions left by Multi Lex, against $1/2(n-1)!$, the predicted asymptotic limit. A clear correlation is shown, with $\mathsf{mlx}/\mathsf{sol} \leq 1/2(n-1)!$ in all cases. Overall, predicting $\mathsf{mlx}/\mathsf{sol}$ with $1/2(n-1)!$ gives a root mean squared error of 0.09, and when restricted to matrices with multi-row-column symmetries, simple graphs or DAGs it is 0.04, 0.12 and 0.06, respectively.

Prediction errors decrease as the model size $|\mathbf{M}|$ increases and n decreases. Figure 3.9 shows this trend by plotting $|\mathbf{M}|$ against $\Delta_m = (\mathsf{mlx}/\mathsf{sol}) - (1/2(n-1)!)$ over values of $n \leq 4$ for which multiple data points are available. (For all test cases not included in the graph, $\Delta_m \leq 0.018$.) All matrix models with $r \times c \geq 12$ are within 0.02 of the asymptotic limit, all simple graphs and DAGs with $v \geq 6$ are within 0.01 and 0.04, respectively.

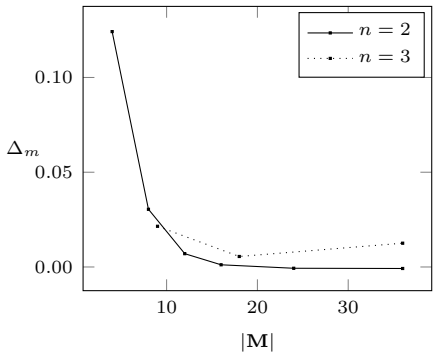
This observed trend indicates that Multi Lex performs better than could be expected. Katsirelos et al. [74] demonstrate that in the worst case, Double Lex, and therefore Multi Lex, leaves an exponential number of non-canonical solutions in each symmetry class. However, the more general asymptotic limits described above show that as the matrix size increases, this number tends towards a constant factor of n. Since the number of symmetry groups (i.e., canonical solutions) tends towards $\mathsf{sol}/n!$, and the



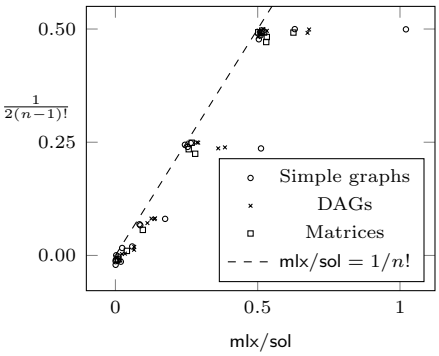
(a) Simple graph models.



(b) DAG models.



(c) Matrix models with multi-row-column symmetry.



(d) mlx/sol *vs* $1/2(n-1)!$ for all simple graph models, DAG models, and matrix models with multi-row-column symmetry.

Figure 3.9: The asymptotic limit of Multi Lex efficiency in simple graph models, DAG models, and matrix models with multi-row-column symmetry. Figures 3.9a–3.9c plot model size $|\mathbf{M}|$ against $\Delta_m = (\mathsf{mlx/sol}) - (1/2(n-1)!)$ for selected values of n, and Figure 3.9d plots mlx/sol against $1/2(n-1)!$ for all values of n.

number of solutions allowed by Multi Lex tends towards $\mathsf{sol}/2(n-1)!$, it follows that the number of non-canonical solutions per symmetry class that are not removed by Multi Lex tends towards $n!/2(n-1)! = n/2$.

3.5 Discussion

The main contributions of this chapter were as follows:

- This chapter formally defined a hierarchy of generalised symmetries in graph and matrix models. While standard approaches to symmetry breaking in matrix models break *row symmetries* and *column symmetries*, in which rows and columns are individually interchangeable, this chapter identified *multi-row symmetries* and *multi-column symmetries*, in which combinations of rows and columns are interchangeable but individual rows and columns are not, and *multi-row-column symmetries*, in which combinations of rows and columns are interchangeable.

- Similarly generalised symmetries in graph models, termed *multi-vertex symmetries*, were defined and shown to be special cases of multi-row-column symmetries.
- A series of examples show that these symmetries can occur in many plan optimisation problems and CSP benchmark domains [1].
- As these symmetries cannot be broken with standard techniques such as Double Lex [42] and Snake Lex [56], a novel symmetry breaking constraint was introduced, Multi Lex, that generalises Double Lex and can break generalised symmetries in matrix and graph models, including directed graphs with self-loops.
- An empirical evaluation was performed over a series of unconstrained matrix, simple graph and DAG models of different sizes and with different symmetries. Results show that *(i)*, despite being weaker than row and column symmetries, multi-row-column symmetries still result in a factorial size increase in the search space, and *(ii)*, Multi Lex is a compact and efficient symmetry breaking constraint that removes all but a constant factor of non-canonical solutions.

Sections 3.3 and 3.4 establish that Multi Lex is capable of efficiently breaking a range of generalised symmetries in matrix and graph models of all kinds. While this allows for an exploitation of symmetries not possible by previous means, and with no additional constraint size cost, a drawback is that the detection of multi-row-column symmetries can be a complex task.

While row and column symmetries are often obvious from the domain description, or can be found by a simple analysis of a problem instance, detecting multi-row-column symmetries can require specialised graph automorphism software such as NAUTY [85]. For example, in the *Social Golfers* problem (CSPLib problem 10), a weekly golf schedule must be generated where players play together no more than once. It is clear from the domain definition that the times, groups and golfers are all interchangeable, and so any problem instance will have total row and column symmetry. Symmetries in the *Progressive Party* problem (CSPLib problem 13), in which yacht parties must be scheduled such that crews meet no more than once and boat capacities are respected, are not immediate from the domain description. However, detecting them in a problem instance is a simple matter of finding all boats with identical capacities and crew sizes.

In contrast, the examples in Section 3.2 show that finding multi-row-column symmetries requires finding equivalent *combinations* of objects in the problem instance. The standard approach [32] of reducing this complex task to the graph automorphism problem (Sections 2.1.7.3 and 2.3.1) requires (possibly domain-specific) techniques for generating coloured graph representations of problem instances. This technique, while well established [68, 86, 100, 105], is not trivial.

A notable exception to this is the *Vessel Loading* problem (Figure 3.2), in which vertical and horizontal reflection symmetries are immediate from the domain description, and, interestingly, can be expressed as multi-row and multi-column symmetries, respectively. An obvious further consideration, as to whether the rotational symmetries exhibited by domains such as

N-Queens (CSPLib problem 54) can also be handled by Multi Lex, reveals the limitations of the generalised symmetry definitions in Section 3.2. Multi-row-column symmetries swap rows with rows and columns with columns, however, a rotational symmetry swaps row with columns. Thus, further work could explore Multi Lex as a novel approach to statically breaking reflection symmetries, or extend it to break rotational symmetries.

The results in Section 3.4, in which the exponential expansion of the search space created by multi-row-column symmetries was nearly entirely removed by Multi Lex, suggest that the additional work required to detect generalised symmetries is worthwhile. However, further work is required to determine whether these results can be replicated under the more complex constraint structures of standard benchmark problems.

Section 3.4.2.2 proposed an asymptotic limit on Multi Lex's efficiency. While this was justified empirically, an analytical justification would shed light on Multi Lex's, and therefore Double Lex's, behaviour, possibly leading to more efficient symmetry breaking techniques.

CHAPTER 4

Optimising Partial-Order Plans via Action Reinstantiation

4.1 Introduction

The least commitment approach to planning [135] aims to provide a scheduler or executing agent with additional flexibility at execution time. One way to achieve this is to post-process a ground, totally ordered plan into a minimally constrained partial-order plan (POP). This approach, known as *plan deordering* or *plan reordering* depending on the type of post-processing applied, has been extensively studied from both a technical [4, 8] and practical [71, 91, 116, 120] perspective (Section 2.2.2).

An aspect of least commitment which has received less attention is the optimisation of an agent's commitment to the domain objects that are used in the course of executing a plan. By their nature, deordering and reordering processes can only remove or modify the agent's commitment to the *timing* of actions: the resulting optimised plan will contain the same actions, and therefore make use of the same domain objects, as the original. This over-commitment to a particular set of objects can have a number of detrimental effects on the agent's execution-time options, including an over-commitment to a particular action ordering.

As discussed in Section 2.1.2, an action can be defined as a combination of an operator (a schematised action type) and a set of bindings for the operator's variables, that is, an action is an *instantiation* of an operator. A *reinstantiation* of an action is therefore a modification of its variable bindings, that is, a modification of the domain objects used by the agent in the course of executing the action. This chapter will introduce the notions of *reinstantiated deorderings* and *reinstantiated reorderings*: transformations of a plan under which ordering constraints can be removed or arbitrarily modified, respectively, *and the plan's actions' variable bindings can be changed*. The motivation behind these transformations is the possibility of finding a new set of variable bindings that allows for greater minimisation of the plan's ordering constraints–resulting in less commitment and more flexibility–than would be possible had the bindings remained unchanged.

Allowing a plan optimisation process to change variable bindings results in an exponentially larger search space. However, this space is often

1. $navigate_1(\mathsf{R}_1, \mathsf{W}_1, \mathsf{W}_2)$
2. $sample\text{-}soil(\mathsf{R}_1, \mathsf{W}_2)$
3. $navigate_2(\mathsf{R}_1, \mathsf{W}_2, \mathsf{W}_3)$
4. $sample\text{-}rock(\mathsf{R}_1, \mathsf{W}_3)$

(a) Plan P_1.

1. $navigate_1(\mathsf{R}_1, \mathsf{W}_1, \mathsf{W}_2)$
2. $sample\text{-}soil(\mathsf{R}_1, \mathsf{W}_2)$
3. $navigate_2(\mathsf{R}_2, \mathsf{W}_1, \mathsf{W}_3)$
4. $sample\text{-}rock(\mathsf{R}_2, \mathsf{W}_3)$

(b) Plan P_2.

$$O = \{\ navigate_1(x_1, x_2, x_3),\ sample\text{-}soil(x_4, x_5),\ navigate_2(x_6, x_7, x_8),\ sample\text{-}rock(x_9, x_{10})\ \}$$

$$\theta = \{\ x_1\backslash\mathsf{R}_1, x_2\backslash\mathsf{W}_1, x_3\backslash\mathsf{W}_2,\ x_4\backslash\mathsf{R}_1, x_5\backslash\mathsf{W}_2,\ x_6\backslash\mathsf{R}_2, x_7\backslash\mathsf{W}_1, x_8\backslash\mathsf{W}_3,\ x_9\backslash\mathsf{R}_2, x_{10}\backslash\mathsf{W}_3\ \}$$

$$\prec = \{\ navigate_1 \prec sample\text{-}soil, navigate_2 \prec sample\text{-}rock\ \}$$

(c) Partial-order plan P_3.

Figure 4.1: Three plans from the *rovers* domain. Subscripts have been used to better distinguish between actions of the same type.

highly *symmetrical*, meaning that a plan has many transformations that are identical, and equally optimal, up to an irrelevant permutation of, for example, operators of the same type, or interchangeable domain objects. As an optimisation algorithm need only visit one element from each set of symmetrical plans, search times can be greatly reduced through the addition of symmetry breaking constraints. Thus, this chapter will introduce a number of symmetry breaking techniques to aid in the reinstantiation task. In particular, the notion of a *causal structure symmetry* is introduced, a symmetry that results from interchangeable (potential) causal links between operator preconditions and postconditions.

4.1.1 Example

Consider a small planning task from (a reduced version of) the IPC *rovers* domain, in which a fleet of rovers must navigate the surface of a planet, collecting soil and rock samples. The domain objects comprise two rovers (R_1 and R_2) and three connected waypoints (W_1–W_3). Both rovers begin at W_1, there are soil and rock samples at W_2 and W_3, respectively, and paths connect all waypoints. The goal is to gather both samples.

Classical plan P_1 in Figure 4.1a[1] is an optimal solution to this problem: R_1 navigates to W_2 and collects the soil sample, then navigates to W_3 and collects the rock sample. An examination of the causal structure of P_1 shows that its ordering constraints cannot be modified without invalidating the plan. The preconditions of both *sample-soil* and $navigate_2$ require that R_1 be at W_2, a condition which is brought about only by $navigate_1$. Thus, for P_1 to remain valid, $navigate_1$ must precede both *sample-soil* and $navigate_2$, and for similar reasons $navigate_2$ must precede *sample-rock*. Additionally, $navigate_2$ moves R_1 away from W_2; it therefore *threatens* the causal link between $navigate_1$ and *sample-soil* and cannot be ordered between them. Thus, validity-preserving ordering constraints exist between all actions in P_1. As a result, standard de/reordering techniques cannot provide any

[1] Actions representing the initial state and goal have been omitted for space, and subscripts have been used to better distinguish between actions of the same type.

additional flexibility, as any relaxation or modification of P_1's ordering will render the plan invalid.

However, if P_1's variable bindings are allowed to change, then a more relaxed ordering can be found. Classical plan P_2 in Figure 4.1b is a modification of P_1 that, while still using the same operator *types*, instead uses rover R_2 to collect the rock sample. This modification allows its ordering constraints to be relaxed: the actions can be executed in any order so long as $navigate_1$ precedes *sample-soil*, and $navigate_2$ precedes *sample-rock*. The POP in Figure 4.1c is such a relaxation.

The rest of this chapter will continue as follows. Section 4.2 defines new optimality criteria for POPs which take all possible sets of variable bindings into consideration. Section 4.3 introduces a technique for computing optimally reinstantiated POPs by encoding the problem into a MaxSAT instance, and Section 4.4 introduces two symmetry-breaking techniques to improve this encoding. Section 4.5 experimentally assesses the benefits provided by reinstantiated reordering through a comparison with the de/reordering encodings of Muise et al. [91] and the non-optimal but effective EOG technique of Kambhampati and Kedar [71] (Section 2.2.2).

Results show that optimising both bindings and orderings incurs a much greater computational cost than optimising orderings alone. However, when (even non-optimal) reinstantiated de/reorderings are found, they are significantly more flexible than plans optimised with standard methods.

4.2 Optimality Criteria for Reinstantiated Partial-Order Plans

The standard optimality criteria for POPs[2] (Definition 2.18) are based on a set-wise comparison of ordering constraints, and assume that the variable bindings in the POPs remain static. This section will present generalised versions of these optimality criteria which lift this restriction. The aim of the new definitions remains the same, which is to compare the relative constrained-ness of two POPs. However, these extended definitions make use of additional information contained in the POP, the variable bindings, and so are capable of comparing POPs that would not be comparable under standard definitions.

First, the notions of deordering and reordering are extended to allow modifications to the POP's variable bindings. A *reinstantiated deordering* of a POP removes ordering constraints and allows the variable bindings to change, while a *reinstantiated reordering* allows arbitrary changes to both orderings and variable bindings.[3] In all cases, the resulting POP must remain both ground (i.e., all variables are bound, as in Definition 2.1) and valid (i.e., all linearisations are executable, as in Definition 2.3):[4]

[2] To more easily express changes in variable bindings, this chapter denotes a POP as $P = \langle O, \theta, \prec \rangle$, where O, θ and $\prec$ denote the operators, bindings and orderings as in Definition 2.1.

[3] There is no notion of "deinstantiation" as a POP $P = \langle O\theta, \prec \rangle$ is always ground, i.e., bindings θ are ground and complete *w.r.t.* the operators O.

[4] From Definition 2.1, these optimality criteria always compare the transitive closure of the POPs' ordering constraints.

Definition 4.1. *Let $P = \langle O, \theta, \prec \rangle$ and $Q = \langle O, \theta', \prec' \rangle$ be two POPs. Then:*

- *Q is a* **reinstantiated reordering** *of P* iff *both P and Q are valid,*
- *Q is a* **reinstantiated deordering** *of P* iff *Q is a reinstantiated reordering of P and $\prec' \subseteq \prec$, and*
- *Q is a* **reinstantiated strict deordering** *of P* iff *Q is a reinstantiated deordering of P and $\prec' \subset \prec$.*

For example, consider classical plans P_1 and P_2 and POP P_3 in Figure 4.1. As the three plans contain different actions (more specifically, the same operator types but different variable bindings), their flexibility cannot be compared under standard optimality definitions. However, the generalised definitions above do allow a comparison: as all plans are valid and use the same operators, they are reinstantiated reorderings of each other, and P_3 is a reinstantiated (strict) deordering of both P_1 and P_2.

The notion of a least-constrained POP can now be generalised to allow for changes in variable bindings. A *minimal reinstantiated deordering* of a POP is a reinstantiated deordering that cannot be relaxed any further, that is, there is no modification to the POP's variable bindings that will allow for the removal of any ordering constraints. A *minimum reinstantiated deordering* of a POP has the fewest ordering constraints of all of its reinstantiated deorderings. Out of all the ways that a POP's ordering and binding constraints can be modified, a *minimum reinstantiated reordering* contains the fewest ordering constraints:

Definition 4.2. *Let $P = \langle O, \theta, \prec \rangle$ and $Q = \langle O, \theta', \prec' \rangle$ be two POPs. Then:*

- *Q is a* **minimal reinstantiated deordering** *of P* iff *it is a reinstantiated deordering of P and there is no POP R such that R is a reinstantiated strict deordering of Q,*
- *Q is a* **minimum reinstantiated deordering** *of P* iff *it is a reinstantiated deordering of P and there is no POP $R = \langle O, \theta'', \prec'' \rangle$ such that R is a reinstantiated deordering of P and $| \prec'' | < | \prec' |$, and*
- *Q is a* **minimum reinstantiated reordering** *of P* iff *it is a reinstantiated reordering of P and there is no POP $R = \langle O, \theta'', \prec'' \rangle$ such that R is a reinstantiated reordering of P and $| \prec'' | < | \prec' |$.*

For example, plan P_3 has two ordering constraints, and as there is no reinstantiation of P_1 which allows for fewer than this, P_3 is a minimum reinstantiated reordering of P_1.

4.2.1 Computing Optimal Reinstantiations

Given that the size of a POP's ordering relation can be measured directly, and that reinstantiated de/reordering is a generalisation of standard de/reordering, it is no surprise that deciding whether a POP has a reinstantiated de/reordering with fewer than k ordering constraints is NP-complete:[5]

[5] Proofs for all theorems in this chapter are in Appendix C.

Theorem 4.1. Minimum Reinstantiated Reordering *Given a POP* $P = \langle O, \theta, \prec \rangle$ *and an integer* $k > 0$*, determining whether there exists a POP* $Q = \langle O, \theta', \prec' \rangle$ s.t. Q *is a reinstantiated reordering (or deordering) of* P *and* $| \prec' | < k$ *is* NP*-complete.*

Additionally, the optimisation problems of finding a minimum reinstantiated de/reordering are NP-hard and cannot be approximated within a constant factor:

Theorem 4.2. Approximate Minimum Reinstantiated Reordering *The problem of finding a minimum reinstantiated deordering (or reordering) of a POP is not in* APX *unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{poly\ log\ n})$.

Optimality Under POCL-validity The results and definitions above use the standard definition of POP validity, that requires that all of a POP's linearisations be executable (Definitions 2.2 and 2.3). Under the additional requirement that the input and optimised POPs be POCL-valid, that is, that each operator precondition be supported by a postcondition of some previous operator (Definition 2.7), minimum de/reordering remain NP-complete (a trivial corollary of Theorem 4.8 in [8]). However, due to the stronger requirements of POCL-validity there exist minimum POCL-valid de/reorderings that can be further optimised while remaining valid under Definition 2.3 [72]. These complexity and completeness results can be trivially extended to minimum reinstantiated POCL-valid de/reordering.

4.3 MaxSAT Encodings

The problem of optimising a POP's ordering can be naturally expressed as an instance of the partial weighted MaxSAT problem (Section 2.1.5). This approach was introduced by Muise et al. [91], whose MD and MR encodings transform a POP into MaxSAT instances with optimal solutions corresponding to minimum POCL-valid de/reorderings, resp.

This section introduces MRD and MRR: generalised encodings that *allow the POP to be reinstantiated*. Their optimal solutions therefore correspond to minimum POCL-valid reinstantiated de/reorderings, respectively. As allowing variable bindings to change exponentially expands the search space, MRD and MRR also optimise MD and MR by removing propositions and clauses that are not required to enforce plan validity. Later sections will introduce further optimisations that exploit symmetries between interchangeable operators and domain objects.

Due to the difficulty of propositionalising Definition 2.3, all four encodings instead preserve plan validity with the POCL requirements in Definition 2.7. While easier to encode, this raises the possibility that the resulting optimised POPs contain unnecessary ordering constraints.

4.3.1 The MD and MR Encodings

Muise et al. [91]'s encodings are reproduced here for ease of reference. MD and MR use two "types" of propositional variables. If $P = \langle O, \prec, \theta \rangle$ is the input POP, then for each pair of actions $a_1, a_2 \in O\theta$, a proposition $\mathrm{p}_{a_1 \prec a_2}$

is introduced to indicate that a_1 must precede a_2 in the resulting POP. For all a_c, a_p and q such that a_c consumes q and a_p produces q, a proposition p_{a_p,q,a_c} is introduced to encode the requirement that in the final POP, a_p be causally linked to a_c with respect to q.

MD and MR encode a POP into a partially-weighted MaxSAT instance with the clauses described below. The first three sets of hard clauses ensure that the output POP is "well-formed", that is, acyclic and transitively closed with all actions ordered between the initial state and goal:

$$\bigwedge_{a} \neg \mathrm{p}_{a \prec a} \,. \tag{4.1}$$

$$\bigwedge_{a_1,a_2,a_3} \mathrm{p}_{a_1 \prec a_2} \wedge \mathrm{p}_{a_2 \prec a_3} \rightarrow \mathrm{p}_{a_1 \prec a_3} \,. \tag{4.2}$$

$$\bigwedge_{a \notin \{a_I, a_G\}} \mathrm{p}_{a_I \prec a} \wedge \mathrm{p}_{a \prec a_G} \,. \tag{4.3}$$

The next two sets of hard clauses ensure that the solution represents a POCL-valid POP as in Definition 2.7:

$$\bigwedge_{\substack{a_p,a_c,q: \\ \mathsf{cons}(a_c,q), \\ \mathsf{prods}(a_p,q)}} \Big(\mathrm{p}_{a_p,q,a_c} \rightarrow \bigwedge_{\substack{a_t : a_t \neq a_c, \\ \mathsf{thrtns}(a_t,q)}} \mathrm{p}_{a_t \prec a_p} \vee \mathrm{p}_{a_c \prec a_t} \Big). \tag{4.4}$$

$$\bigwedge_{\substack{a_c,q: \\ \mathsf{cons}(a_c,q)}} \bigvee_{a_p : \mathsf{prods}(a_p,q)} \mathrm{p}_{a_p \prec a_c} \wedge \mathrm{p}_{a_p,q,a_c} \,. \tag{4.5}$$

Formula 4.6 adds a soft unit clause with the *negation* of each ordering proposition, meaning that higher weight (i.e., preferred) solutions will have fewer ordering constraints:

$$\bigwedge_{a_1,a_2}{}^{1} \neg \mathrm{p}_{a_1 \prec a_2}. \tag{4.6}$$

Definition 4.3. *The* MR *encoding transforms a POP into a partially weighted* MaxSAT *instance through Formulae 4.1–4.6.*

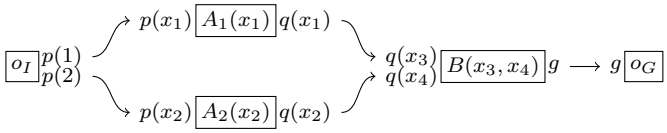
The MD encoding extends MR with clauses that disallow any ordering constraints that were not in the input plan, thus forcing the resulting POP to be deordering of the input:

$$\bigwedge_{(a_1,a_2) \notin \prec} \neg \mathrm{p}_{a_1 \prec a_2} \,. \tag{4.7}$$

Definition 4.4. MD *encodes a POP into a partial weighted* MaxSAT *instance through Formulae 4.1–4.7.*

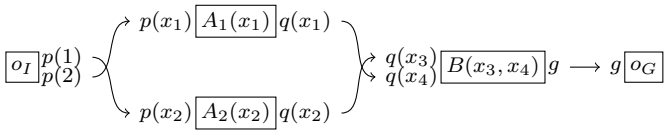
4.3.2 The MRD and MRR Encodings

MRD and MRR allow the input POP to be reinstantiated, and so use different variable "types" to express the allowable rebindings. Three types are used, as per the following sets:



	$p(x_1)$	$p(x_2)$	$q(x_3)$	$q(x_4)$	g
$p(1)$	1	0	0	0	0
$p(2)$	0	1	0	0	0
$q(x_1)$	0	0	1	0	0
$q(x_2)$	0	0	0	1	0
g	0	0	0	0	1

(a) Plan P_4.



	$p(x_1)$	$p(x_2)$	$q(x_3)$	$q(x_4)$	g
$p(1)$	0	1	0	0	0
$p(2)$	1	0	0	0	0
$q(x_1)$	0	0	0	1	0
$q(x_2)$	0	0	1	0	0
g	0	0	0	0	1

(b) Plan P_5.

Figure 4.2: Two plans with symmetrical causal structures from a synthetic domain. Operator pre/postconditions are to the left and right respectively, and arrows indicate supporting relationships.

- $\mathcal{P}_{\prec}$ contains propositions of the form $\mathrm{p}_{o_1 \prec o_2}$, which indicate that $o_1 \prec o_2$ in the resulting POP,
- $\mathcal{P}_{\theta}$ contains propositions of the form $\mathrm{p}_{u=s}$, which encode a requirement that in the final POP $\theta(u) = \theta(s)$, and
- $\mathcal{P}_{\rightarrow}$ contains propositions of the form $\mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}}$, indicating that there must be some q *s.t.* causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle$ is unthreatened in the final POP. (Abbreviated to $\mathrm{p}_{\vec{u}, \vec{s}}$ when clear from context).

Muise et al. [91] observed that the cubic number of clauses generated by Formula 4.2 make MD and MR infeasible for large plans: for plans with more than (approximately) 200 steps, the implemented encoder simply runs out of memory before generating a SAT formula. As this is exacerbated in MRD and MRR by the need to also close the variable equality relation, a number of optimisations are introduced that reduce the encoding size by removing redundant clauses and propositions.

The first derives from the observation that any ordering or binding constraints (and propositions that encode them) that are not required to preserve validity can be removed. For example, consider plan P_4 in Figure 4.2a (ignoring the matrix below for now). Operator A_1 cannot be used to support any of A_2's preconditions, and nor does it threaten any of its

postconditions (nor *vice-versa*). Because there is no possible causal connection between A_1 and A_2, no minimum de/reordering of P_4 will require that $A_1 \prec A_2$ or *vice-versa*, or that x_1 and x_2 be (non-) codesignated. An optimised encoding can therefore exclude the propositions $\mathrm{p}_{A_1 \prec A_2}$, $\mathrm{p}_{A_2 \prec A_1}$, $\mathrm{p}_{x_1=x_2}$ and $\mathrm{p}_{x_2=x_1}$, reducing the number of clauses required to enforce ordering and binding transitivity.

The second optimisation prevents the encoding of possible causal links if the required ordering constraints are known to be unsatisfiable. For example, Formula 4.7 in MD enforces deordering with negated unit clauses. Rather than include propositions which are *a priori* false, the second optimisation instead removes them and limits possible supporting links to those where the producer preceded the consumer in the input plan.

The definition below formalises and generalises these ideas. If o_1 and o_2 are operators and $\prec_A$ is a set of "allowable" orderings, then o_1 and o_2 are *causally related iff* $o_1 \prec_A o_2$ and there is a possible causal or threat relationship between them:

Definition 4.5. *o_1, o_2, $q(\vec{u})$ and $q(\vec{s})$ are* **causally related** w.r.t. $\prec_A$, iff $o_1 \prec_A o_2$ *and either:*

- $\mathsf{prods}(o_1, q(\vec{u}))$ *and* $\mathsf{cons}(o_2, q(\vec{s}))$,
- $\mathsf{thrtns}(o_1, q(\vec{u}))$ *and* $\mathsf{prods}(o_2, q(\vec{s}))$, *or*
- $\mathsf{cons}(o_1, q(\vec{u}))$ *and* $\mathsf{thrtns}(o_2, q(\vec{s}))$.

This idea of "allowable" orderings and causally related operators is used to minimise the number of propositions that appear in the MRD and MRR encodings. The three sets of propositional variables can now be defined:

Definition 4.6. *Let $P = \langle O, \theta, \prec \rangle$ be a POP and $\prec_A \subseteq O^2$ be a set of allowable orderings. Then $\mathcal{P}_\prec$, $\mathcal{P}_\theta$, and $\mathcal{P}_\rightarrow$ are the smallest sets* s.t.*:*

- $\mathrm{p}_{o_1 \prec o_2} \in \mathcal{P}_\prec$ iff *either* (i) *there exists a $q(\vec{u})$ and $q(\vec{s})$* s.t. *o_1, o_2, $q(\vec{u})$ and $q(\vec{s})$ are causally related* w.r.t. *$\prec_A$, or* (ii) *there exists an o_3* s.t. *$\mathrm{p}_{o_1 \prec o_3}, \mathrm{p}_{o_3 \prec o_2} \in \mathcal{P}_\prec$,*
- $\mathrm{p}_{u=s} \in \mathcal{P}_\theta$ iff *either* (i) *there exists a o_1, o_2, $q(\vec{u})$ and $q(\vec{s})$ that are causally related* w.r.t. *$\prec_A$ and some i* s.t. *$\vec{u}[i] = u$, $\vec{s}[i] = s$, or* (ii) *there exists a t* s.t. *$\mathrm{p}_{u=t}, \mathrm{p}_{t=s} \in \mathcal{P}_\theta$,*[6] *and*
- $\mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}} \in \mathcal{P}_\rightarrow$ iff *there exists a q* s.t. *$\mathsf{prods}(o_p, q(\vec{u}))$, $\mathsf{cons}(o_c, q(\vec{s}))$ and $o_p \prec_A o_c$.*

MRD and MRR encode a POP $P = \langle O, \theta, \prec \rangle$ and a set of allowable orderings $\prec_A \subseteq O^2$ into a partially-weighted MaxSAT instance through the formulae described below. Formulae 4.8–4.10 ensure that any solution represents a well-formed POP, that is, one that is acyclic and transitively closed with all operators ordered between the initial state and goal. They derive from Formulae 4.1–4.3 in MR, but remove propositions and constraints not required to preserve validity:

[6] Implemented code merges all $\mathrm{p}_{u=s}$ and $\mathrm{p}_{s=u}$ and updates Formulae 4.11 and 4.12 accordingly.

$$\bigwedge_{\mathrm{p}_{o_1 \prec o_2}, \mathrm{p}_{o_2 \prec o_1} \in \mathcal{P}_\prec} \neg \mathrm{p}_{o_1 \prec o_2} \vee \neg \mathrm{p}_{o_2 \prec o_1} \,. \tag{4.8}$$

$$\bigwedge_{\substack{\mathrm{p}_{o_1 \prec o_2}, \mathrm{p}_{o_1 \prec o_3}, \\ \mathrm{p}_{o_2 \prec o_3} \in \mathcal{P}_\prec}} \mathrm{p}_{o_1 \prec o_2} \wedge \mathrm{p}_{o_2 \prec o_3} \rightarrow \mathrm{p}_{o_1 \prec o_3} \,. \tag{4.9}$$

$$\bigwedge_{\mathrm{p}_{o_I \prec o}, \mathrm{p}_{o \prec o_G} \in \mathcal{P}_\prec} \mathrm{p}_{o_I \prec o} \wedge \mathrm{p}_{o \prec o_G} \,. \tag{4.10}$$

Formulae 4.11 and 4.12 ensure that the equality relation over variables and constants is symmetric and transitive, and Formula 4.13 ensures that each variable is bound to exactly one object. They are extensions to MD and MR that ensure the consistency of the POP's variable bindings:

$$\bigwedge_{\mathrm{p}_{u=s} \in \mathcal{P}_\theta} \mathrm{p}_{u=s} \leftrightarrow \mathrm{p}_{s=u} \,. \tag{4.11}$$

$$\bigwedge_{\substack{\mathrm{p}_{s=u}, \mathrm{p}_{u=v}, \\ \mathrm{p}_{s=v} \in \mathcal{P}_\theta}} \mathrm{p}_{s=u} \wedge \mathrm{p}_{u=v} \rightarrow \mathrm{p}_{s=v} \,. \tag{4.12}$$

$$\bigwedge_{x \in \mathsf{vars}(O)} \Big(\bigvee_{\substack{c \in \mathsf{consts}(O): \\ \mathrm{p}_{x=c} \in \mathcal{P}_\theta}} \mathrm{p}_{x=c} \quad \wedge \bigwedge_{\substack{c_1, c_2 \in \mathsf{consts}(O): \\ \mathrm{p}_{x=c_1}, \mathrm{p}_{x=c_2} \in \mathcal{P}_\theta, \\ c_1 \neq c_2}} \neg \mathrm{p}_{x=c_1} \vee \neg \mathrm{p}_{x=c_2} \Big). \tag{4.13}$$

Formulae 4.14 and 4.15 encode POCL-validity. Formula 4.14 requires that each consumer be causally linked to at least one producer, and Formula 4.15 defines the ordering, binding and threat protection constraints required for a causal link to hold. These derive from Formulae 4.4 and 4.5 in MD and MR, but have been optimised to remove redundant propositions, and generalised to allow the POP's variable bindings to change:

$$\bigwedge_{\substack{o_c, q(\vec{s}): \\ \mathsf{cons}(o_c, q(\vec{s}))}} \bigvee_{\mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}} \in \mathcal{P}_\rightarrow} \mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}} \,. \tag{4.14}$$

$$\begin{aligned} \bigwedge_{\mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}} \in \mathcal{P}_\rightarrow} \mathrm{p}_{o_p, \vec{u}, o_c, \vec{s}} \;\rightarrow\; \Big[\bigwedge_{1 \leq i \leq |\vec{u}|} \mathrm{p}_{\vec{u}[i] = \vec{s}[i]} \wedge \mathrm{p}_{o_p \prec o_c} \;\wedge \\ \bigwedge_{\substack{o_t, q(\vec{v}): o_t \neq o_c, \\ \mathsf{thrtns}(o_t, q(\vec{v}))}} \Big(\bigvee_{\substack{\mathrm{p} \in \mathcal{P}_\prec \cap \\ \{\mathrm{p}_{o_t \prec o_p}, \mathrm{p}_{o_c \prec o_t}\}}} \mathrm{p} \quad \vee \bigvee_{\substack{\mathrm{p} \in \mathcal{P}_\theta \cap \\ \{\mathrm{p}_{\vec{t}[i] = \vec{s}[i]} : 1 \leq i \leq |\vec{t}|\}}} \neg \mathrm{p} \Big) \Big]. \end{aligned} \tag{4.15}$$

Formula 4.16, an optimised version of Formula 4.6 in MD and MR, adds soft clauses that minimise the ordering constraints:

$$\bigwedge_{\mathrm{p}_{o_1 \prec o_2} \in \mathcal{P}_\prec} {}^{1}\neg \mathrm{p}_{o_1 \prec o_2}. \tag{4.16}$$

The MRD and MRR encodings can now be defined. The allowed orderings for MRD are those from the input plan, forcing the resulting POP to be a deordering of the input:

Definition 4.7. MRD *encodes a POP* $P = \langle O, \theta, \prec \rangle$ *into a partially weighted* MaxSAT *instance through Formulae 4.8–4.16, with the allowable orderings* $\prec_A = \prec$.

MRR is more general, and allows all ordering constraints:

Definition 4.8. MRR *encodes a POP* $P = \langle O, \theta, \prec \rangle$ *into a partially weighted* MaxSAT *instance through Formulae 4.8–4.16, with the allowable orderings* $\prec_A = O^2$.

Solutions to the MRD and MRR encodings are defined as below:

Definition 4.9. *If* $P = \langle O, \theta, \prec \rangle$ *be a POP. Then, function* $S : \mathcal{P}_{\prec} \cup \mathcal{P}_{\theta} \rightarrow \{\text{true}, \text{false}\}$ *is a solution to the* MRD/MRR MaxSAT *if it satisfies all hard clauses, and is an optimal solution if it also minimises the size of* $\{\text{p} : \text{p} \in \mathcal{P}_{\prec}, S(\text{p}) = \text{true}\}$. *For any such function* S, $\text{POP}(S) = \langle O, \theta', \prec' \rangle$ *denotes its transformation into a POP, where:*

- $o_1 \prec' o_2$ iff $S(\text{p}_{o_1 \prec o_2}) = \text{true}$, *and*
- *for all* $x \mathsf{vars}(O)$ *and* $c \in \mathsf{consts}(O)$, $\theta'(x) = c$ iff $S(\text{p}_{x=c}) = \text{true}$, *and*
- *for all* $x, y \in \mathsf{vars}(O)$, $\theta'(x) = \theta'(y)$ iff $S(\text{p}_{x=y}) = \text{true}$.

The following theorem establishes that an optimal solution to the MRD/MRR MaxSAT encoding corresponds to a minimum POCL-valid reinstantiated de/reorder, respectively:

Theorem 4.3. MRD/MRR Correctness *Let* $P = \langle O, \theta, \prec \rangle$ *be a POP and* S *be an optimal solution to the* MRD/MRR MaxSAT *encoding. Then,* $\text{POP}(S) = \langle O, \theta', \prec' \rangle$ *is a POCL-valid reinstantiated de/reorder of* P, *and there is no* $R = \langle O, \theta'', \prec'' \rangle$ *that is a POCL-valid reinstantiated de/reorder of* P *where* $\prec'' \subset \prec'$.

4.4 Symmetry Breaking

A search or optimisation problem displays *symmetry* when subsets of its solutions are identical up to an irrelevant permutation of variables and/or constants. As a sound and complete search algorithm need only explore one element per set, exploiting symmetries can exponentially reduce the search times (Section 2.3). Here, symmetries are defined *syntactically*, that is, as automorphisms (Section 2.1.7.3) of a problem description that, when applied to a solution, preserves its validity and optimality, and are broken *statically*, that is, by extending the problem description with constraints that rule out redundant areas of the search space.

Reinstantiated de/reordering differs from standard de/reordering by allowing variable bindings to change. This raises the possibility of finding more flexible relaxations of a POP, but also creates an exponentially larger search space. Thus, any technique for reducing search times is welcome.

This section introduces extensions to MR and MRR that exploit three types of symmetry in POPs: MRR_{O} breaks *operator symmetries*, which occur when a POP contains multiple operators of the same type, MR_{A} breaks *action symmetries*, which occur when those operators also have the same parameters, and MRR_{C} breaks *causal structure symmetries*, which are produced by equivalent combinations of (potential) causal links.

4.4.1 Breaking Operator and Action Symmetries

The MRR_O encoding derives from the observation that swapping the ordering and binding constraints of operators of the same type will not affect the POP's validity or optimality. For example, consider two simple symmetrical plans from a path-finding domain, constructed from two instances of the *go* operator: $\langle go_1(1,2), go_2(2,3)\rangle$ and $\langle go_2(1,2), go_1(2,3)\rangle$.[7] The plans are identical up to swapping the orderings and bindings of go_1 and go_2.

Despite permuting both orderings and bindings, operator symmetries can, in many cases, be broken with ordering constraints alone. For example, the symmetry above can be broken by simply requiring that $go_2 \not\prec go_1$. The MRR_O encoding generalises this approach by selecting a linearisation of operators $\langle o_1, \ldots, o_n\rangle$ and requiring that for all $1 \leq i < j \leq n$ *s.t.* $\mathsf{name}(o_i) = \mathsf{name}(o_j)$, $o_j \not\prec o_i$.

This constraint is *correct* (i.e., at least one element is left per symmetry class) as every POP in which $o_j \prec o_i$ has a symmetric alternative in which the two operators' orderings and bindings have been swapped. However, it is not *complete* as it cannot rule out symmetrical bindings of unordered operators. For example, plans P_4 and P_5 in Figure 4.2 are identical up to a permutation of $A(x_1)$ and $A(x_2)$, but requiring that $A(x_1) \not\prec A(x_2)$ will rule out neither.

As discussed in Section 4.3.2, rather than breaking symmetries by adding negated unit clauses (e.g., $\neg\, \text{p}_{go_2 \prec go_1}$) it is more efficient to remove those propositions from the encoding entirely, and update the "allowable orderings" $\prec_A$. Thus, under MRR_O, $\prec_A$ does not contain any orderings between operators of the same type that contradict an arbitrarily chosen linearisation. Unlike standard approaches which break symmetries with additional constraints, this approach results in a *smaller* encoding:

Definition 4.10. *If $P = \langle O, \theta, \prec\rangle$ is a POP and $\prec'$ is an arbitrary linearisation of P, then* MRR_O *encodes P into a partially-weighted* MaxSAT *instance through Formulae 4.8–4.16, with the allowable orderings $\prec_A = O^2 \setminus \{(o_2, o_1) : \mathsf{name}(o_1) = \mathsf{name}(o_2), o_1 \prec' o_2\}$.*

Say et al. [116]'s work on plan de/reordering (Section 2.2.2.4) addresses a similar form of symmetry that occurs when a plan contains multiple identical *actions*. Operator and action symmetries are subtly different: the former occur when plan steps have the same type, the latter when they also have the same bindings. Nevertheless, the MR_A encoding breaks action symmetries in a similar way. If $\prec'$ is an arbitrary linearisation of the POP, then Formula 4.17 defines hard clauses that break action symmetries:

$$\bigwedge_{\substack{a_1(\vec{t}), a_2(\vec{u}):\\ \mathsf{name}(a_1)=\mathsf{name}(a_2),\\ \vec{t}=\vec{u}, a_1(\vec{t}) \prec' a_2(\vec{u})}} \neg\, \text{p}_{a_2 \prec a_1}\,. \tag{4.17}$$

The MR_A encoding extends MR by breaking action symmetries:

Definition 4.11. MR_A *encodes a POP into a partial weighted* MaxSAT *instance through Formulae 4.1–4.6 and 4.17.*

[7] Subscripts are used to better distinguish between different instances of actions or operators of the same type.

4.4.2 Breaking Causal Structure Symmetries

A POP's *causal structure* is an implicit set of *unthreatened causal links* (Definition 2.6) that associate operator postconditions (i.e., producers) with causally dependent preconditions (i.e., consumers). A causal structure can be represented as a matrix (Section 2.1.1) of Boolean variables, $\mathbf{C}$, that associates producers and consumers with rows and columns, respectively, with $\mathbf{C}[i][j] = 1$ indicating that there is an unthreatened causal link between producer i and consumer j.

A *causal structure symmetry* occurs when a plan's operators contain combinations of pre/postconditions that are *functionally equivalent* in that their causal links can be swapped without affecting the plan's validity or optimality. Such symmetries can be expressed as permutations of rows and/or columns of variables in $\mathbf{C}$, and efficiently broken with lexicographic constraints over the same.

As discussed above, MRR_O breaks operator symmetries, which occur when a POP contains multiple operators of the same type. However, a common form of symmetry not removed by MRR_O is object symmetry, which occurs when a POP's variables' domains contain (combinations of) interchangeable values. The MRR_C encoding is motivated by the observation that *breaking causal structure symmetries simultaneously breaks both operator and object symmetries.*

For example, Figure 4.2 depicts two symmetrical plans from a synthetic domain, with causal structures represented by both arrows and matrices. The two instances of A create an operator symmetry, meaning that swapping the bindings and orderings of $A(x_1)$ and $A(x_2)$ will produce an equally valid and optimal plan. This can be also described as a causal structure symmetry: modifying the plans so that any producer supporting $p(x_1)$ instead supports $p(x_1)$, and any consumer supported by $q(x_1)$ is instead supported by $q(x_2)$, and *vice versa*, will not affect their validity or optimality. Plan P_5 is the result of applying the above symmetry to P_4.

The plans also display an object symmetry that is expressible as a causal structure symmetry: as objects 1 and 2 are interchangeable, modifying the plans so that all consumers supported by $p(1)$ are instead supported by $p(2)$, and *vice versa*, preserves validity and optimality.

A key element of the causal structure symmetries described above is that they *swap multiple producers and consumers simultaneously*, and are thus *multi-row-column symmetries* (Definition 3.4) over $\mathbf{C}$. As such symmetries *cannot* in general be broken with standard approaches such as Double Lex [42] or Snake Lex [56] (Section 2.3.2), the Multi Lex constraint introduced in Chapter 3 (Definition 3.12) must be used instead.

For the object symmetry, this resolves to a simple requirement that the rows in $\mathbf{C}$ associated with $p(1)$ and $p(2)$ are lexicographically ordered, which holds for P_5 but not P_4. As the operator symmetry permutes producers and consumers simultaneously, the resulting constraint is more complex, resolving to a lexicographic comparison of the shaded areas in Figure 4.2. As the shaded area in P_5 lexicographically precedes that in P_4, the constraint once again holds for P_5 but not P_4.

Causal structure symmetries can be detected using the standard technique introduced by Crawford et al. [32] (Section 2.3.1), in which a prob-

lem's symmetries are found by computing the automorphisms of a structurally equivalent coloured graph. This section defines symmetry *w.r.t.* POPs, introduces the *plan description graph*, and then defines MRR_{C}, an extension to MRR that breaks causal structure symmetries with Multi Lex.

4.4.2.1 Symmetrical Partial-Order Plans

A symmetry is an automorphism of a problem description that preserves the problem's solutions. As the definition of a minimum reinstantiated reordering (Definition 4.2) makes no reference to the variable bindings or ordering constraints of the input POP, in this case the problem description is simply the input POP's operators, O. A symmetry of a POP is therefore a permutation of its variables and constants that leaves O unchanged and preserves the validity and optimality of any reinstantiated reorderings:[8]

Definition 4.12. *Permutation σ is a* **symmetry** *of POP $P = \langle O, \theta, \prec \rangle$* iff*:*

- $\mathsf{domain}(\sigma) = \mathsf{vars}(O) \cup \mathsf{consts}(O)$,
- $\sigma(O) = O$, *and*
- *if $Q = \langle O, \theta', \prec' \rangle$ is a reinstantiated reorder of P, then Q is valid* iff *$\sigma(Q)$ is valid and $| \prec' | = |\sigma(\prec')|$.*

A POP's symmetries form a *symmetry group* (Section 2.1.7.2) that partitions the space of reinstantiated reorderings into sets of symmetrically equivalent solutions. A POP P with symmetry group Γ is symmetrical to a POP Q *iff* there is some $\sigma \in \Gamma$ *s.t.* $\sigma(P) = Q$.

4.4.2.2 Detecting POP Symmetries

A POP's symmetries are detected by finding the automorphisms of its *plan description graph* (PDG), an undirected coloured graph that encodes the relevant aspects of its structure. This approach derives from the commonly used *problem description graph* (Section 2.3.4.2), which is used to find symmetries in planning instances in the context of heuristic state-space planning. The PDG is an adaptation of this for POP optimisation problems:

Definition 4.13. *The* **plan description graph** *of a POP $P = \langle O, \theta, \prec \rangle$ is an undirected coloured graph $G_P = \langle V, E, C \rangle$ with vertices V, edges E and colour function C defined as follows:*

- *V is the smallest set* s.t.*:*
  - *if $t \in \mathsf{vars}(O) \cup \mathsf{consts}(O)$ then $\mathrm{v}_t \in V$,*
  - *if $o \in O$ then $\mathrm{v}_o^{pre}, \mathrm{v}_o^{post} \in V$, and*
  - *if $o \in O$, $q(\vec{t}) \in \mathsf{post}(o) \cup \mathsf{pre}(o)$ and $1 \leq n \leq |\vec{t}|$ then $\mathrm{v}^n_{q(\vec{t})} \in V$.*

[8] The notation $\sigma(\eta)$ is used to denote the result of simultaneously replacing every variable or constant t in η with $\sigma(t)$, while preserving the structure (or type) of η.

- *E is the smallest relation* s.t.*:*
  - *if* $o(\vec{x}) \in O$ *then* $(\mathrm{v}^{pre}_{o(\vec{x})}, \mathrm{v}_{\vec{x}[1]}), (\mathrm{v}^{post}_{o(\vec{x})}, \mathrm{v}_{\vec{x}[1]}) \in E$,
  - *if* $o(\vec{x}) \in O$ *and* $1 \leq i < |\vec{x}|$ *then* $(\mathrm{v}_{\vec{x}[i]}, \mathrm{v}_{\vec{x}[i+1]}) \in E$,
  - *if* $o \in O$ *and* $q(\vec{t}) \in \mathsf{pre}(o)$ *then* $(\mathrm{v}^{pre}_{o}, \mathrm{v}^{1}_{q(\vec{t})}) \in E$,
  - *if* $o \in O$ *and* $q(\vec{t}) \in \mathsf{post}(o)$, $(\mathrm{v}^{post}_{o}, \mathrm{v}^{1}_{q(\vec{t})}) \in E$, *and*
  - *if* $o \in O$, $q(\vec{t}) \in \mathsf{post}(o) \cup \mathsf{pre}(o)$ *and* $1 \leq i < |\vec{t}|$ *then* $(\mathrm{v}^{i}_{q(\vec{t})}, \mathrm{v}^{i+1}_{q(\vec{t})}) \in E$.
- *For all* $\mathrm{v}_1, \mathrm{v}_2 \in V$, $C(\mathrm{v}_1) = C(\mathrm{v}_2)$ iff*:*
  - *there is a* $t_1, t_2 \in \mathsf{vars}(O) \cup \mathsf{consts}(O)$ s.t. $\mathsf{type}(t_1) = \mathsf{type}(t_2)$, $\mathrm{v}_1 = \mathrm{v}_{t_1}$ *and* $\mathrm{v}_2 = \mathrm{v}_{t_2}$,
  - *there is an* $o_1, o_2 \in O$ s.t. $\mathsf{name}(o_1) = \mathsf{name}(o_2)$ *and either* $\mathrm{v}_1 = \mathrm{v}^{pre}_{o_1}$ *and* $\mathrm{v}_2 = \mathrm{v}^{pre}_{o_2}$, *or* $\mathrm{v}_1 = \mathrm{v}^{post}_{o_1}$ *and* $\mathrm{v}_2 = \mathrm{v}^{post}_{o_2}$, *or*
  - *there is a* $q(\vec{t}_1), q(\vec{t}_2), i, j$ s.t. $\mathrm{v}_1 = \mathrm{v}^{i}_{q(\vec{t}_1)}$ *and* $\mathrm{v}_2 = \mathrm{v}^{j}_{q(\vec{t}_2)}$.

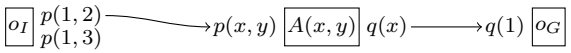
A vertex v_t is introduced for each variable and constant, a pair of vertices v^{pre}_{o} and v^{post}_{o} are introduced for each operator, representing its sets of preconditions and postconditions, respectively, and for every pre/postcondition of every operator the vertices $\mathrm{v}^{1}_{q(\vec{t})}, \ldots, \mathrm{v}^{n}_{q(\vec{t})}$ are introduced, where n is the arity of q. A different colour is introduced for every type of variable, constant and predicate, and for each operator type two colours are introduced, for its pre/postconditions, respectively. All constant, variable and operator vertices are coloured by type, and all pre/postcondition vertices are coloured according to the type and polarity of the predicate symbol.

Each operator's parameters are represented by a chain of vertices, with the first such parameter connected to the vertices representing the operator pre/postconditions. For example, if $o(\vec{x}) \in O$ and $\vec{x} = \langle x_1, \ldots, x_n \rangle$, then $(\mathrm{v}^{pre}_{o(\vec{x})}, \mathrm{v}_{x_1})$, $(\mathrm{v}^{post}_{o(\vec{x})}, \mathrm{v}_{x_1})$, $(\mathrm{v}_{x_1}, \mathrm{v}_{x_2}), \ldots, (\mathrm{v}_{x_{n-1}}, \mathrm{v}_{x_n}) \in E$. The vertices representing the pre/postconditions are also connected in a chain, with the first such vertex connected to that representing either the operator pre/postconditions as appropriate. For example, if $q(\vec{t}) \in \mathsf{post}(o(\vec{x}))$ and $\vec{t} = \langle t_1, \ldots, t_n \rangle$, then $(\mathrm{v}^{post}_{o(\vec{x})}, \mathrm{v}^{1}_{q(\vec{t})}), (\mathrm{v}^{1}_{q(\vec{t})}, \mathrm{v}^{2}_{q(\vec{t})}), \ldots, (\mathrm{v}^{n-1}_{q(\vec{t})}, \mathrm{v}^{n}_{q(\vec{t})}) \in E$. Additionally, to represent the specific terms found in each pre/postcondition, each vertex is connected to the term found in that position, that is, $(\mathrm{v}^{1}_{q(\vec{t})}, \mathrm{v}_{t_1})$, $\ldots$, $(\mathrm{v}^{n}_{q(\vec{t})}, \mathrm{v}_{t_n}) \in E$.

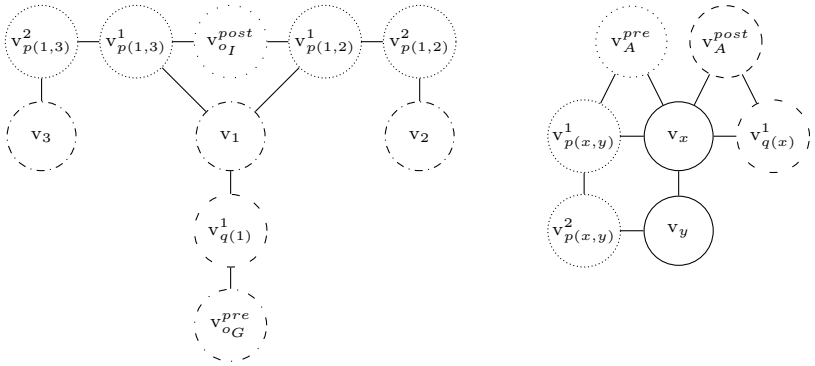
For example, Figure 4.3 depicts a POP and its PDG. The component on the left of the PDG represents the initial state, goal and the constants (1,2 and 3), and that on the right represents the operator $A(x, y)$. Vertex borders indicate colours.

The automorphisms of a POP's PDG correspond to symmetries of the POP. If $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ represents its translation into a permutation over the terms (i.e., variables and constants) appearing in P:

Definition 4.14. *If* $P = \langle O, \theta, \prec \rangle$ *is a POP and* $\sigma \in \mathsf{Aut}(G_P)$, *then* $\hat{\sigma}$ *is the permutation* s.t. $\hat{\sigma}(t_1) = t_2$ iff $\sigma(\mathrm{v}_{t_1}) = \mathrm{v}_{t_2}$ *and* $t_1, t_2 \in \mathsf{vars}(O) \cup \mathsf{consts}(O)$.



(a) Plan P_6.



(b) The plan description graph of P_6.

Figure 4.3: Plan description graph example.

For example, in Figure 4.3, the objects 2 and 3 are interchangeable. From Figure 4.3b, P_6's PDG has a single automorphism, σ, that swaps v_2, $\mathrm{v}^1_{p(1,2)}$ and $\mathrm{v}^2_{p(1,2)}$ with v_3, $\mathrm{v}^1_{p(1,3)}$ and $\mathrm{v}^2_{p(1,3)}$, respectively. This translates into the POP symmetry $\hat{\sigma} = (2\ 3)$, as expected.

The theorem below demonstrates the correctness of this approach:

Theorem 4.4. PLAN DESCRIPTION GRAPH *If P is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}$ is a symmetry of P.*

Of particular interest here are symmetries permute the POP's causal structure. A *consumer symmetry* or *producer symmetry* swaps the terms appearing in a pair of operator pre/postconditions, respectively, and a *causal structure symmetry* is a generalisation that swaps multiple producers and consumers simultaneously:

Definition 4.15. *Let $P = \langle O, \theta, \prec \rangle$ be a POP with symmetry $\hat{\sigma}$, and $\langle \vec{t}_1, \ldots, \vec{t}_n \rangle$ and $\langle \vec{u}_1, \ldots, \vec{u}_m \rangle$ be the pre/postcondition parameters for all operators in O. Then:*

- *$\hat{\sigma}$ is* **producer symmetry** iff *there exists an i, j* s.t. *$\hat{\sigma}$ swaps $\vec{u}_i$ and $\vec{u}_j$ and fixes all other terms,*
- *$\hat{\sigma}$ is* **consumer symmetry** iff *there exists an i, j* s.t. *$\hat{\sigma}$ swaps $\vec{t}_i$ and $\vec{t}_j$ and fixes all other terms, and*
- *$\hat{\sigma}$ is* **causal structure symmetry** iff *it is expressible as $\hat{\sigma} = \hat{\sigma}_1 \circ \cdots \circ \hat{\sigma}_k$ where each $\hat{\sigma}_i$ is either a producer or consumer symmetry.*

For example, in Figure 4.3, $\langle 1, 2 \rangle$ and $\langle 1, 3 \rangle$ are the parameters for o_I's postconditions. As $\hat{\sigma} = (2\ 3)$ is a symmetry of P_6, and $\hat{\sigma}(\langle 1, 2 \rangle) = \langle 1, 3 \rangle$ and *vice versa*, $\hat{\sigma}$ is a causal structure symmetry.

4.4.2.3 Breaking Causal Structure Symmetries with Multi Lex

Causal structure symmetries can be broken by applying Multi Lex (as described in Section 3.3 of Chapter 3) to a matrix representation of a causal structure. In the MRR and MRD encodings above, the optimised POP's causal structure is determined by the values of the variables in $\mathcal{P}_{\rightarrow}$ (Definition 4.6), with each $\mathrm{p}_{\vec{t},\vec{u}} \in \mathcal{P}_{\rightarrow}$ indicating that there is some o_p, o_c and q *s.t.* causal link $\langle o_p, q(\vec{t}), o_c, q(\vec{u})\rangle$ is unthreatened in the output optimised POP. These variables can be arranged into a matrix, $\mathbf{C}$, that associates producers and consumers with rows and columns, respectively, with $\mathbf{C}[i][j] = 1$ *iff* producer i supports consumer j:

Definition 4.16. *Let $P = \langle O, \theta, \prec\rangle$ be a POP, and $\langle \vec{t}_1, \ldots, \vec{t}_n\rangle$ and $\langle \vec{u}_1, \ldots, \vec{u}_m\rangle$ be the pre/postcondition parameters for all operators in O. Then, $\mathbf{C}$ is an $n \times m$ matrix of variables from $\mathcal{P}_{\rightarrow}$* s.t. *$\mathbf{C}[i][j]$ contains variable $\mathrm{p}_{\vec{t}_i,\vec{u}_j}$.*

Causal structure symmetries can be expressed as permutations of $\mathbf{C}$'s rows and/or columns. If $\hat{\sigma}$ is a causal structure symmetry, then $\bar{\sigma}$ denotes its transformation into a multi-row-column symmetry over $\mathbf{C}$:[9]

Definition 4.17. *Let $P = \langle O, \theta, \prec\rangle$ be a POP with symmetry $\hat{\sigma}$, and $\langle \vec{t}_1, \ldots, \vec{t}_n\rangle$ and $\langle \vec{u}_1, \ldots, \vec{u}_m\rangle$ be the pre/postcondition parameters for all operators in O* s.t. *$\mathbf{C}[i][j] = \mathrm{p}_{\vec{u}_i,\vec{t}_j}$. Then:*

- *if $\hat{\sigma}$ is a producer symmetry that swaps $\vec{u}_i$ and $\vec{u}_j$, then $\bar{\sigma} = \sigma_R^{i,j}$,*
- *if $\hat{\sigma}$ is a consumer symmetry that swaps $\vec{t}_i$ and $\vec{t}_j$, then $\bar{\sigma} = \sigma_C^{i,j}$, and*
- *if $\hat{\sigma} = \hat{\sigma}_1 \circ \cdots \circ \hat{\sigma}_k$ is a causal structure symmetry, then $\bar{\sigma} = \bar{\sigma}_1 \circ \cdots \circ \bar{\sigma}_k$.*

A set of causal structure symmetries, that is, multi-row-column symmetries over $\mathbf{C}$, $\bar{\Sigma}$, can be broken by posting the lexicographic constraint $\textsc{MultiLex}(\mathbf{C}, \bar{\Sigma})$.

4.4.2.4 The $\mathrm{MRR_C}$ Encoding

The $\mathrm{MRR_C}$ encoding is an extension of MRR that partially breaks causal structure symmetries with a set of hard MaxSAT clauses that encode the Multi Lex constraint. There are several steps to the construction of these clauses. First, the POP's PDG G_P is constructed, and Σ, the generating set of $\mathsf{Aut}(G_P)$, is computed. Then, Σ is transformed into $\hat{\Sigma}$, a set of symmetries over P *s.t.* $\hat{\sigma} \in \hat{\Sigma}$ *iff* $\sigma \in \Sigma$ (Definition 4.14). Finally, $\bar{\Sigma}$ is constructed, a set of multi-row-column symmetries over $\mathbf{C}$ *s.t.* $\bar{\sigma} \in \bar{\Sigma}$ *iff* $\hat{\sigma} \in \hat{\Sigma}$ and $\hat{\sigma}$ is a causal structure symmetry (Definitions 4.15 and 4.17). Formulae 4.18–4.21 encode a set of hard MaxSAT clauses propositionalise the Multi Lex constraint $\textsc{MultiLex}(\mathbf{C}, \bar{\Sigma})$:

[9] From Definition 3.1, $\sigma_R^{i,j}$ and $\sigma_C^{i,j}$ denote permutations that exchange the variables in rows i and j, or columns i and j, respectively, and fix all others.

$$\bigwedge_{\substack{\bar{\sigma},\vec{\mathrm{p}}:\,\bar{\sigma}\in\bar{\Sigma},\\ \vec{\mathrm{p}}=\mathbf{C}\restriction\langle R_{\bar{\sigma}},C_{\bar{\sigma}}\rangle}} \mathrm{a}^1_{\bar{\sigma}} \leftrightarrow (\vec{\mathrm{p}}[1] \leftrightarrow \bar{\sigma}(\vec{\mathrm{p}}[1])). \tag{4.18}$$

$$\bigwedge_{\substack{\bar{\sigma},\vec{\mathrm{p}},i:\,\bar{\sigma}\in\bar{\Sigma},\\ \vec{\mathrm{p}}=\mathbf{C}\restriction\langle R_{\bar{\sigma}},C_{\bar{\sigma}}\rangle,\\ 2\leq i<|\vec{\mathrm{p}}|}} \mathrm{a}^i_{\bar{\sigma}} \leftrightarrow (\mathrm{a}^{i-1}_{\bar{\sigma}} \wedge (\vec{\mathrm{p}}[i] \leftrightarrow \bar{\sigma}(\vec{\mathrm{p}}[i]))). \tag{4.19}$$

$$\bigwedge_{\substack{\bar{\sigma},\vec{\mathrm{p}}:\,\bar{\sigma}\in\bar{\Sigma},\\ \vec{\mathrm{p}}=\mathbf{C}\restriction\langle R_{\bar{\sigma}},C_{\bar{\sigma}}\rangle}} \vec{\mathrm{p}}[1] \rightarrow \bar{\sigma}(\vec{\mathrm{p}}[1]). \tag{4.20}$$

$$\bigwedge_{\substack{\bar{\sigma},\vec{\mathrm{p}},i:\,\bar{\sigma}\in\bar{\Sigma},\\ \vec{\mathrm{p}}=\mathbf{C}\restriction\langle R_{\bar{\sigma}},C_{\bar{\sigma}}\rangle,\\ 2\leq i\leq|\vec{\mathrm{p}}|}} \mathrm{a}^{i-1}_{\bar{\sigma}} \rightarrow \vec{\mathrm{p}}[i] \rightarrow \bar{\sigma}(\vec{\mathrm{p}}[i]). \tag{4.21}$$

For each formula above, $\bar{\sigma} \in \bar{\Sigma}$ is a multi-row-column symmetry over $\mathbf{C}$, and $\vec{\mathrm{p}} = \mathbf{C} \restriction \langle R_{\bar{\sigma}}, C_{\bar{\sigma}} \rangle$ is the list of Boolean variables found on the left-hand side of the Multi Lex constraint as in Definition 3.12. Formulae 4.18–4.21 encode a requirement that $\vec{\mathrm{p}} \preceq_{\mathrm{L}} \bar{\sigma}(\vec{\mathrm{p}})$. The AND-CSE propositionalisation technique of Elgabou [37] is used, which reduces the sizes of the formulae by replacing common subexpressions with auxiliary variables.

For each symmetry $\bar{\sigma}$, the auxiliary variables $\mathrm{a}^1_{\bar{\sigma}}, \ldots, \mathrm{a}^{n-1}_{\bar{\sigma}}$ are introduced, where $n = |\vec{\mathrm{p}}|$. Formulae 4.18 and 4.19 state that auxiliary variable $\mathrm{a}^i_{\bar{\sigma}}$ is *true iff* the first i elements of $\vec{\mathrm{p}}$ and $\bar{\sigma}(\vec{\mathrm{p}})$ are equal, that is, all *true* or all *false*. Formulae 4.20 and 4.21 require that for all $\bar{\sigma}$, if $i = 1$, or if all previous auxiliary propositions hold (i.e., all $\mathrm{a}^j_{\hat{\sigma}}$ such that $j < i$), then $\vec{\mathrm{p}}[i] \prec_{\mathrm{L}} \sigma(\vec{\mathrm{p}}[i])$, that is, $\vec{\mathrm{p}}[i] \rightarrow \sigma(\vec{\mathrm{p}}[i])$

The $\mathrm{MRR_C}$ encoding is an extension of MRR that breaks causal structure symmetries with the above technique:

Definition 4.18. *The* $\mathrm{MRR_C}$ *encoding transforms a POP into a partially-weighted* MaxSAT *instance through Formulae 4.8–4.21 with the allowable orderings* $\prec_A = O^2$.

4.5 Experimental Evaluation

While a minimum reinstantiated de/reorder of a POP can never be less flexible than a minimum de/reorder (Definitions 2.18 and 4.2), the question remains whether, in practice, reinstantiated de/reordering provides any significant increase in flexibility over standard de/reordering. The search for a minimum de/reorder often times out before an optimal (or indeed any) solution has been found [91], and allowing variable bindings creates a harder problem with an exponentially larger search space. This section will thus address the question of whether reinstantiated de/reordering can provide more flexibility than standard de/reordering under the same resource constraints, and will assess the ability of the symmetry breaking techniques introduced in Section 4.4 to ameliorate this increase in search space.

Results show that reinstantiation provides significantly more flexibility than standard approaches – a 21.1% *flex* increase on average, and up to 89.3% depending on domain. However, this comes at a computational

cost, with coverage dropping to 55.6%. And while symmetry breaking produces little additional flexibility, it allows the optimality of solutions to be exhaustively proved.

4.5.1 Experimental Setup

The questions above are answered by comparing optimised plans produced by the Loandra MaxSAT solver [12] using the MD, MR, MR_A, MRD, MRR, MRR_O and MRR_C encodings.[10] Additionally, to evaluate the optimisations described in Definitions 4.5 and 4.6, unoptimised variants of MR and MRR, labelled MR_N and MRR_N, are also briefly examined.

As a baseline, the MaxSAT-based techniques will be compared with the *explanation-based order generalisation* (EOG) deordering technique of Kambhampati and Kedar [71] (Section 2.2.2.2). The EOG algorithm relies on a *validation structure*, that is, a set of causal links that "explain" the POP's validity:

Definition 4.19. *Let $P = \langle O, \theta, \prec \rangle$ be a POP and $\prec'$ be an arbitrary linearisation of $\prec$. Then, a* **validation structure** *V_P is a set of causal links* s.t. *$\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle \in V_P$* iff*:*

1. *$o_p \prec o_c$,*
2. *$\theta(\vec{u}) = \theta(\vec{s})$, and*
3. *for all o_t, $\vec{v}$* s.t. *$\mathsf{thrtns}(o_t, q(\vec{v}))$, either $o_t \prec o_p$, $o_c \prec o_t$ or $\theta(\vec{s}) \neq \theta(\vec{v})$, and*
4. *for all $\langle o'_p, q(\vec{u}'), o_c, q(\vec{s}) \rangle$* s.t. *1–3 above hold and $o_p \neq o'_p$, $o_p \prec o'_p$.*

Orderings not required to maintain validity under V_P are removed:

Definition 4.20. *If $P = \langle O, \theta, \prec \rangle$ is a POP with validation structure V_P, then the* **explanation-based order generalisation** *of P is the POP $\text{EOG}(P) = \langle O, \theta, \prec'^{+} \rangle$ where $o_1 \prec' o_2$* iff *there exists a $q(\vec{u})$, $q(\vec{s})$* s.t. *either:*

- *$\langle o_1, q(\vec{u}), o_2, q(\vec{s}) \rangle \in V_P$,*
- *there exists an o_3 and $q(\vec{v})$* s.t. *$\langle o_2, q(\vec{s}), o_3, q(\vec{v}) \rangle \in V_P$, $o_1 \prec o_2$, $\mathsf{thrtns}(o_1, q(\vec{u}))$, and $\theta(\vec{v}) = \theta(\vec{u})$, or*
- *there exists an o_3 and $q(\vec{v})$* s.t. *$\langle o_3, q(\vec{v}), o_1, q(\vec{u}) \rangle \in V_P$, $o_1 \prec o_2$, $\mathsf{thrtns}(o_2, q(\vec{s}))$ and $\theta(\vec{v}) = \theta(\vec{s})$.*

While EOG cannot guarantee even a minimal deorder of its input [8], it can be computed in polynomial time and in practice consistently finds minimum deorderings [91]. It thus serves an effective baseline for evaluating the MaxSAT-based optimisation techniques.

Figure 4.4 depicts the hierarchy of (the problems solved by) these algorithms. EOG is in P, all others are NP-hard to compute optimally. MD (*minimum deorder*) searches for a minimally ordered POP but disallows reorderings and changes to variable bindings. This is relaxed by both MR

[10] https://bitbucket.org/max_waters/phd-plan-flexibility-tools

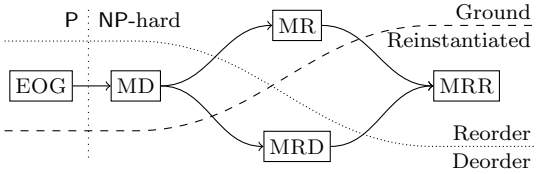


Figure 4.4: Partial-order plan optimisation hierarchy. Arrows lead from less to more general optimisations.

(*minimum reorder*), which allows reorderings, and MRD (*minimum reinstantiated deorder*), which allows bindings to change. The most general is MRR (*minimum reinstantiated reorder*), which allows both reorderings and changes to bindings. MR_A, MRR_O and MRR_C are extended encodings that break action, operator and causal structure symmetries, respectively.

Test cases (i.e., input plans) were generated by finding plans for all first-order IPC STRIPS planning instances. To ensure a variety of plans, three planners of distinct "types" were used: the novelty-driven best-first search planner Dual-BFWS [81], the heuristic forward-search planner LAMA [108] and the SAT planner Madagascar [112]. Each (unique) plan was encoded with each encoding, and the resulting MaxSAT instances were preprocessed with MaxPre [77] and given to the Loandra MaxSAT solver. For the MRR_C encoding, PDG automorphisms were found with NAUTY [85]. Plan generation and encoding/optimisation were both limited to 8GB and 30m of CPU time at 3.2GHz.

Because the optimality criteria in Definitions 2.18 and 4.2 cannot compare POPs with different operators, POP flexibility is here assessed with the commonly-used [91, 95, 120] *flex* measure, which is computed from the proportion of operators that are not (transitively) ordered:

Definition 4.21. *The* flex *of POP* $P = \langle O, \theta, \prec \rangle$ *is defined as follows:*

$$\text{flex}(P) \stackrel{\text{def}}{=} 1 - \frac{|\prec|}{\sum_{i=1}^{|O|-1} i}.$$

A POP's *flex* ranges from 0 to 1 (a totally ordered and unordered POP, respectively) and is a strong indicator of the number of linearisations it represents [91].

4.5.2 Results

Results are summarised in Tables 4.1–4.5. Table 4.1 shows, for each domain, the average *flex* found the EOG baseline, and the percentage increase in flex provided by each encoding (e.g., for *cybersec*, $f_\text{EOG} = 0.04$ and $\text{MRR}_\text{O} = 98\%$ meaning that the average *flex* found by MRR_O is $0.04 \times 1.98 = 0.0792$). Table 4.5 shows the same for each planner. To allow a meaningful comparison, mean *flex* differences are computed from plans for which a solution was found by all encoders. All stated *flex* differences are statistically significant with $p < 0.01$ as calculated from a single-tailed, paired t-test. Table 4.2 shows the average run time for each encoder and domain, excluding plans that one or more encoders failed to encode due to memory limitations.

			Δ_{EOG}					
Domain	n	f_{EOG}	MR	MR_A	MRD	MRR	MRR_O	MRR_C
airport	28	0.25	29.9%	29.9%	0%	29.9%	29.9%	29.9%
childsnack	82	0.68	4.3%	4.3%	8.1%	17.8%	18.4%	16.4%
cybersec	90	0.04	88.6%	98%	0%	86%	98%	92.2%
depot	21	0.35	5.1%	5.1%	27.1%	45.3%	45.7%	42%
driverlog	41	0.36	2.5%	2.5%	18.1%	26.5%	35.4%	31.2%
elevators	54	0.42			11.9%	28.5%	29.2%	27.7%
freecell	68	0.14			36.4%	83.6%	89.3%	80.4%
hiking	22	0.17				40.9%	42.2%	40.8%
mprime	69	0.23			54.8%	61.1%	60.9%	60%
mystery	43	0.18	0%	0%	56.7%	62%	62.1%	61.8%
nomystery	39	0.05	8.4%	8.4%		30.3%	31.2%	30.5%
pipesworld	30	0.21			62.9%	80.8%	80.8%	77%
psr-small	140	0.04	21.8%	21.8%	0%	21.8%	21.8%	21.8%
rovers	56	0.66	1.1%	1.1%	6%	12.5%	12.7%	11.3%
satellite	26	0.4	0%	0%	7.9%	13.1%	14.2%	14.3%
scanalyzer	21	0.36			30.9%	32%	31.9%	
tpp	22	0.27	8.8%	8.8%	12.3%	29.5%	29.9%	26.5%
transport	23	0.33	0%	0%	24.8%	35.4%	35.4%	35.4%
woodworking	71	0.87	0%	0%	3.1%	3.3%	3.3%	3.2%
zenotravel	35	0.4	0.9%	0.9%	17.7%	20.9%	22%	22%
ALL	1,209	0.34	3.2%	3.3%	12.1%	20.2%	21.1%	19.5%

Table 4.1: Mean *flex* increase by domain. For each domain, f_{EOG} is the mean *flex* produced by EOG, and for each encoder, Δ_{EOG} is the mean % *flex* increase over EOG. All *flex* values are computed from the n plans for which every encoder found a solution, and all Δ_{EOG} values are statistically significant with $p < 0.01$ as calculated from a single-tailed, paired t-test. Empty cells indicate no data, or no significant difference.

Table 4.3 indicates the *coverage* for each domain and encoding, that is, the proportion for which a satisficing or optimal solution was found within the time limit (where optimal means either a minimum de/reorder, or minimum reinstantiated de/reorder, as appropriate to the encoding), and Table 4.4 indicates the *optimal coverage* that is, the proportion of plans that were optimally relaxed.

Domains where no plan was improved by any technique (*agricola*, *organic-synth-split*, *pegsol*, *snake*, *sokoban*, *termes*, *visitall*) have been excluded from all tables, and domains where no technique produced a statistically significant improvement over EOG (*barman*, *data-network*, *floortile*, *ged*, *gripper*, *logistics98*, *organic-synthesis*, *parcprinter*, *parking*, *tetris*, *thoughful*) have been excluded from Table 4.1.

Effectiveness of EOG The results confirm Muise et al. [91]'s observation that in all cases, EOG finds a minimum deorder of the input plan, despite a lack of optimality guarantees. Additionally, EOG takes $< 3s$ in 88% of test cases, with a coverage close to 100%. MD is therefore excluded from the rest of the discussion.

As well as consistently finding minimum deorders, EOG found a minimum reorder in 65% of plans solved optimally by MR, a minimum reinstantiated deorder in 56.6% of the plans solved optimally by MRD, and a minimum reinstantiated reorder in 46% of the plans solved optimally by MRR_O. This indicates that in nearly half of cases, a minimum reinstantiated reorder can be found without modifying variables.

Domain	EOG	MR	MR_A	MRD	MRR	MRR_O	MRR_C
airport	0.02	0.1	0.1	3.26	12.35	5.9	5.91
barman	0.02	30	18.7	2.64	30	30	30
childsnack	0	15.54	7.82	20.62	29.99	29.99	29.99
cybersec	0.1	9.33	7.91	0.15	9.54	8.15	8.23
data-network	0.02	6.94	4.28	25.53	30	30	30
depot	0.01	9.34	7.08	15.85	27.89	26.38	25.28
driverlog	0.01	4.38	2.92	8.07	23.13	16.35	16
elevators	0	3.64	3.17	6.47	28.79	23.97	23.56
floortile	0.01	20.04	0.51	17.34	30	30	30
freecell	0.01	1.13	0.94	8.02	26.66	21.61	23.82
ged	0.01	9.41	9.1	9.46	11.18	10.07	10.38
gripper	0.01	26.16	24.15	23.91	28.89	25.36	27.23
hiking	0.01	3.88	3.08	9.54	27.3	21.48	25.76
logistics98	0.02	5.81	2.22	20.87	28.65	26.89	26.61
mprime	0.01	0.16	0.16	12.42	16.47	14.35	14.95
mystery	0.01	0.06	0.04	5.01	11.06	8.17	9.71
nomystery	0.06	0.91	0.98	4.11	25.93	14.79	20.24
organic-synthesis	0.02	0.05	0.05	12.35	12.62	12.65	11.24
parcprinter	0.01	0.1	0.08	14.2	25.18	24.85	21.25
parking	0.01	0.76	0.71	26.73	30	29.4	30
pipesworld	0.01	0.89	0.17	7.99	26.2	22.49	25.1
psr-small	0.01	1.99	0.78	0.03	2.04	0.79	0.95
rovers	0.03	9.46	5.55	15.6	24.57	23.77	22.5
satellite	0.01	1.26	0.91	22.13	27.52	26.27	26.27
scanalyzer	0	6.26	6	20.35	28.98	24.65	28.06
tetris	0.01	0.59	0.28	20.51	28.12	25.4	27.75
thoughtful	0.02	4.83	0.23	11.6	30	22.52	26.63
tpp	0.01	13.05	7.15	19.35	24.32	22.94	22.36
transport	0.01	7.57	1.31	16.91	30	26.77	27.97
woodworking	0.01	0.07	0.07	15.96	21.25	20.78	15.44
zenotravel	0.01	1.92	0.92	11.63	22.81	20.6	20.28
ALL	0.02	5.61	3.41	12.74	23.17	20.51	20.96

Table 4.2: Run time. For each domain and encoder, the average run time in minutes, limited to plans that were successfully encoded by all encoders.

Benefit of Optimisation The aim of the optimisations described in Definitions 4.5 and 4.6 is to reduce the likelihood of the encoding process exceeding the memory limits. These optimisations can be assessed by comparing the proportion of test cases that are succesfully encoded by MR and MRR with the proportion encoded by MR_N and MRR_N, respectively. These unoptimised variants do not minimise the number of propositions appearing in the encoding, that is, when building constraints, the proposition sets $\mathcal{P}_{\prec}$ and $\mathcal{P}_{\theta}$ contain a proposition for each pair of operators and variables, respectively, and $\mathcal{P}_{\rightarrow}$ contains a proposition for every possible causal link.

Table 4.3 indicates that overall, MRR encodes 52.1% of test cases to MRR_N's 48%, indicating that the optimisations result in a 3.1pp[11] coverage increase. More significant benefits are seen when the results are split by domain, for example in *airport*, *barman*, *parcprinter* and *woodworking*, the optimisations increase coverage by 7.8pp, 9.5pp, 40.9pp and 26.7pp, respectively. However, the optimisations have little benefit on the MR encoding, with Table 4.3 showing a 0.2pp coverage increase overall. This difference suggests that the real benefit of the optimisations is the reduction in the number of variable equality constraints (i.e., produced by Formulae 4.11–4.13) rather than ordering constraints (Formulae 4.8–4.9).

[11] A percentage point pp measures the difference between two percentages, e.g., 50% and 55% differ by 10% but 5pp.

Domain	n	MR_N	MR	MR_A	MRD	MRR_N	MRR	MRR_O	MRR_C
airport	115	63.5%	63.5%	63.5%	27%	16.5%	24.3%	24.3%	24.3%
barman	74	40.5%	37.8%	39.2%	82.4%	8.1%	17.6%	32.4%	8.1%
childsnack	90	100%	100%	100%	98.9%	88.9%	92.2%	97.8%	91.1%
cybersec	90	100%	100%	100%	100%	100%	100%	100%	100%
data-network	22	81.8%	86.4%	86.4%	50%	13.6%	18.2%	18.2%	9.1%
depot	63	88.9%	88.9%	90.5%	69.8%	46%	46%	41.3%	33.3%
driverlog	59	94.9%	94.9%	96.6%	78%	72.9%	71.2%	72.9%	71.2%
elevators	101	65.3%	65.3%	65.3%	59.4%	59.4%	59.4%	57.4%	53.5%
floortile	65	100%	100%	100%	96.9%	10.8%	13.8%	6.2%	1.5%
freecell	147	100%	100%	100%	95.9%	49.7%	51%	55.1%	46.3%
ged	91	91.2%	92.3%	92.3%	64.8%	60.4%	60.4%	62.6%	61.5%
gripper	60	86.7%	86.7%	95%	95%	20%	20%	25%	20%
hiking	82	93.9%	93.9%	93.9%	92.7%	39%	35.4%	46.3%	26.8%
logistics98	101	66.3%	67.3%	68.3%	40.6%	18.8%	17.8%	27.7%	16.8%
mprime	97	100%	100%	100%	76.3%	77.3%	78.4%	79.4%	76.3%
mystery	51	100%	100%	100%	94.1%	92.2%	92.2%	94.1%	86.3%
nomystery	95	100%	100%	98.9%	93.7%	52.6%	50.5%	62.1%	43.2%
organic-synthesis	3	100%	100%	100%	100%	100%	100%	100%	100%
parcprinter	118	100%	100%	100%	100%	55.9%	96.6%	99.2%	98.3%
parking	80	100%	100%	100%	35%	5%	5%	5%	2.5%
pipesworld	89	100%	100%	100%	89.9%	39.3%	41.6%	52.8%	33.7%
psr-small	140	100%	100%	100%	100%	100%	100%	100%	100%
rovers	120	82.5%	82.5%	81.7%	63.3%	47.5%	47.5%	49.2%	46.7%
satellite	100	85%	89%	88%	58%	45%	51%	27%	31%
scanalyzer	104	97.1%	97.1%	98.1%	77.9%	36.5%	35.6%	50%	20.2%
tetris	79	96.2%	96.2%	96.2%	49.4%	17.7%	15.2%	32.9%	10.1%
thoughtful	40	100%	100%	100%	100%	20%	20%	32.5%	22.5%
tpp	89	66.3%	65.2%	67.4%	51.7%	27%	29.2%	41.6%	24.7%
transport	98	70.4%	70.4%	71.4%	58.2%	31.6%	33.7%	44.9%	26.5%
woodworking	120	100%	100%	100%	89.2%	52.5%	79.2%	78.3%	59.2%
zenotravel	57	94.7%	94.7%	94.7%	77.2%	70.2%	71.9%	64.9%	63.2%
ALL	2,640	88.9%	89%	89.4%	75.6%	48%	52.1%	55.6%	46.6%

Table 4.3: Coverage. For each domain and encoder, the percentage of the n plans for which a satisficing or optimal solution was found.

Benefit of reinstantiation To assess the practical benefit of reinstantiation, the coverage, run time and final *flex* of MRD, MRR_O and MRR_C are compared with those of MR_A and EOG.

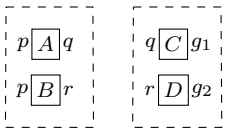
Overall, MRR_O and MRR_C provide a 21.1% and 19.5% *flex* increase, respectively, over EOG. This contrasts with the 3.3% *flex* increase provided by MR_A, meaning that reinstantiation has yielded a massive further *flex* increase of 17.8%. Furthermore, they consistently improve on EOG and MR_A: MRR_O times out before finding as flexible a solution as EOG in $< 1\%$ of the plans solved by both, with similar results for MR_A. MRR_C performs similarly.

MRR_O and MRR_C improve on EOG in 20 and 19 domains, respectively. The largest differences are in *freecell*, *mprime*, *mystery* and *pipesworld*, where MRR_O provides *flex* increases of 89.3%, 60.9%, 62.1% and 80.8%, respectively (and MRR_C is similar), while MR_A provides no significant benefit over EOG.

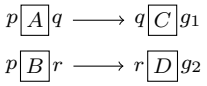
However, this additional flexibility comes at significant computational cost. MRR_O and MRR_C have a mean coverage of 55.6% and 46.6%, respectively, and their mean run time increases by an order of magnitude, to 20.51m and 20.96m, respectively. Coverage drops significantly in *floortile* (6.2% and 5%) and *parking* (5% and 2%).

Domain	n	MR_N	MR	MR_A	MRD	MRR_N	MRR	MRR_O	MRR_C
airport	115	63.5%	63.5%	63.5%	27%	16.5%	18.3%	24.3%	24.3%
barman	74	0%	0%	23%	75.7%	0%	0%	0%	0%
childsnack	90	50%	50%	76.7%	34.4%	0%	0%	0%	0%
cybersec	90	71.1%	71.1%	78.9%	100%	71.1%	71.1%	78.9%	77.8%
data-network	22	68.2%	68.2%	77.3%	13.6%	0%	0%	0%	0%
depot	63	61.9%	63.5%	71.4%	42.9%	6.3%	6.3%	12.7%	15.9%
driverlog	59	81.4%	81.4%	86.4%	71.2%	23.7%	23.7%	44.1%	45.8%
elevators	101	60.4%	61.4%	61.4%	55.4%	3%	4%	15.8%	15.8%
floortile	65	35.4%	35.4%	100%	44.6%	0%	0%	0%	0%
freecell	147	97.3%	97.3%	98%	80.3%	11.6%	12.2%	29.9%	21.8%
ged	91	64.8%	63.7%	65.9%	64.8%	59.3%	59.3%	61.5%	60.4%
gripper	60	11.7%	11.7%	18.3%	20%	3.3%	3.3%	15%	8.3%
hiking	82	89%	89%	91.5%	69.5%	9.8%	9.8%	32.9%	15.9%
logistics98	101	51.5%	52.5%	60.4%	19.8%	3%	3%	6.9%	6.9%
mprime	97	100%	100%	100%	66%	52.6%	54.6%	61.9%	59.8%
mystery	51	100%	100%	100%	82.4%	64.7%	66.7%	74.5%	68.6%
nomystery	95	100%	100%	98.9%	91.6%	12.6%	12.6%	45.3%	30.5%
organic-synthesis	3	100%	100%	100%	100%	100%	100%	100%	100%
parcprinter	118	100%	100%	100%	54.2%	16.9%	16.9%	16.9%	29.7%
parking	80	98.8%	98.8%	98.8%	11.3%	0%	0%	2.5%	0%
pipesworld	89	98.9%	97.8%	100%	76.4%	13.5%	13.5%	27%	18%
psr-small	140	95%	95%	97.9%	100%	95%	95%	97.9%	97.1%
rovers	120	56.7%	57.5%	69.2%	40%	15%	15%	18.3%	20.8%
satellite	100	82%	84%	85%	22%	7%	7%	10%	10%
scanalyzer	104	78.8%	78.8%	79.8%	32.7%	2.9%	3.8%	20.2%	6.7%
tetris	79	93.7%	94.9%	96.2%	31.6%	6.3%	6.3%	15.2%	7.6%
thoughtful	40	82.5%	82.5%	100%	62.5%	0%	0%	25%	12.5%
tpp	89	39.3%	39.3%	52.8%	23.6%	12.4%	12.4%	15.7%	16.9%
transport	98	55.1%	55.1%	69.4%	32.7%	0%	1%	8.2%	5.1%
woodworking	120	100%	100%	100%	48.3%	30.8%	30.8%	32.5%	49.2%
zenotravel	57	87.7%	89.5%	93%	59.6%	22.8%	22.8%	33.3%	35.1%
ALL	2,640	74.4%	74.6%	81.2%	53.3%	20.7%	21%	29.3%	27.5%

Table 4.4: Optimal coverage. For each domain and encoder, the percentage of the n plans for which an optimal solution was found.



(a) Parallel Plan P_7 with a makespan of two, and four (implicit) ordering constraints.



(b) POP P_8, a minimal deorder of P_7 with two ordering constraints.

Figure 4.5: Parallel plan deordering example.

The less general MRD approach provides less additional *flex* at less computational cost. With a mean coverage and run time of 75.6% and 12.74m, respectively, it provides an overall *flex* increase of 12.1% over EOG and statistically significant increases in 15 domains, *pipesworld* being the highest (62.9%).

Influence of planner The source of the input plan also influences plan flexibility. Madagascar produces *parallel plans*: generalised plans where each step is a set of actions with non-contradictory postconditions:

Definition 4.22. *A* **parallel plan** $T = \langle S_1, \ldots, S_n \rangle$ *is a sequence of sets of actions* s.t. *for all* a_1, a_2 *and* $1 \leq k \leq n$, *if* $a_1, a_2 \in S_k$ *and* $\mathsf{prods}(a_1, l)$ *then* $\neg\mathsf{prods}(a_2, \neg l)$.

				Δ_{EOG}					
Planner	n	f_P	f_{EOG}	MR	MR_A	MRD	MRR	MRR_O	MRR_C
Dual-BFWS	379	0	0.31	4%	4%	18.6%	30.5%	31.6%	29.9%
LAMA	415	0	0.29	2.2%	2.2%	7.2%	13.6%	14.2%	13.2%
Madagascar	361	0.07	0.4	3.4%	3.6%	10.6%	17.2%	18.2%	16.6%

Table 4.5: Mean *flex* increase by planner. For each planner, f_P is the mean *flex* of the initial plan, f_{EOG} is the mean *flex* produced by EOG, and for each encoder, Δ_{EOG} is the mean % *flex* increase over EOG. All *flex* values are computed from the n plans for which every encoder found a solution, and all Δ_{EOG} values are statistically significant with $p < 0.01$ as calculated from a single-tailed, paired t-test.

Parallel plans are special cases of POPs. A parallel plan $T = \langle S_1, \ldots, S_n \rangle$ is equivalent to the POP $P = \langle A, \prec \rangle$ where $A = S_1 \cup \cdots \cup S_n$ and $a_1 \prec a_2$ *iff* $a_1 \in S_i$, $a_2 \in S_j$ and $i < j$.

A parallel plan with a minimal makespan is not necessarily a minimum (or even minimal) deorder. For example, parallel plan P_7 in Figure 4.5a has four (implicit) ordering constraints: both A and B must precede C and D. The division of P_7 into steps has thus resulted in two unnecessary ordering constraints that can be removed to produce plan P_8 in Figure 4.5b.

On average, EOG increases the initial mean *flex* of the parallel plans produced by Madagascar from 0.07 to 0.4, and MRR_O further improves on this by 18.2%. Over the sequential (*flex* = 0) plans generated by Dual-BFWS and LAMA, EOG produces POPs with a mean *flex* of 0.31 and 0.29, respectively, which is improved upon by MRR_O by 31.6% and 14.2%, respectively Thus, while Dual-BFWS benefits most from reinstantiation, optimised Madagascar plans are the most flexible.

Benefit of symmetry breaking To assess the practical benefit of the symmetry breaking techniques presented in Section 4.4, the coverage, run time and final *flex* of MR_A, MRR_O and MRR_C are compared with those of MR and MRR.

Overall, Tables 4.2 and 4.4 show that symmetry breaking allows more plans to be optimally relaxed (i.e., more minimum de/reorders or minimum reinstantiated de/reorders are found), with a significantly reduced run time. But interestingly, Tables 4.1 and 4.3 show that this results in little increase in coverage or *flex*, that is, the benefit of symmetry breaking is not that it allows better solutions to be found, but rather, it *allows the optimality of existing solutions to be proved.* Alternatively, this suggests that MR and MRR are capable of finding an optimal solution within the time limit, but cannot exhaustively prove this without the search space reduction provided by symmetry breaking.

For example, while MR_A's coverage is just 0.4pp greater than MR's, and it provides a 3.3% *flex* increase over EOG to MR's 3.2%, its optimal coverage is 6.6pp greater and its average run time 2.2m quicker. With the exception of *nomystery*, this improvement is consistent across domains, with the biggest changes seen in *barman* (23pp increase in optimal coverage and 11.3m run time reduction) and *floortile* (64.6pp and 19.53m).

The addition of operator symmetry breaking improves MRR in a similar way. It provides little increase in *flex*–MRR_O provides a 21.1% *flex* increase over EOG to MRR's 20.2%–and while its coverage improves on MRR by 3.5pp, coverage drops in five domains, with *satellite* the most significant (24pp coverage decrease). However, MRR_O sees an increase in optimal coverage of 8.3pp and a run time reduction of 2.66m. Improvements are consistent across domains and largest in *hiking* (23.1pp increase in optimal coverage and 5.82m run time reduction) and *thoughful* (25pp and 7.48m). This suggests that symmetry breaking has a "polarising" effect, resulting in more timeouts (i.e., reduced coverage) but also more optimal solutions. For example, in *rovers*, MRR_O has a coverage decrease of 4.7pp, but an optimal coverage increase of 7.4pp.

MRR_C also displays this polarisation. It performs, on average, slightly worse than MRR: it has less coverage, most significantly in *scanalyzer* (a decrease of 29.8pp) and a 5.5pp decrease overall, and provides only 19.5% *flex* increase over EOG to MRR's 20.2%. Nevertheless, its optimal coverage and run time improves on MRR by 6.5pp, and 2.21m, respectively. The polarisation is most pronounced in *woodworking*, where MRR_C has 20pp less coverage than MRR, but 18.4pp more optimal coverage.

MRR_C achieves better *flex* and/or optimal coverage than MRR_O in 8 domains, however, any common property causing this is not immediately clear. While it is tempting to attribute its success in *satellite*, *woodworking* and *zenotravel* to the large number of object symmetries, MRR_C also outperforms MRR_O in *parcprinter* and *rovers*, which display no such symmetries, and performs poorly in the highly symmetrical *scanalyzer*.

4.6 Discussion

The main contributions of this chapter were as follows:

- This chapter introduced, and formally defined, the notions of *reinstantiated reordering* and *reinstantiated deordering*, transformations of a plan under which both ordering constraints and variable bindings can be changed. These ideas generalise the well-studied *deordering* and *reordering*, with the aim of finding optimised POPs with fewer ordering constraints than would be possible had the bindings remained unchanged. It was shown that finding an optimally reinstantiated POP is NP-hard and cannot be approximated within a constant factor.

- A technique was presented for encoding the problem of finding a minimum reinstantiated de/reorder of a POP into an instance of the partial weighted MaxSAT problem. This encoding is an extension and optimisation of Muise et al. [91]'s approach.

- A number of symmetry breaking techniques were introduced to counter the exponential increase in search space that results from allowing variable bindings to change. The notion of a *causal structure symmetry* was introduced, a symmetry resulting from equivalent combinations of (potential) causal links between operator preconditions and postconditions. To detect these symmetries, a POP's *plan description*

graph was defined, an undirected coloured graph that encodes relevant aspects of the POP's structure, with automorphisms that correspond to the POP's symmetries.

- An empirical evaluation over all previous IPC STRIPS domains assessed the MaxSAT-based reinstantiation approaches' ability to minimise a POP's ordering constraints, as compared to the polynomial, non-optimal but effective EOG technique. Results show that in 55.6% of cases, reinstantiation provides a 21.1% *flex* increase over EOG. Interestingly, in nearly half of cases EOG found a minimum reinstantiated reorder, indicating that the original choice of variables already allowed for optimal reordering. A comparison by planner showed that Dual-BFWS benefits most from reinstantiation, with MRR_O providing a 31.6% *flex* increase, and that the optimised Madagascar plans are the most flexible, with an average *flex* of 0.47. While symmetry breaking does not result in a further increase in *flex*, it does speed up search and allow the optimality of solutions to be exhaustively proved.

This chapter presented a practical technique for plan optimisation through the modification of both ordering and variable binding constraints. Results show that in 55.6% of cases, reinstantiation provides a *flex* increase of 21.1% over the EOG baseline in 20.51m, with results varying by domain.

The addition of symmetry breaking allows more plans to be optimally relaxed, but interestingly does not result in a further increase in *flex*. This suggests that the encodings without symmetry breaking do find optimal solutions, but time out before this can be proved by the MaxSAT solver. Comparing the two symmetry breaking techniques shows that their success is dependent on domain, but it remains unclear what aspects of a domain allow a technique to succeed or not. Determining this is complicated by the fact that solvers rely on variable ordering and branching heuristics that can clash with symmetry breaking constraints [61, 123], and the choice of encoding of lex-leader constraints can significantly affect run time [37].

There are clear similarities between the causal structure symmetries introduced in this chapter and the commonly studied idea of *strong stubborn sets* (Section 2.3.4.4), a pruning technique in state-space planning that avoids exploring any two plans that are identical up to an interleaving of causally unrelated sequences of actions. Both approaches examine possible causal connections between actions with the aim of reducing symmetry, and recognise that plans with equivalent causal structures produce redundancy in the search space. However, beyond the obvious differences (stubborn sets are used by state-space planners to dynamically break symmetries, whereas causal structures are used to statically break symmetries when optimising POPs), the two approaches also break different types of symmetry. Stubborn sets are used to (broadly) prune classical plans with identical causal structures and actions, but symmetrical orderings, whereas the approach here is to prune POPs with symmetrical causal structures, but (possibly) different actions (i.e., bindings) and orderings.

As reflected by the low coverage of reinstantiation-based encodings, the increase in *flex* produced by allowing variable bindings to change comes with considerable computational cost. Additionally, in a significant number of

cases, the additional computation simply reveals that the original bindings were already optimal.

Whether MRR_O and MRR_C is preferable to less costly methods such as EOG depends on the application. If offline preprocessing time is available and execution-time flexibility is paramount, or the application domain is one where reinstantiation can quickly and consistently improve plan flexibility (e.g., *mystery*), then reinstantiation is clearly worthwhile.

This work generalises the POP optimality definitions of Bäckström [8] and optimisation techniques of Muise et al. [91] to allow changes in variable bindings. As discussed in Section 2.2.2, other POP optimisation techniques have been studied, for example those that de/reorder via block decomposition [120], speed up search with MILP models [116], or optimise plan size or makespan [11, 91, 96, 116]. However, unlike the work presented here, these techniques do not allow variable bindings to change. Thus, further work could investigate whether generalising these techniques by allowing reinstantiation results in more flexible block decompositions or further reductions in plan size or makespan, and whether a MILP encoding of MRD, MRR or their symmetry breaking extensions improves their execution time.

CHAPTER 5

Plan Relaxation via Action Debinding and Deordering

5.1 Introduction

Under the *least commitment* approach to planning, an agent maximises its flexibility by postponing decisions regarding actions' orderings and parameters for as long as possible. While seminal work (Section 2.2.2.1), and the previous chapter, studied the problem of minimising commitment to action ordering, the equally important problem of minimising commitment to a particular set of domain objects has received less attention.

A common way to achieve least commitment is to relax a totally ordered plan into a minimally constrained partial-order plan (POP) through the processes of *deordering*, in which ordering constraints can be removed but not added, and *reordering*, in which any modification can be made. These techniques have two notable limitations. Firstly, de/reordering only modifies actions' ordering not parameters, and since the original plan was likely not generated with flexibility in mind, remaining committed to the original choice parameters can unnecessarily restrict the ordering options. Secondly, the result of the optimisation process is a POP, which offers flexibility regarding actions' orderings, but not their parameters.

Chapter 4 addressed the first limitation by introducing the processes of *reinstantiated deordering* and *reinstantiated reordering*, generalisations of standard de/reordering that simultaneously optimise both ordering and variable bindings, with the aim of finding the bindings that allow for the fewest ordering constraints. However, this approach cannot address the second limitation. While reinstantiated de/reordering allows the plan's actions' parameters to change at *optimisation time*, the result is still a POP with fixed parameters that cannot be varied at *execution time*.

Thus, this chapter studies the problem of relaxing a plan into one that *allows both action ordering and domain objects to be selected at execution time*. The notion of a *partial plan* is introduced: a generalised plan comprising a set of operator types to be executed, and a set of constraints defining the allowable combinations of orderings and variable bindings. Of particular interest is the problem of *relaxing* a plan into a partial plan by

1. $navigate_1(\mathsf{R}_1, \mathsf{W}_1, \mathsf{W}_2)$
2. $sample\text{-}soil(\mathsf{R}_1, \mathsf{W}_2)$
3. $navigate_2(\mathsf{R}_1, \mathsf{W}_2, \mathsf{W}_3)$
4. $sample\text{-}rock(\mathsf{R}_1, \mathsf{W}_3)$

(a) Plan P_1.

$navigate_1(\mathsf{R}_1, \mathsf{W}_1, \mathsf{W}_2) \prec sample\text{-}soil(\mathsf{R}_1, \mathsf{W}_2)$
$navigate_2(\mathsf{R}_2, \mathsf{W}_1, \mathsf{W}_3) \prec sample\text{-}rock(\mathsf{R}_2, \mathsf{W}_3)$

(b) Partial-order plan P_2.

Operators:
$navigate_1(r_1, w_1, w_2)$
$sample\text{-}soil(r_2, w_3)$
$navigate_2(r_3, w_4, w_5)$
$sample\text{-}rock(r_4, w_6)$

Constraints:
(i) $navigate_1 \prec sample\text{-}soil$, $navigate_2 \prec sample\text{-}rock$.
(ii) $r_1 = r_2$ and $r_3 = r_4$.
(iii) $w_2 = w_3 = \mathsf{W}_2$, $w_5 = w_6 = \mathsf{W}_3$
(v) If $r_1 \neq r_3$ then $w_1 = w_4 = \mathsf{W}_1$.
(iv) If $r_1 = r_3$ then either:
$sample\text{-}soil \prec navigate_2$, $w_1 = \mathsf{W}_1$ and $w_4 = \mathsf{W}_2$, or
$sample\text{-}rock \prec navigate_1$, $w_1 = \mathsf{W}_3$ and $w_4 = \mathsf{W}_1$.

(c) Partial plan P_3.

Figure 5.1: Three plans from the *rovers* domain. Subscripts have been used to better distinguish between actions of the same type.

"lifting" its ordering and variable binding constraints. Interestingly, finding a *minimum relaxation* of a plan, that is, a partial plan with constraints that are as relaxed as possible while remaining valid, is a polynomial time operation. However, this encouraging result is undermined by the fact that the problem of *instantiating* a partial plan, that is, finding a ground, totally ordered plan that satisfies its constraints, is intractable, clearly a serious drawback at execution time.

Nevertheless, a parameterised complexity analysis reveals that islands of tractability exist within the space of partial plans, and an algorithm of bounded complexity is introduced that maximises a partial plan's flexibility while keeping its "complexity" (as measured by the treewidth of its primal graph) below an input parameter.

5.1.1 Example

Consider again the small planning problem from the (reduced version of the) IPC *rovers* domain described in Section 4.1.1. The domain objects comprise two rovers (R_1 and R_2) and three waypoints (W_1–W_3). Both rovers begin at W_1, there are soil and rock samples at W_2 and W_3, respectively, and paths connect all waypoints. The goal is to gather both samples.

Figure 5.1 shows three solutions to this problem. Classical plan P_1 is an optimal solution in which R_1 navigates to W_2 and collects the soil sample, then navigates to W_3 and collects the rock sample. Partial-order plan P_2 is a *reinstantiated deorder* (Definition 4.1) of P_1 in which R_2 collects the rock sample. This change in domain objects allows the ordering to be relaxed: the actions can be executed in any order so long as the rovers are moved into place before collecting the samples.

While more flexible than P_1, P_2 only allows variation in the order of actions, not to the domain objects used in the course of executing the plan. As a result, P_2 does not permit the other ways in which domain objects

can be utilised. For example, R_1 could instead be used to collect the rock sample, and R_2 the soil sample, or a single rover could collect both samples, in which case they could be collected in any order. A generalised plan that *allows domain objects to be selected at execution time* could encapsulate these variations. This would provide the executing agent with more flexibility, increasing its capacity to recover from unexpected events (e.g., paths or rovers being unavailable) without resorting to replanning.

The *partial plan* depicted in Figure 5.1c is a further generalisation of P_1 and P_2, and a compact representation of the above options. On the left, the plan's operators are shown in their uninstantiated form, that is, constants have been replaced by variables, and on the right is a list of constraints that define the allowable combinations of operator orderings and variable bindings. Constraints *(i)–(iii)* enforce the requirement that a rover must be moved to the correct waypoint before collecting a sample. The cases when the samples are collected by different rovers are covered by constraint *(iv)*. If $navigate_1$ and $navigate_2$ move different rovers (i.e., $r_1 \neq r_2$), then they must move them from the starting location W_1 to W_2 and W_3, respectively. Constraint *(v)* covers the cases when samples are collected by the same rover (i.e., $r_1 = r_2$). If the soil sample is collected first, the constraints ensure that the rover first moves from W_1 to W_2, and then from W_2 to W_3, but not before collecting the sample at W_2. Similar constraints are required for when the rock sample is collected first.

Partial plan P_3 generalises both P_1 and P_2, in that P_1 satisfies P_3's constraints, as does every plan that satisfies P_2's ordering constraints. Indeed, P_3 is "minimally constrained" in that it encapsulates exactly those orderings and variable bindings of the four operators that result in a valid plan, and no others.

This chapter will continue as follows. Section 5.2 formally defines partial plans, and provides a complexity analysis of the problems of instantiating and validating them. Section 5.3 introduces a number of criteria to compare the degree of flexibility provided by a partial plan, based on the relative strengths of their constraints and the computational cost of finding an instantiation. Section 5.4 introduces a restricted form of partial plans that represent constraints as a set of allowable causal links, and introduces a greedy fpt-algorithm, MkTR, that searches for optimally relaxed plans of this form. Section 5.6 empirically compares MkTR with the non-optimal but nevertheless effective EOG plan deordering technique of Kambhampati and Kedar (Section 2.2.2.2). Results show that while some domains resist any kind of optimisation, MkTR on average finds a quadratic-time instantiatable partial plan with 163.2% more instantiations than EOG, *and in some domains can relax plans that cannot be improved by either* EOG *or any de/ordering techniques studied in Chapter 4*.

5.2 Partial Plans

A *partial plan* is a generalised plan that specifies which operators must be executed, without fully specifying their order or object values for their parameters. Instead, it provides a *constraint formula* that defines the allowable combinations of orderings and variable bindings. Thus, unlike a

POP, which compactly represents a set of plans with different orderings, a partial plan represents a set of plans that can differ both in their orderings and domain objects.

More formally, a constraint formula is an unquantified Boolean formula in which each atom is either a codesignation constraint between free variables or between a free variable and a constant, or an ordering constraint over two operators:

Definition 5.1. *Let $\mathcal{L} = \langle V, C, P \rangle$ be a first-order language where V, C, and P are finite sets of variable, constant, and predicate symbols, respectively, and let $\mathcal{O}$ be a set of operators constructed in $\mathcal{L}$. The* **constraint language** *$\mathcal{L}_C$ is generated using the following context-free grammar, where $x, y \in V$, $c \in C$ and $o_1, o_2 \in \mathcal{O}$:*

$$\phi ::= \top \mid \bot \mid x = y \mid x = c \mid o_1 \preceq o_2 \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi).$$

As is standard, the ordering relation $\preceq$ is transitive and reflexive, and $o_1 \prec o_2$ is shorthand for $(o_1 \preceq o_2 \wedge o_2 \not\preceq o_1)$. Codesignation constraints restrict the allowable variable bindings. For all $x, y \in V$, $x = y$ requires that x and y be bound to the same constant, and for all $x \in V$ and $c \in C$, $x = c$ requires that x be bound to c. These semantics can be defined formally:

Definition 5.2. *For any classical plan $\vec{a}$, terms $t_1, t_2 \in V \cup C$, operators $o_i, o_j \in \mathcal{O}$ and constraint formulae $\phi_1, \phi_2 \in \mathcal{L}_C$, the $\models$ and $\equiv$ operators are defined as follows:*

$$\vec{a} \models t_1 = t_2 \text{ iff } \vec{a} = \langle o_1, \ldots, o_n \rangle\theta \text{ and } \theta(t_1) = t_2.$$
$$\vec{a} \models o_i \prec o_j \text{ iff } \vec{a} = \langle o_1, \ldots, o_n \rangle\theta \text{ and } 1 \leq i < j \leq n.$$
$$\vec{a} \models \phi_1 \wedge \phi_2 \text{ iff } \vec{a} \models \phi_1 \text{ and } \vec{a} \models \phi_2.$$
$$\vec{a} \models \phi_1 \vee \phi_2 \text{ iff } \vec{a} \models \phi_1 \text{ or } \vec{a} \models \phi_2.$$
$$\phi_1 \models \phi_2 \text{ iff for all } \vec{a}, \text{ if } \vec{a} \models \phi_1 \text{ then } \vec{a} \models \phi_2.$$
$$\phi_1 \equiv \phi_2 \text{ iff } \phi_1 \models \phi_2 \text{ and } \phi_2 \models \phi_1.$$

A *partial plan* comprises a set of operators and a constraint formula:

Definition 5.3. *A* **partial plan** *is a tuple $P = \langle O, \phi \rangle$, where O is a finite set of operators and ϕ is a constraint formula.*

As with classical and partial-order plans (Section 2.1.2.2), it is assumed that a partial plan has a planning task "embedded" within it through the operators o_I and o_G, which simulate the initial state and goal condition, respectively. Additionally, it is assumed that no variable appears in more than one operator.

5.2.1 Validating and Instantiating Partial Plans

A partial plan is a compact representation of the set of classical plans that satisfy its constraints, that is, of its *ground instantiations*:

Definition 5.4. *Classical plan $\vec{a} = \langle o_1, \ldots, o_n \rangle\theta$ is a* **ground instantiation** *of partial plan $P = \langle O, \phi \rangle$ iff $\vec{a} \models \phi$ and $O = \{o_1, \ldots, o_n\}$.*

Plan $\vec{a}$ is a ground instantiation of partial plan $P = \langle O, \phi \rangle$ *iff* $\vec{a}$ contains precisely the operators in O, and satisfies all ordering and binding constraints in ϕ. (For brevity, when clear from context, such plans will be referred to as *instantiations*.)

Standard definitions of plan validity can be extended to cover partial plans. As a classical plan is assumed to contain actions representing the initial state and goal, it is valid *iff* it is executable (Section 2.1.2.2). This can be extended to partial plans. A partial plan is valid *iff* two conditions hold: it must be *sound*, that is, all of its instantiations must be executable, and, to prevent a plan with contradictory constraints from being trivially valid, it must also be *satisfiable*, that is, it must admit at least one instantiation:

Definition 5.5. *Let $P = \langle O, \phi \rangle$ be a partial plan. Then:*

- *P is* **sound** iff *every instantiation of P is executable,*
- *P is* **satisfiable** iff *there exists a classical plan $\vec{a}$ that is an instantiation of P, and*
- *P is* **valid** iff *it is satisfiable and sound.*

The following two theorems establish the complexity of determining the soundness and satisfiability of a partial plan:[1]

Theorem 5.1. Partial Plan Soundness. *Determining the soundness of a partial plan is* co-NP*-complete.*

Theorem 5.2. Partial Plan Satisfiability. *Determining the satisfiability of a partial plan is* NP*-complete.*

From the above, it follows that determining a partial plan's validity is the conjunction of an NP-complete problem and a co-NP-complete problem, that is, it is DP-complete:

Theorem 5.3. Partial Plan Validity. *Determining the validity of a partial plan is* DP*-complete.*

5.2.2 Special Cases of Partial Plans

Partial-order and classical plans are special cases of partial plans. A POP is a partial plan with fully specified variable bindings, and a classical plan is a partial plan which is both totally ordered and ground. Though not studied here in any detail, an additional special case is a *partially-instantiated plan*, which specifies a total order over the operators, but need not fully specify the variable bindings:

[1] Proofs for all theorems in this chapter are in Appendix D.

Definition 5.6. *Let $P = \langle O, \phi \rangle$ be a partial plan. Then:*

- *P is a* **partially-instantiated plan** iff *for all $o_1, o_2 \in O$, either $\phi \models o_1 \prec o_2$ or $\phi \models o_2 \prec o_1$,*
- *P is a* **partial-order plan** iff *there exists a POP $Q = \langle O, \theta, \prec' \rangle$* s.t. *$\phi \equiv \bigwedge\{x = c : \theta(x) = c\} \wedge \bigwedge\{o_1 \prec o_2 : (o_1, o_2) \in \prec'\}$, and*
- *P is a* **classical plan** iff *it is a partially instantiated plan, and for all $x \in \mathsf{vars}(O)$ there is a $c \in \mathsf{consts}(O)$* s.t. *$\phi \models x = c$.*

5.2.3 Parameterised Complexity of Partial Plans

Theorem 5.2 above shows that finding a classical plan that satisfies a partial plan's constraints is NP-complete. While it may seem that this result renders partial plans unsuitable for real-time use, further analysis from a parameterised complexity perspective shows that the problem of partial plan instantiation is tractable when the plan's constraint formula is subject to certain syntactic limitations.

Unlike classical complexity analysis, in which a problem's space and time requirements are measured as a function of the instance size, the parameterised complexity approach (Section 2.1.3), measures them *w.r.t.* to the instance size and one or more additional parameters. This fine-grained analysis aims to identify problems that are NP-hard but nevertheless "tractable" in that they are computable in a time that is polynomial *w.r.t.* the instance size and exponential *w.r.t.* a small parameter.

Many NP-hard graph problems display this form of tractability when parameterised with the *treewidth* (Section 2.1.4) of the graph, and so a common technique in parameterised complexity analysis is to show that an otherwise intractable problem is tractable when limited to instances with an underlying graphical structure of bounded treewidth.

This technique can be applied to the problem of partial plan instantiation. In this context, the underlying structure is the plan's *primal graph*, that is, the undirected graph containing a vertex for each variable and operator appearing in the plan, and an edge between vertices *iff* the corresponding variables or operators appear together in some clause of the partial plan's constraint formula. A partial plan's treewidth is simply the treewidth of its primal graph:

Definition 5.7. *Let $P = \langle O, \phi \rangle$ be a partial plan where ϕ is of the form $\phi_1 \wedge \cdots \wedge \phi_n$, where each ϕ_i is a disjunction of constraint formulae. Then:*

- *the* **primal graph** *of P is the graph $\langle V, E \rangle$ where $\mathrm{v}_t \in V$* iff *$t \in O \cup \mathsf{vars}(\phi)$, and $(\mathrm{v}_t, \mathrm{v}_s) \in E$* iff *there exists $1 \leq i \leq n$ such that both $t, s \in \mathsf{vars}(\phi_i) \cup \mathsf{consts}(\phi_i)$, and*
- *the* **treewidth** *of P, denoted $\mathsf{treewidth}(P)$, is equal to the treewidth of the primal graph of P.*

The satisfiability of a partial plan with bounded treewidth can be determined in polynomial time:

Theorem 5.4. PARAMETERISED PARTIAL PLAN SATISFIABILITY. *Determining the satisfiability of a partial plan P is in* XP *and is* W[1]*-hard when parameterised with* treewidth(P).

While computing a graph's treewidth is intractable, determining if it is bounded by some constant $k > 0$ is in FPT with respect to k [15]. As the size of the primal graph of a partial plan $P = \langle O, \phi \rangle$ is bounded by $(|\mathsf{vars}(\phi)| + |O|)^2$, it follows that determining whether a partial plan's treewidth is bounded by k is also in FPT:

Observation 5.1. *Determining whether* treewidth$(P) \leq k$ *for some partial plan P and $k > 0$ is in* FPT.

5.3 Optimality Criteria for Partial Plans

Since the purpose of partial plans is to increase execution-time flexibility, this section will discuss two ways to measure the amount of flexibility provided by a partial plan. A partial plan is a compact representation of a set of classical plans, each representing a different sequence of actions for realising the goal. The simplest flexibility measure is thus the size of this set, that is, *the number of instantiations the plan admits.*

However, an equally important consideration is the computational cost of instantiating the partial plan: a plan cannot, in practice, provide any flexibility if the executing agent cannot determine in a reasonable amount of time which actions are compatible with it. The second flexibility measure is thus the partial plan's tractability, that is, *the complexity of the problem of finding an instantiation of the plan.* While Theorem 5.2 above shows that instantiating a partial plan is, in general, intractable, Theorem 5.4 demonstrated that there are nevertheless islands of tractability within the space of partial plans. This section will show that partial plans can be classified by their instantiation cost.

5.3.1 Minimally Constrained Partial Plans

The relative "constrained-ness" of two partial plans can be determined by comparing the relative strengths of their constraint formulae. If $P = \langle O, \phi \rangle$ and $Q = \langle O, \phi' \rangle$ are two valid partial plans, then Q is a *relaxation* of P *iff* every model of ϕ is also a model of ϕ', and Q is a *minimum relaxation* of P *iff* it has the most models while remaining valid:

Definition 5.8. *Let $P = \langle O, \phi \rangle$ and $Q = \langle O, \phi' \rangle$ be partial plans. Then:*

- *Q is a* **relaxation** *of P* iff *they are both valid and $\phi \models \phi'$,*
- *Q is a* **proper relaxation** *of P* iff *it is a relaxation of P and $\phi' \not\models \phi$, and*
- *Q is a* **minimum relaxation** *of P* iff *it is a relaxation of P and there are no proper relaxations of Q.*

There are two key differences between the optimality criteria above and those found in the field of POP optimisation (e.g., Definitions 2.18

and 4.2). Firtly, while there is a clear difference between the idea of a *minimal* and *minimum* relaxation of a POP, there is no such distinction in the context of partial plans. Secondly, while the problems of finding a minimal or minimum relaxation of a POP are polynomial and NP-complete, respectively, a minimum relaxation of a partial plan can, interestingly, be found in polynomial time. These differences both derive from the fact that a minimum relaxation of any valid partial plan can be directly defined with reference to the Modal Truth Criterion [22] (MTC).

Typically, the MTC determines the validity of a classical plan or POP in the context of partial-order planning (Section 2.2.1) by requiring that it be *necessarily* true that the preconditions of all actions in the plan hold at the point when that action is executed. This can be generalised to cover partial plans—a partial plan meets the MTC *iff* all of its instantiations meet the MTC—and expressed as a constraint formula as follows:

Definition 5.9. *The* **modal truth criterion** *for a partial plan* $P = \langle O, \phi \rangle$ *requires that* $\phi \models \mathrm{MTC}(O)$*, where:*

$$\mathrm{MTC}(O) \stackrel{\text{def}}{=} \bigwedge_{o_c, q(\vec{s}):\mathsf{cons}(o_c, q(\vec{s}))} \mathrm{MTC}'(o_c, q(\vec{s})), \text{ where}$$

$$\mathrm{MTC}'(o_c, q(\vec{s})) \stackrel{\text{def}}{=} \bigvee_{o_p, \vec{u}:\mathsf{prods}(o_p, q(\vec{u}))} \vec{s} = \vec{u} \wedge o_p \prec o_c \wedge \bigwedge_{o_t, \vec{v}:\mathsf{thrtns}(o_t, q(\vec{v}))} \big(\vec{s} \neq \vec{v} \vee o_t \prec o_p \vee o_c \preceq o_t \vee \big[\bigvee_{o_w, \vec{r}:\mathsf{prods}(o_w, q(\vec{r}))} \vec{r} = \vec{s} \wedge o_t \prec o_w \prec o_c\big]\big).$$

The definition MTC′ applies the MTC to a single consumer, that is, a single precondition $q(\vec{s})$ of an operator o_c. The first line requires that there be some producer, that is, a postcondition $q(\vec{u})$ of an operator o_p, such that $\vec{s}$ and $\vec{u}$ are codesignated and $o_p \prec o_c$. The second and third lines require that for all threats to this link, that is, any postcondition $\neg q(\vec{v})$ of an operator o_t, either o_t is not ordered between o_p and o_c, $\vec{v}$ and $\vec{s}$ are not codesignated, or that there is a "white knight" o_w ordered after o_t that re-establishes the causal link. The definition MTC applies this to every consumer in the plan.

An alternative definition of partial plan soundness follows from the above. A partial plan is sound *iff* its constraint formula implies the MTC:

Theorem 5.5. Partial Plan MTC Soundness. *A partial plan* $P = \langle O, \phi \rangle$ *is sound* iff $\phi \models \mathrm{MTC}(O)$.

If $P = \langle O, \phi \rangle$ is a valid partial plan, then *the valid partial plan* $Q = \langle O, \mathrm{MTC}(O) \rangle$ *is a minimum relaxation of* P, and can be constructed in polynomial time:

Theorem 5.6. Minimum Relaxation. *A minimum relaxation of a valid partial plan can be found in polynomial time.*

It follows from Theorem 5.5 that there is no concept of a "minimal" partial plan. In the context of POPs, a *minimal relaxation* of a POP cannot be further relaxed while remaining valid, and a *mimimum relaxation*

has the fewest ordering constraints of all relaxations. This distinction does not apply to partial plans. Let $P = \langle O, \phi \rangle$ be a valid partial plan. From Definitions 5.5 and 5.8, any relaxation of P must be valid, and therefore sound, and so must be of the form $Q = \langle O, \phi' \rangle$ where $\phi \models \phi' \models \text{MTC}(O)$. From Theorem 5.5, the partial plan $R = \langle O, \text{MTC}(O) \rangle$ is a minimum relaxation of P, and as $\phi' \models \text{MTC}(O)$, it is also a (minimum) relxation of Q. Thus, the notion of a "minimal" partial plan does not apply, as any non-minimum relaxation can always be further relaxed. Alternatively, any "minimal" partial plan, that is, one that cannot be relaxed any further, must also be a minimum.

5.3.2 Tractable Partial Plans

As flexibility cannot, in practice, be provided by an intractable partial plan, optimality criteria should ideally compare both the strength and treewidth of the plan's constraint formulae. Thus, the notion of a *k-treewidth relaxation* is introduced. If partial plan Q is a relaxation of partial plan P, then Q is a *minimal k-treewidth relaxation* of P if it cannot be relaxed any further without its treewidth exceeding k, and is a *minimum k-treewidth relaxation* of P if, amongst all possible relaxations of P, it admits the most models while keeping its treewidth bounded by k:

Definition 5.10. *Let P and Q be partial plans, let integer $k > 0$ and let $\#P$ denote the number of instantiations of partial plan P. Then:*

- *Q is a* **k-treewidth relaxation** *of P* iff *Q is a relaxation of P and* $\mathsf{treewidth}(Q) \leq k$,
- *Q is a* **proper k-treewidth relaxation** *of P* iff *Q is a proper relaxation of P and* $\mathsf{treewidth}(Q) \leq k$,
- *Q is a* **minimal k-treewidth relaxation** *of P* iff *it is a k-treewidth relaxation of P and there is no R* s.t. *R is a proper k-treewidth relaxation of Q, and*
- *Q is a* **minimum k-treewidth relaxation** *of P* iff *it is a k-treewidth relaxation of P and there is no R such that R is a k-treewidth relaxation of P and $\#P < \#R$.*

While the complexity of finding minimum and minimal k-treewidth relaxations remains open, deciding whether a partial plan has a proper k-treewidth relaxation is W[1]-hard:

Theorem 5.7. Proper k-Treewidth Relaxation. *For any partial plan P and integer $k > 0$, deciding the existence of a proper k-treewidth relaxation of P is* W[1]*-hard when parameterised with k.*

5.4 Restricted Cases

The previous sections demonstrated that while islands of tractability exist in the space of partial plans, determining whether a partial plan has a tractable relaxation is an intractable problem (Theorem 5.7). To address

this shortcoming, this section will introduce the idea of a *causal link plan* (CLP), a specialised partial plan that expresses constraints as a set of allowable causal links. Constraints represented in this way are less expressive than constraint formulae. However, the benefit is that, when limited to plans of this form, *the problem of finding a minimally constrained, tractable plan is fixed parameter tractable.* A practical fpt-algorithm for finding such plans, MkTR, is introduced.

5.4.1 Causal Link Plans

Unlike partial plans, in which the allowable variable bindings and orderings are explicitly defined with a constraint formula, causal link plans instead place high-level constraints on their instantiations' *causal structures.* In the context of classical plans and POPs, a casual structure is an implicit set of unthreatened *causal links* between producers and consumers:

Definition 2.5. *A* **causal link** *is a 4-tuple* $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle$ *such that* $\mathsf{prods}(o_p, q(\vec{u}))$ *and* $\mathsf{cons}(o_c, q(\vec{s}))$.

A consumer in a classical plan is *supported* by a causal link *iff* it is preceded by, and codesignated with, the associated producer, and the link is *unthreatened iff* any threats to this link are either non-codesignated with, or not ordered between, the producer and consumer. The presence of unthreatened causal links is central to various notions of plan validity, such as POCL-validity (Definition 2.7) and the MTC (Definition 5.9).

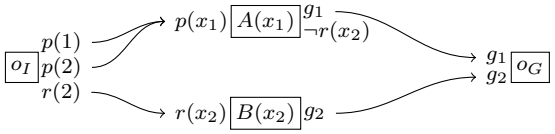
A CLP constrains its instantiations' causal structures by providing a *disjunctive causal structure*, that is, as a set of "allowable" causal links between consumers and possibly multiple producers, that implies the constraint that *each consumer be supported by at least one unthreatened causal link from the set.* More formally, a CLP comprises a set of operators O and a disjunctive causal structure L, with the condition that every consumer is linked to least one producer:

Definition 5.11. *A* **causal link plan** *(CLP) is a tuple* $P = \langle O, L\rangle$*, where* O *is a finite set of operators and* L *is a set of causal links such that:*

- *if* $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle \in L_O$ *then* $o_p, o_c \in O$*, and,*
- *for all* $o_c, q(\vec{s})$ s.t. $cons(o_c, q(\vec{s}))$ *and* $o_c \in O$*, there exists an* o_p *and* $q(\vec{u})$ *such that* $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle \in L$.

Figure 5.2a depicts a simple example of a CLP, with arrows indicating the contents of L. The causal links, that is, the plan's constraints, require that unthreatened causal links exist between A's precondition $p(x_1)$ and *either* $p(1)$ or $p(2)$, between B's precondition $r(x_2)$ and $r(2)$, and between the postconditions of both A and B and the goal state o_G.

A CLP's implicit constraints can be made explicit by transforming L into a constraint formula. The translation, denoted $F(P)$, encodes the MTC for partial plans as in Definition 5.9, but with the additional requirement that a consumer be supported by a causal link in L:



(a) CLP P_1 from a synthetic domain. Arrows indicate allowable causal links.

$$O = \{o_I,\ A(x_1),\ B(x_2),\ o_G\} \qquad \phi = ((x_1 = 1 \wedge o_I \prec A) \vee (x_1 = 2 \wedge o_I \prec A)) \wedge x_2 = 2 \wedge o_I \prec B \wedge (x_1 \neq x_2 \vee A \prec o_I \vee B \prec A) \wedge A \prec o_G \wedge B \prec o_G$$

(b) Partial plan $P_2 = \langle O, \phi \rangle$, an equivalent representation of P_1

Figure 5.2: Causal link plan (CLP) example

Definition 5.12. *For any CLP $P = \langle O, L \rangle$, $F(P)$ encodes O and L into a constraint formula as follows:*

$$F(P) \stackrel{\text{def}}{=} \bigwedge_{o_c, q(\vec{s}) : \mathsf{cons}(o_c, q(\vec{s}))} F'(o_c, q(\vec{s})),\ \textit{where}$$

$$F'(o_c, q(\vec{s})) \stackrel{\text{def}}{=} \bigvee_{\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle \in L} \vec{s} = \vec{u} \wedge o_p \prec o_c \wedge \bigwedge_{o_t, \vec{v} : \mathsf{thrtns}(o_t, q(\vec{v}))} \left(\vec{s} \neq \vec{v} \vee o_t \prec o_p \vee o_c \preceq o_t \vee \left[\bigvee_{o_w, \vec{r} : \mathsf{prods}(o_w, q(\vec{r}))} \vec{s} = \vec{r} \wedge o_t \prec o_w \prec o_c \right] \right).$$

For example, partial plan P_2 in Figure 5.2b is the result of using F to transform the causal structure in Figure 5.2a into a constraint formula.

Because the above encoding translates any CLP into a standard partial plan, any property of partial plans, such as satisfiability, soundness, validity or treewidth, can be applied to CLPs. If $P = \langle O, L \rangle$ is a CLP and $Q = \langle O, F(P) \rangle$ is a partial plan, then P is satisfiable, sound or valid *iff* Q is, respectively, and $\mathsf{treewidth}(P) = \mathsf{treewidth}(Q)$.

Optimising causal structures There are a number of aspects of CLPs and the F encoding that greatly simplify the plan relaxation process. The first is that the F encoding is *at least as strong as the MTC* (Definition 5.9). The MTC requires that each consumer in a CLP $P = \langle O, L \rangle$ be supported by a causal link which is either unthreatened, or re-established by a "white knight", while F requires that the link also be selected from L. Thus, for any CLP $P = \langle O, L \rangle$, $F(P) \models \mathrm{MTC}(O)$, and so, from Observation 5.5, a CLP is *always sound*.

The second key aspect of the F encoding is that if $P = \langle O, L \rangle$ and $Q = \langle O, L' \rangle$ are two CLPs such that $L \subseteq L'$, that is, P has a smaller set of causal links available, then $F(P) \models F(Q)$. Thus, Q is a relaxation of P, and if P is satisfiable, so is Q.

As a CLP is valid *iff* it is sound and valid, an observation that is central to the CLP relaxation process follows, that is, *adding a causal link into a valid CLP's causal structure will maintain its validity*:

Observation 5.2. *Let $P = \langle O, L\rangle$ and $Q = \langle O, L'\rangle$ be CLPs such that $L \subseteq L'$. Then, if P is valid so is Q.*

The F encoding also allows a comparison of the treewidths of CLPs. It follows from Definitions 5.12 and 5.7 that if $P = \langle O, L\rangle$ and $Q = \langle O, L'\rangle$ are two CLPs such that $L \subseteq L'$, then P's primal graph is a subgraph of Q's, meaning that P's treewidth is bounded by that of Q [16]. Thus, *expanding a CLP's causal structure cannot decrease its treewidth*:

Observation 5.3. *If $P = \langle O, L\rangle$ and $Q = \langle O, L'\rangle$ are two CLPs such that $L \subseteq L'$, then* $\mathsf{treewidth}(P) \leq \mathsf{treewidth}(Q)$.

5.4.2 Optimising Causal Link Plans

The optimality criteria for standard partial plans can be slightly modified and applied to CLPs. The relative flexibility of CLPs can be determined with a set-wise comparison of their constraints, and their relative tractability can be determined by comparing the treewidths of the formulae produced by F.

Thus, a *k-treewidth CS relaxation* of a CLP is an expansion of its disjunctive causal structure that keeps its treewidth below k, a *minimal k-treewidth CS relaxation* is a CLP with a causal structure that cannot be expanded any further without its treewidth exceeding k, and a *minimum k-treewidth CS relaxation* is the k-treewidth CS relaxation with the *largest* causal structure:

Definition 5.13. *Let $P = \langle O, L\rangle$ and $Q = \langle O, L'\rangle$ be two valid CLPs.*

- *Q is a* **k-treewidth CS relaxation** *of P* iff $\mathsf{treewidth}(Q) \leq k$ *and $L \subseteq L'$.*
- *Q is a* **proper k-treewidth CS relaxation** *of P* iff $\mathsf{treewidth}(Q) \leq k$ *and $L \subset L'$.*
- *Q is a* **minimal k-treewidth CS relaxation** *of P* iff *it is a k-treewidth CS relaxation of P and there is no CLP R such that R is a proper k-treewidth CS relaxation of Q.*
- *Q is a* **minimum k-treewidth CS relaxation** *of P* iff *it is a k-treewidth CS relaxation of P and there is no CLP $P = \langle O, L''\rangle$ such that R is a k-treewidth CS relaxation of Q and $|L''| > |L'|$.*

While the complexity of finding a minimum k-treewidth CS relaxation remains open, the MkTR algorithm in Figure 5.3 demonstrates that the problem of finding a minimal k-treewidth CS relaxation is in FPT:

Theorem 5.8. Minimal k-treewidth CS Relaxation. *Finding a minimal k-treewidth CS relaxation of a CLP is in* FPT *for parameter k.*

The MkTR algorithm takes as its input a valid CLP, $P = \langle O, L\rangle$. It first checks P's treewidth, and, consistent with Observation 5.3, returns failure if it already exceeds k. It then constructs A, a set containing every possible causal link between operators in O that do not already appear in

Input: Valid CLP $P = \langle O, L \rangle$, integer $k > 0$.
Result: A minimal k-treewidth CL relaxation of P.
1: **if** treewidth$(P) > k$ **then**
2: **return** $\bot$
3: **end if**
4: $A \leftarrow \{\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle : o_p, o_c \in O\} \setminus L$.
5: **for all** $l \in A$ **do**
6: **if** treewidth$(\langle O, L \cup \{l\}\rangle) \leq k$ **then**
7: $L \leftarrow L \cup \{l\}$
8: **end if**
9: **end for**
10: **return** P

Figure 5.3: MkTR: minimal k-treewidth CL relaxation

L. At each iteration of the loop, an element of A is added into L, and its treewidth is measured. From Observation 5.2, expanding L will not render the plan invalid, meaning that if the treewidth is less than k, the resulting CLP is a proper k-treewidth CS relaxation of P. Once all links have been tried, P is returned. The minimality of P follows from Observation 5.3, that is, any links that have been rejected cannot be added to P without causing its treewidth to exceed k. (For a complete proof see Appendix D).

5.5 Implementation

The MkTR algorithm has been implemented, and its ability to find flexible, tractable partial plans has been assessed over a suite of tests drawn from standard IPC domains. However, preliminary testing showed that MkTR, in the naive form presented in Figure 5.3, suffers from a number of deficiencies. Firstly, MkTR relaxes CLPs, meaning that classical plans must be first transformed into a suitable format. Secondly, the F encoding in Definition 5.12 tends to transform CLPs into partial plans with excessively high treewidths, meaning that very little relaxation is possible. Thirdly, as causal links are selected and tested in an essentially random order, MkTR frequently times out before finding a significantly more flexible relaxation. And finally, it was observed that (as in Chapter 4) the search space of possible causal structures is highly symmetrical. Thus, the implemented version of MkTR[2] optimises constraint formulae, uses a *relaxation policy* that defines which causal link is tested at each iteration, and uses symmetry breaking techniques to reduce the search space. This section will discuss these optimisations in more detail.

5.5.1 Relaxing Classical Plans

A classical plan must be transformed into a CLP before it can be relaxed with MkTR, that is, an equivalent causal structure must be found. As there may be multiple such structures, the implemented MkTR takes the same approach as the EOG deordering algorithm, that is, it uses a *validation*

[2] https://bitbucket.org/max_waters/phd-plan-flexibility-tools

structure (Definition 4.19). If $\vec{a}$ is a classical plan, then validation structure $V_{\vec{a}}$ is a set of unthreatened causal links that associates each consumer in $\vec{a}$ with exactly one supporting producer. If a consumer has multiple possible producers, then the earliest is selected.

The implemented MkTR first converts classical plan $\vec{a} = \vec{o}\theta$ into a CLP $P = \langle O, L\rangle$ *s.t.* $O = \{o : o \in \vec{o}\}$ and $L = V_{\vec{a}}$.

5.5.2 Optimisation

Two optimisations have been introduced to reduce the treewidths of the formulae produced by the F encoding (Definition 5.12). As a graph containing a clique of n vertices will have a treewidth of at least $n-1$, and a formula's primal graph connects variables that appear in the same clause, a partial plan's treewidth has a lower bound of $n-1$, where n is the most variables appearing in the same clause. Thus, the optimisations aim to reduce the number of variables appearing in the encoding.

Firstly, all constraint formulae are simplified with a modified AC-3 algorithm that propagates domains to constraints and *vice versa*, replaces variables with singleton domains with constants, and merges any variables that are necessarily codesignated into "meta-variables".

Secondly, the F encoding in Definition 5.12, which transforms a CLP's disjunctive causal structure into a constraint formula, has been replaced with one that produces formulae that are simpler at the expense of having fewer models. The input (classical) plan to MkTR, $\vec{a}$, is used as a heuristic to ensure satisfiability:

Definition 5.14. *Let $P = \langle O, L\rangle$ be a CLP, $\vec{a} = \vec{o}\theta$ be an instantiation of P, and $\prec'$ be the total order over $\vec{o}$. Then, $G(P, \vec{a})$ encodes a constraint formula as follows:*

$$G(P,\vec{a}) \overset{\text{def}}{=} \bigwedge_{o_c,q(\vec{s}):\mathsf{cons}(o_c,q(\vec{s}))} G'(o_c, q(\vec{s})),\ \textit{where}$$

$$G'(o_c, q(\vec{s})) \overset{\text{def}}{=} \bigvee_{\langle o_p,q(\vec{u}),o_c,q(\vec{s})\rangle\in L} \vec{u} = \vec{s} \wedge o_p \prec o_c \wedge$$

$$\bigwedge_{\substack{o_t,\vec{v}:\\ \mathsf{thrtns}(o_t,q(\vec{v}))}} \begin{cases} \vec{s} \neq \vec{v} & \textit{if } \theta(\vec{s}) \neq \theta(\vec{v}) \textit{ or } \theta(\vec{u}) \neq \theta(\vec{v}) \\ o_t \prec o_p & \textit{if } \theta(\vec{s}) = \theta(\vec{v}) \textit{ and } o_t \prec' o_p \\ o_c \prec o_t & \textit{if } \theta(\vec{s}) = \theta(\vec{v}),\ o_t \not\prec' o_p \textit{ and } o_c \prec' o_t \\ o_t \prec o_p \vee o_c \prec o_t & \textit{otherwise} \end{cases}$$

While F and G both encode the requirement that every consumer in the plan be supported by an unthreatened causal link, they differ in how causal links are protected from threats. The final line of F allows four ways for a causal link to be protected from a threat: non-codesignation, ordering the threat before the producer, or after the consumer, or re-establishing the link with a "white knight". However, the final line of G is stronger and simpler than this, it removes the 'white knight" option and requires that threat protection constraints must be, where possible, *consistent with the original input plan.* Thus, if the threat was not codesignated with either the producer or consumer in $\vec{a}$, then it must remain so, otherwise the original

ordering must be respected. This ensures that the partial plan produced by G still has at least one instantiation – $\vec{a}$ – and is thus still valid. The final case occurs when $o_p \prec' o_t \prec' o_c$, and all were codesignated, meaning that the causal link could not have been in $\vec{a}$. Thus, G arbitrarily requires the threat to be resolved through ordering constraints.

A key property of this encoding is that if a classical plan $\vec{a}$ is transformed into a CLP P as in Section 5.5.1, *then the partial plan produced by* $G(P, \vec{a})$ *is equivalent to the POP produced by* $\text{EOG}(\vec{a})$:

Observation 5.4. *Let* $\vec{a} = \vec{o}\theta$ *be a classical plan with validation structure* $V_{\vec{a}}$, $P = \langle O, L \rangle$ *be a CLP* s.t. $O = \{o : o \in \vec{o}\}$ *and* $L = V_{\vec{a}}$, *and* $Q = \langle O, G(P, \vec{a}) \rangle$ *be the encoding of* P *into a partial plan. Then, classical plan* $\vec{a}'$ *is an instantiation of* Q iff *it is a linearisation of* $\text{EOG}(\vec{a})$.

This observation follows directly from the definitions of EOG and G. As P and Q were constructed from the same validation structure ($L = V_{\vec{a}}$) that links every consumer to exactly one producer, all instantiations of P will have the same bindings as $\vec{a}$ and $\text{EOG}(\vec{a})$. And, when all variables are ground, G and EOG are identical. If causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle \in L$, then both EOG and G require that $o_p \prec o_c$. If postcondition $\neg q(\vec{v})$ of o_t is a threat to this link and $\theta(\vec{v}) \neq \theta(\vec{s})$, then EOG can ignore the threat, and G will place the constraint $\vec{v} \neq \vec{s}$, which simplifies to *true*. If $\theta(vecv) = \theta(\vec{s})$, then both EOG and G will resolve the threat by requiring that either $o_t \prec o_p$ or $o_c \prec o_t$, depending on the ordering in $\vec{a}$.

As the implemented MkTR transforms the input classical plan into a CLP, it follows from this observation that *the output of* MkTR *can never be less flexible than the POP found by* EOG.

5.5.3 Relaxation Policies

Because the success of MkTR will be influenced by which causal link is selected at each step, two selection policies have been tested.

5.5.3.1 Minimise Threats Policy

The *Minimise Threats* policy (MT) is based on two observations. It can be seen in Definition 5.14 that the more threats there are to a causal link, the larger the constraint formula must be that defines when the link is unthreatened, that is, the more likely it is that adding the link to the plan will increase its treewidth. Furthermore, the more threats there are to a link, the less likely it is that those threat protection constraints will be satisfiable, that is, the less likely it is that adding the causal link will result in an increase in possible instantiations.

Therefore, at each step in the MkTR loop, MT selects the causal link with the fewest threats, that is, it selects the link $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle$ for which there are the fewest producers o_t and $q(\vec{v})$ *s.t.* $\mathsf{thrtns}(o_t, q(\vec{v}))$.

5.5.3.2 Relax Producers Policy

Because relaxing a producer's variable bindings can in turn relax those of any dependent consumers, the *Relax Producers* policy (RP) *relaxes the*

bindings of operators with the most causally dependent consumers. The policy considers two ways to do this. The first way to increase the possible bindings for operator o is to simply add more producer options for o's preconditions. However, if o's postconditions threaten other links in the plan, then the non-codesignation constraints required to protect such links from threats may rule out some bindings for o. In such cases, adding more producer options for o's preconditions may have no effect. Thus, a second way is to add more producer options to preconditions of other operators that are threatened by o's postconditions.

This translates into two measures. If o is an operator and $\vec{a}$ is the input plan, then $n_c(o, \vec{a})$ denotes the number of actions in $\vec{a}$ with preconditions that are causally dependent on a postcondition of o. And $n_t(o, \vec{a})$ is equal to the highest $n_c(o_t, \vec{a})$ for any o_t that threatens any causal link to any precondition of o. RP selects the causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle$ that maximises $\max(n_c(o_c, \vec{a}), n_t(o_c, \vec{a}))$.

5.5.4 Symmetry Breaking

A search or optimisation problem is said to exhibit *symmetry* (Section 2.3) when states or branches in its search space are identical up to an irrelevant permutation of variables and/or constants. As these permutations preserve solution validity and optimality, symmetries can be exploited by limiting search to just one of each equivalent region.

Chapter 4 introduced the notion of *causal structure symmetry* (Section 4.4.2), which occurs when a plan's operators contain combinations of pre/postconditions that are *functionally equivalent* in that any associated causal links can be swapped without affecting the plan's validity or optimality. The symmetry breaking technique presented here is motivated by the observation that CLPs with symmetrical causal structures will have the same treewidth. Thus, if MkTR rejects a particular causal structure due to its treewidth exceeding the input parameter, any symmetrically equivalent causal structures can also be ruled out without the expensive optimisation and treewidth calculation processes.

Symmetries are detected and represented with the techniques introduced in Chapter 4. A (disjunctive) causal structure L can be equivalently represented as a Boolean (0/1) matrix $\mathbf{C}_L$ that associates producers and consumers with rows and columns, respectively, with $\mathbf{C}[i][j] = 1$ indicating that producer i and consumer j are causally linked (Definition 4.16). As a causal structure symmetry permutes producers and consumers, it can be represented as a permutation of entire rows and/or columns of variables in $\mathbf{C}_L$ (Definition 4.15). Symmetries are detected from the input plan's *plan description graph* (PDG, Definition 4.13), an undirected coloured graph with automorphisms corresponding to the plan's symmetries. The automorphisms' generating set, Σ, is computed, and from this, $\bar{\Sigma}$, a set of causal structure symmetries over $\mathbf{C}_L$, is constructed (Definitions 4.14 and 4.17).

Unlike previous chapters, in which symmetries are broken *statically*, that is, by expanding the problem description with additional constraints that rule out non-canonical solutions, there are here broken *dynamically*, that is, by directing MkTR to *explore no more than one causal structure from each symmetry class.*

Input: Causal structure $\mathbf{C}$, set of causal structure symmetries $\bar{\Sigma}$
Result: A symmetrical form of $\mathbf{C}$ that satisfies Multi Lex.

```
1:  loop
2:    c ← true
3:    for all σ ∈ Σ̄ do
4:      if ¬MultiLex(C, σ) then
5:        C = σ(C)
6:        c ← false
7:      end if
8:    end for
9:    if c then
10:     return C
11:   end if
12: end loop
```

Figure 5.4: ApxC: approximate canonicalisation with Multi Lex.

As symmetrical causal structures have identical *canonical forms* (Definition 3.9), this can be achieved by caching the canonical form of any causal structure found to have a treewidth exceeding k. At each iteration of the MkTR loop, the current disjunctive causal structure's canonical form is found, and its treewidth is only computed if that form is not in the cache.

As finding a canonical form is NP-hard [83], an heuristic approach to canonicalisation is taken, similar to that of Pochter et al. [100], that trades efficiency for search space reduction. The ApxC algorithm in Figure 5.4 uses the symmetry breaking constraint Multi Lex (as introduced in Definition 3.12 of Chapter 3) to compute an approximate canonical form. It takes as input a causal structure in matrix form, $\mathbf{C}$, and a set of causal structure symmetries, $\bar{\Sigma}$. Elements of $\bar{\Sigma}$ are applied to $\mathbf{C}$ until a symmetric equivalent is found that satisfies Multi Lex. As Multi Lex is a partial symmetry breaking constraint, there is no guarantee that ApxC will find the lexicographic least element of the symmetry class, or that all symmetrical causal structures will be mapped to the same approximation. Thus, not all symmetries will be broken. However, the results in Chapter 3 show that Multi Lex, in practice, rules out practically all non-canonical solutions.

5.6 Experimental Evaluation

The notions of optimality discussed in Section 5.3 reflect the central challenge of partial plans, which is to find a plan that is flexible while still being tractable enough to be useful at execution time. But, this theoretical examination cannot reveal whether, in practice, the strict tractability requirements are so limiting that they render the plans unable to provide any flexibility. Section 5.4 demonstrated that finding a minimally constrained, tractable CLP is feasible. However, the question remains whether MkTR can generate (or indeed, whether there even exist) partial plans that *(i)* provide more flexibility than the POPs produced by simpler plan de/reordering techniques, and *(ii)* have an instantiation cost within the bounds of what is suitable for real-time decision-making.

To address this question, MkTR has been assessed over all previous IPC STRIPS domains. As a baseline, MkTR was compared with the *explanation-based order generalisation* deordering technique of Kambhampati and Kedar [71] (EOG, Definitions 4.19 and 4.20). As discussed in Section 2.2.2.2, EOG is a greedy POP deordering algorithm that uses the input plan's causal structure as a deordering heuristic. Despite this lack of optimality guarantee, Chapter 4 confirmed the results of Muise et al. [91] that in all cases, EOG finds a minimal deorder (Definition 2.18) of the input plan.

Results show that in all test cases, MkTR produces a partial plan of quadratic "complexity" that encapsulates all linearisations of the POP found by EOG. While certain planning domains resist any kind of relaxation, on average MkTR finds a partial plan with 163.2% more instantiations than EOG, and increasing the maximum treewidth from two to five increases this to 302.3% with little change in instantiation time.

5.6.1 Experimental Setup

As in Chapter 4, test cases (i.e., input plans) were generated by finding plans for all first-order IPC STRIPS planning instances. To ensure a variety of plans, three planners of distinct "types" were used: the novelty-driven best-first search planner Dual-BFWS [81], the heuristic forward-search planner LAMA [108] and the SAT planner Madagascar [112].

Each plan was relaxed with MkTR under eight different configurations. The *Minimise Threats* and *Relax Producers* policies were tested, both with and without symmetry breaking (denoted MT, RP, MT_S and RP_S, respectively), and maximum treewidths of two and five. Treewidths were calculated with treewidth-exact [3] and libtw [129], and for the MT_S and RP_S policies, PDG automorphisms were found with NAUTY [85]. As a baseline, each plan was also deordered with EOG (Definition 4.20). The flexibility of the partial plans generated by MkTR and the POPs generated by EOG was measured by counting their instantiations (which, in the case of EOG, is their linearisations). The GANAK [117] model counter was used.

All of the planning, plan optimisation and model counting tasks above were given resource limits of 8GB RAM and 30m at 3.2GHz.

5.6.2 Results

Results are summarised in Tables 5.1–5.5. Tables 5.1, 5.2 and 5.5 compare the flexibility of the plans found by EOG with those found by MkTR. The $\#_{\text{EOG}}$ columns indicates the average number of instantiations (i.e., linearisations) of the POPs found by EOG, and Δ_{EOG} indicates the average percentage improvement found by each MkTR configuration. As the instantiation counts can vary exponentially, averages are computed from the *geometric mean*, which, for a set of positive real-valued numbers $\{x_1, \ldots, x_n\}$, is equal to $\sqrt[n]{x_1 \times \cdots \times x_n}$. This better reflects the nature of the data, and prevents a few extremely large values skewing the central tendency of the model counts.

[3] http://github.com/TCS-Meiji/treewidth-exact

			Δ_{EOG}			
Domain	n	$\#_{\text{EOG}}$	MT	MT_{S}	RP	RP_{S}
agricola	17	1	$2.3 \cdot 10^{20}$%	$2.3 \cdot 10^{20}$%	$2.4 \cdot 10^{20}$%	$2.4 \cdot 10^{20}$%
airport	22	$2.85 \cdot 10^{6}$	0%	0%	0%	0%
barman	62	$4.99 \cdot 10^{8}$	8.1%	6.4%	5.1%	5.1%
childsnack	0					
cybersec	62	56,389.96	81.6%	84.1%	80.1%	77.8%
data-network	0					
depot	24	$4.17 \cdot 10^{8}$	44.2%	44.2%	30.1%	31%
driverlog	37	$4.31 \cdot 10^{5}$	54.5%	28.1%	95.6%	75.6%
elevators	39	$1.32 \cdot 10^{6}$	76.4%	77.2%	24.4%‡	18.3%
floortile	2	$3.03 \cdot 10^{17}$				
freecell	45	792.13	0%	0%		
ged	91	115.12	18.3%	18.3%	20.1%	20.1%
gripper	60	52,336.19		4.2%		
hiking	46	3,738.84	55.4%	73.8%	2,088.9%‡	2,089.2%
logistics98	6	$9.89 \cdot 10^{7}$				
mprime	71	25.46	18.2%	13.3%	33.4%	25%
mystery	42	19.46	56.6%	47.4%	27.5%‡	24.9%
nomystery	36	255.85	0%	0%	0%	0%
org-synth	3	3.27				
org-synth-split	27	1	285.8%	281.7%	280.2%	270.6%
parcprinter	15	$3.19 \cdot 10^{5}$	14.8%	14.7%	12.5%	12.5%
parking	80	36.02			0%	0%
pipesworld	48	2,399.91	65%	61.9%	103.3%	88.8%
psr-small	140	52.97	45.5%	45.5%	46.3%	46.3%
rovers	17	$1.8 \cdot 10^{5}$	270.3%	294.5%	123.3%‡	113.2%
satellite	32	$3.4 \cdot 10^{6}$	207.4%	200.8%	114%	109.5%
scanalyzer	62	1,061.55		7.3%	7.1%	
tetris	18	$1.21 \cdot 10^{7}$			0%	0%
thoughtful	4	$3.04 \cdot 10^{5}$	0%	0%	0%	0%
tpp	15	7,646.35				
transport	45	$4.56 \cdot 10^{7}$	75.2%	69.9%		8%
visitall	41	1	14.3%	14.3%	0%‡	0%
woodworking	19	$6.93 \cdot 10^{6}$	1,035.4%	1,050.8%	528.8%‡	687.4%
zenotravel	30	23,976.53	60.6%	65.1%	19.2%‡	22.9%
ALL	1,258	4,799.14	151.5%	150.1%	163.2%	161.1%

Table 5.1: Comparison of MkTR with treewidth of two and EOG. For each domain, $\#_{\text{EOG}}$ is the mean number of instantiations of the POP produced by EOG, and Δ_{EOG} is the mean % increase in instantiations found by MkTR with each policy. All Δ_{EOG} values are computed from the n plans for which instantiation counts are available for all five approaches, and are statistically significant with $p < 0.05$ as calculated from a single-tailed, paired t-test. Empty cells indicate no data, or no significant difference.

To ensure a meaningful comparison between EOG and MkTR, and between the different MkTR configurations, Δ_{EOG} is computed from plans for which a model count is available for every plan relaxation technique. Further, Δ_{EOG} is only reported if there is a statistically significant difference between EOG and MkTR.[4] For most domains and planners, such a difference could be established between EOG and MkTR. However, due to the high variance in model counts and small sample sizes, it was often impossible to establish a statistically significant difference between the different MkTR configurations. The cases when such a comparison is significant are marked. The symbols † and ‡ indicate a significant difference between MT and RP, and between a policy and its symmetry breaking variant (i.e., between MT and MT_{S} or RP and RP_{S}), respectively.

[4] As determined by a single-tailed, paired t-test with $p < 0.05$

			Δ_{EOG}			
Domain	n	$\#_{\text{EOG}}$	MT	MT_{S}	RP	RP_{S}
agricola	17	1	$2.3 \cdot 10^{20}\%$	$2.3 \cdot 10^{20}\%$	$2.4 \cdot 10^{20}\%$	$2.4 \cdot 10^{20}\%$
airport	22	$2.85 \cdot 10^{6}$	0%	0%	0%	0%
barman	62	$4.99 \cdot 10^{8}$	9.4%	8.1%	24.7%	24.7%
childsnack	0					
cybersec	62	56,389.96	1,661.3%	1,633.1%	2,222.9%	1,540.7%†
data-network	0					
depot	24	$4.17 \cdot 10^{8}$	55.6%	61.8%	42.8%	48.3%
driverlog	37	$4.31 \cdot 10^{5}$	133.6%	136.3%	150.5%	191.2%
elevators	39	$1.32 \cdot 10^{6}$	83.8%	83.8%	28.9%‡	33.8%
floortile	2	$3.03 \cdot 10^{17}$				
freecell	45	792.13	11.8%	13.1%	7.4%	7.6%
ged	91	115.12	52.8%	55.4%†	64.2%‡	64.8%
gripper	60	52,336.19			37.6%‡	36.6%
hiking	46	3,738.84	87.1%	93.2%	2,432.3%‡	2,221.3%
logistics98	6	$9.89 \cdot 10^{7}$				
mprime	71	25.46	33.5%	30.6%	44.8%	46%
mystery	42	19.46	66.6%	74%	51.6%	54.3%
nomystery	36	255.85	0%	0%		
org-synth	3	3.27				
org-synth-split	27	1	366.9%	366.9%	321.4%	300.3%
parcprinter	15	$3.19 \cdot 10^{5}$	19.1%	14.6%	12.5%	12.5%
parking	80	36.02	8%	8%	0%‡	0%
pipesworld	48	2,399.91	80.8%	73.2%	121.8%	111.6%
psr-small	140	52.97	396.4%	392.5%	534.9%‡	534.9%
rovers	17	$1.8 \cdot 10^{5}$	749.8%	744.6%	616.8%	576.1%
satellite	32	$3.4 \cdot 10^{6}$	365%	396.1%	192.8%	220.9%
scanalyzer	62	1,061.55	12.6%	13.1%	8.4%	
tetris	18	$1.21 \cdot 10^{7}$			0%	0%
thoughtful	4	$3.04 \cdot 10^{5}$	0%	0%	0%	0%
tpp	15	7,646.35	110.9%	108%	288.2%	310.9%
transport	45	$4.56 \cdot 10^{7}$	75.5%	74.6%	8%‡	12.3%
visitall	41	1	16.3%	14.3%	0%‡	0%
woodworking	19	$6.93 \cdot 10^{6}$	2,181.7%	2,177.4%	1,087.7%‡	1,070.1%
zenotravel	30	23,976.53	111.4%	105.3%	83.7%	113.4%
ALL	1,258	4,799.14	261.6%	261.4%	302.3%	298.3%

Table 5.2: Comparison of MkTR with treewidth of five and EOG. For each domain, $\#_{\text{EOG}}$ is the mean number of instantiations of the POP produced by EOG, and Δ_{EOG} is the mean % increase in instantiations found by MkTR with each policy. All Δ_{EOG} values are computed from the n plans for which instantiation counts are available for all five approaches, and are statistically significant with $p < 0.05$ as calculated from a single-tailed, paired t-test. Empty cells indicate no data, or no significant difference.

Tables 5.3 and 5.4 compare the coverage rates and execution times for EOG and MkTR. The *coverage* C is the proportion of plans for which EOG found a solution. As MkTR defaults to EOG (Observation 5.4), this is also the proportion for which all MkTR configurations found a solution. For each configuration, the *minimal coverage* C_M is the proportion of plans for which a minimal k-treewidth CS relaxation was found. Finally, T is the average execution time in minutes.

Domains in which no plan could be improved by either EOG or MkTR with any configuration have been excluded from the tables (i.e., for all *pegsol*, *snake*, *sokoban*, *termes* instances, $\#_{\text{EOG}} = 1$ and all $\Delta_{\text{EOG}} = 0\%$).

			C_M				T			
Domain	n	C	MT	MT_S	RP	RP_S	MT	MT_S	RP	RP_S
agricola	17	100%	0%	0%	0%	0%	30	30	30	30
airport	115	100%	1.7%	1.7%	1.7%	1.7%	28.54	28.54	28.54	28.55
barman	74	100%	0%	0%	0%	0%	30	30	30	30
childsnack	90	100%	36.7%	34.4%	25.6%	25.6%	24.03	24.23	25.62	25.38
cybersec	90	100%	95.6%	95.6%	95.6%	95.6%	2.99	3.05†	3.06	2.95†
data-network	22	100%	0%	0%	0%	0%	30	30	30	30
depot	63	100%	7.9%	7.9%	7.9%	7.9%	27.99	27.97	28	27.97
driverlog	59	100%	30.5%	30.5%	28.8%	28.8%	23.72	23.68	24.48	24.58
elevators	101	100%	22.8%	21.8%	17.8%	16.8%	25.57	25.74†	26.68	26.67
floortile	65	100%	0%	0%	0%	0%	30	30	30	30
freecell	147	100%	7.5%	8.2%	7.5%	7.5%	28.4	28.34†	28.44	28.45
ged	91	100%	60.4%	60.4%	60.4%	60.4%	13.3	13.32	13.45	13.53
gripper	60	100%	15%	15%	15%	13.3%	26.38	26.39	26.63	26.67
hiking	82	100%	17.1%	19.5%	18.3%	17.1%	25.94	25.93	26.28	26.34
logistics98	101	100%	3%	3%	3%	3%	28.97	28.95	29.03	29
mprime	97	100%	40.2%	39.2%	39.2%	40.2%	20.7	21.08	20.83	20.65†
mystery	51	100%	66.7%	66.7%	62.7%	64.7%	14.11	13.98	14.78	14.78
nomystery	95	100%	0%	0%	0%	0%	30	30	30	30
org-synth	3	100%	0%	0%	0%	0%	30	30	30	30
org-synth-spl	27	100%	29.6%	29.6%	29.6%	29.6%	22.72	22.78	23.54	23.45
parcprinter	118	100%	7.6%	7.6%	7.6%	7.6%	28.35	28.36	28.62	28.59
parking	80	100%	0%	0%	0%	0%	30	30	30	30
pipesworld	89	100%	12.4%	10.1%	10.1%	12.4%	28.39	28.4	28.56	28.58
psr-small	140	100%	100%	100%	100%	100%	0.26	0.35	0.25	0.25
rovers	120	100%	21.7%	20.8%	20%	20%	23.93	23.9	24.07	24.09
satellite	100	100%	16%	16%	16%	16%	25.17	25.19	25.52	25.49
scanalyzer	104	100%	6.7%	6.7%	6.7%	6.7%	28.7	28.75	28.68	28.72
tetris	79	100%	5.1%	5.1%	5.1%	5.1%	28.81	28.76	28.88	28.9
thoughtful	40	100%	0%	0%	0%	0%	30	30	30	30
tpp	89	100%	16.9%	15.7%	15.7%	15.7%	24.34	24.43†	24.55	24.4†
transport	98	100%	12.2%	12.2%	11.2%	10.2%	26.66	26.73	27.54	27.38†
visitall	92	79.3%	0%	0%	0%	0%	30	30	30	30
woodworking	120	100%	28.3%	28.3%	25%	23.3%	24.39	24.55†	25.06	24.95
zenotravel	57	94.7%	38.6%	36.8%	38.6%	40.4%	20.5	20.55	20.98	20.68†
ALL	2,776	99.2%	22.9%	22.7%	21.9%	21.9%	23.53	23.58†	23.82	23.78†

Table 5.3: Coverage rates and execution times for MkTR with treewidth two. For each domain and planner, C indicates the proportion of plans for which EOG, and thus MkTR, found a solution. For each policy, C_M indicates the proportion of plans for which MkTR found a minimal k-treewidth CS relaxation, and T indicates the execution time in minutes.

5.6.2.1 Flexibility provided by MkTR

Tables 5.1 and 5.2 show that MkTR *can find significantly more flexible partial plans than* EOG: with a maximum treewidth of two, the partial plans found by MkTR have, on average, up to 163.2% more instantiations than those found by EOG, depending on policy. Interestingly, there are three domains (*agricola*, *organic-synthesis-split* and *visitall*) in which MkTR *can relax plans that* EOG *cannot deorder at all*. Of these three domains, the most striking improvement is in *agricola*, where MkTR finds partial plans with, on average, 2.4×10^{20} instantiations while EOG cannot improve plan flexibility at all (and indeed, neither can any of the generalised de/reordering approaches introduced in Chapter 4). Other significant results are in *hiking*, where the RP policy yields a 2088.9% increase in instantiations, and *woodworking*, where MT_S yields a 1050.8% increase.

However, while MkTR will, by definition, always find a plan at least

			C_M				T			
Domain	n	C	MT	MT_S	RP	RP_S	MT	MT_S	RP	RP_S
agricola	17	100%	0%	0%	0%	0%	30	30	30	30
airport	115	100%	1.7%	1.7%	1.7%	1.7%	28.54	28.55	28.53	28.53
barman	74	100%	0%	0%	0%	0%	30	30	30	30
childsnack	90	100%	34.4%	33.3%	24.4%	26.7%	24.78	25.07†	25.87	25.76
cybersec	90	100%	95.6%	95.6%	95.6%	95.6%	3.05	3.09	3.09	3.08
data-network	22	100%	0%	0%	0%	0%	30	30	30	30
depot	63	100%	7.9%	7.9%	7.9%	7.9%	27.97	27.98	27.98	28
driverlog	59	100%	30.5%	32.2%	28.8%	27.1%	23.89	23.7†	24.59	24.49
elevators	101	100%	20.8%	21.8%	17.8%	16.8%	25.99	25.6†	26.87	27.02
floortile	65	100%	0%	0%	0%	0%	30	30	30	30
freecell	147	100%	7.5%	8.2%	7.5%	8.2%	28.45	28.4†	28.47	28.45
ged	91	100%	60.4%	60.4%	60.4%	60.4%	13.4	13.34	13.4	13.42
gripper	60	100%	18.3%	15%	15%	13.3%	26.25	26.37	26.63	26.69†
hiking	82	100%	17.1%	15.9%	15.9%	15.9%	26.59	26.57	27.17	27.1
logistics98	101	100%	3%	3%	3%	3%	29.14	29.09	29.19	29.15
mprime	97	100%	40.2%	40.2%	38.1%	38.1%	20.74	20.8	21.1	21.06
mystery	51	100%	66.7%	66.7%	62.7%	64.7%	14.09	13.97	14.58	14.57
nomystery	95	100%	0%	0%	0%	0%	30	30	30	30
org-synth	3	100%	0%	0%	0%	0%	30	30	30	30
org-synth-spl	27	100%	29.6%	29.6%	29.6%	29.6%	22.75	22.73	23.41	23.35
parcprinter	118	100%	7.6%	7.6%	7.6%	7.6%	28.38	28.47†	28.56	28.57
parking	80	100%	0%	0%	0%	0%	30	30	30	30
pipesworld	89	100%	9%	10.1%	9%	7.9%	28.38	28.4	28.57	28.56
psr-small	140	100%	100%	100%	100%	100%	0.26	0.28†	0.25	0.26
rovers	120	100%	21.7%	20.8%	19.2%	19.2%	23.85	23.99†	24.18	24.2
satellite	100	100%	16%	16%	16%	16%	25.15	25.24†	25.52	25.5
scanalyzer	104	100%	3.8%	3.8%	3.8%	3.8%	29.24	29.22	29.19	29.18
tetris	79	100%	5.1%	5.1%	5.1%	5.1%	28.77	28.78	28.85	28.91
thoughtful	40	100%	0%	0%	0%	0%	30	30	30	30
tpp	89	100%	15.7%	15.7%	15.7%	15.7%	24.32	24.34	24.42	24.47
transport	98	100%	12.2%	12.2%	11.2%	10.2%	26.75	26.72	27.53	27.44
visitall	92	79.3%	0%	0%	0%	0%	30	30	30	30
woodworking	120	100%	30%	29.2%	25%	24.2%	24.41	24.46	25.04	25.12
zenotravel	57	94.7%	38.6%	38.6%	38.6%	38.6%	20.53	20.62	20.76	20.91
ALL	2,776	99.2%	22.7%	22.6%	21.6%	21.5%	23.63	23.64	23.88	23.89

Table 5.4: Coverage rates and execution times for MkTR with treewidth five. For each domain and planner, C indicates the proportion of plans for which EOG, and thus MkTR, found a solution. For each policy, C_M indicates the proportion of plans for which MkTR found a minimal k-treewidth CS relaxation, and T indicates the execution time in minutes.

as flexible as that found by EOG (Observation 5.4), in some domains (e.g., *airport*, *thoughtful*, *nomystery*), it does not find a significantly more flexible one, even with a treewidth of five.

This significant increase in flexibility comes at substantial computational cost. Tables 5.3 and 5.4 show that the average run time approaches 24 min, and indeed of the 22560 test cases, 67% ran for the full 30 min. MkTR has a run time of < 20 min in just four domains: *ged*, *cybersec*, *mystery* and *psr-small*. Significantly, however, this does not result in a reduction in flexibility. For example, in *cybersec*, RP with treewidth five provides a 2222.9% increase over EOG in 3.05 min, and in *mystery*, MT with treewidth two provides a 56.6% increase over EOG in 14.11 min.

5.6.2.2 Effect of Relaxation Policy

While Tables 5.1 and 5.2 suggest that overall, RP is more successful than MT, this difference is not statistically significant. There are, however, significant differences in twelve domains (marked with ‡), and these results suggest that some policies are suited to certain domains. For example, in *hiking*, RP is the most successful. With a treewidth of two and five, Δ_{EOG} is 2033.5pp[5] and 2345.2pp higher than MT's, respectively. In contrast, MT is the most effective in *woodworking*, where, with a treewidth of two and five, its Δ_{EOG} is 506.6pp and 1094pp greater than RP's, respectively.

Further, in *visitall*, MT with a treewidth of two finds 14.3% more instantiations than EOG while RP offers no improvement, and in *gripper* the opposite is true: RP with a treewidth of five finds 37.6% more instantiations than EOG, while MT provides no improvement.

5.6.2.3 Effect of Symmetry Breaking

Comparing MT and RP with MT_{S} and RP_{S} suggests that symmetry breaking results in little overall difference in plan flexibility or run time. Furthermore, in the domains for which symmetry breaking does influence results, its effects are not always beneficial, suggesting that the search space reduction it provides is sometimes not enough to counteract the additional overhead of detecting symmetries.

Tables 5.1 and 5.2 show that, when considered over all test cases, symmetry breaking makes no statistically significant difference to the flexibility of the final partial plans. The only domains in which it does produce a significant change (marked by †) are *cybersec* where, with a treewidth of five, RP_{S}'s Δ_{EOG} is 682.2pp *lower* than RP's, and *ged*, where, also with a treewidth of five, symmetry breaking results in a 2.6pp increase in Δ_{EOG} for MT. The only planner for which symmetry breaking significantly effects results is LAMA, where RP_{S} with a treewidth of five improves on EOG by 395.2% to RP's 408.3%.

Tables 5.3 and 5.4 show that symmetry breaking has no significant effect on the proportion of plans that MkTR solves optimally (C_M), and while in some domains it has an effect on the average run time (marked by †), in all cases the difference is under ten seconds.

A possible reason for this ineffectiveness is that MkTR stops on finding a local minimum, rather than backtracking and attempting to find a better solution. As a result, symmetry breaking can only prune single states, and not entire branches.

5.6.2.4 Effect of Treewidth

Tables 5.1 and 5.2 demonstrate that allowing MkTR to search for high-treewidth constraints can yield more flexible partial plans with more instantiations. Overall, increasing the maximum treewidth to five yields an increase in Δ_{EOG} of 110.1pp and 139.1pp for MT and RP, respectively.

Some domains show more extreme differences. In *cybersec*, increasing the treewidth results in a 1579.7pp and 2142.8pp improvement in Δ_{EOG}

[5] A percentage point pp measures the difference between two percentages, e.g., 50% and 55% differ by 10% but 5pp.

				Δ_{EOG}			
Planner	n	$\#_P$	$\#_{\text{EOG}}$	MT	MT_S	RP	RP_S
Dual-BFWS	487		4,143.49	131.8%	128.8%	140.6%	134.8%
LAMA	542		1,916.01	211.6%	211.3%	261%	259.9%
Madagascar	229	1,324.78	57,628.6	79.9%	80%	50.7%‡	53.1%

(a) Maximum treewidth two.

				Δ_{EOG}			
Planner	n	$\#_P$	$\#_{\text{EOG}}$	MT	MT_S	RP	RP_S
Dual-BFWS	487		4,143.49	238.8%	235.7%	268.1%	265.5%
LAMA	542		1,916.01	317.8%	322.6%	408.3%	395.2%†
Madagascar	229	1,324.78	57,628.6	195%	192%	179.4%	185.6%

(b) Maximum treewidth five.

Table 5.5: Comparison of MkTR and EOG by planner. For each planner, $\#_P$ is the number of linearisations of the Madagascar plans, $\#_{\text{EOG}}$ is the mean number of instantiations of the POP produced by EOG, and Δ_{EOG} is the mean % increase in instantiations found by MkTR with each policy. All Δ_{EOG} values are computed from the n plans for which instantiation counts are available for all five approaches, and are statistically significant with $p < 0.05$ as calculated from a single-tailed, paired t-test.

for MT and RP, respectively, and in *tpp*, neither MT nor RP provide a statistically significant benefit with a treewidth of two, but with five they improve on EOG by 110.9% and 288.2%, respectively.

While there is no domain in which increasing the treewidth to five resulted in an average decrease in instantiations, there are 166 individual plans for which using the same policy but increasing the treewidth to five resulted in a *less flexible* plan. There are two possible reasons for this: either the increased cost of treewidth calculations (from n^2 to n^5) slow MkTR, or the additional branches made available by increasing the treewidth in fact lead to worse solutions from which MkTR cannot recover as it always stops upon finding a local optimum.

Interestingly, the flexibility benefit of increasing the treewidth comes at little instantiation cost. With a treewidth of two, 82.8% of the (1200 randomly selected) final partial plans can be instantiated in under 250ms using the MapleSAT [79] SAT solver, and with a treewidth of five this reduces to 78.3% (although in both cases some outliers timed out after half an hour).

5.6.2.5 Effect of Planner

Table 5.5 shows that the source of the input plan influences the final flexibility. Plans generated by LAMA benefit the most from MkTR. With a treewidth of two, MkTR with the MT and RP policies produces 211.6% and 261% more instantiations than EOG, and with a treewidth of five, this increases to 317.8% and 408.3%, respectively.

As discussed in Section 4.5.2, Madagascar produces *parallel plans* (Definition 4.22), that is, generalised plans of the form $T = \langle S_1, \ldots, S_n \rangle$ where each step S_i is a set of actions that can be executed in any order (or indeed,

simultaneously), but must precede all actions in any subsequent steps. On average, EOG increases the number of linearisations of the parallel plans produced by Madagascar from 1324.78 to 57628.6. With a treewidth of two, MkTR with the MT and RP policies further improves on this by 79.9% and 50.7%, respectively, and a treewidth of five increases this to 195% and 179.4%, respectively. Thus, while LAMA plans benefit the most from MkTR, optimised Madagascar plans are the most flexible.

5.7 Discussion

The main contributions of this chapter were as follows:

- This chapter introduced, and formally defined, the notion of a *partial plan*, a generalised plan that specifies which action *types* must be executed, but rather than completely specify their orderings and variable bindings, provides a *constraint formula* that specifies the allowable combinations of orderings and bindings.

- Complexity analysis shows that finding an *instantiation* of a partial plan, that is, a classical plan that satisfies its constraints, is NP-complete. While this suggests that partial plans cannot provide execution-time flexibility, a parameterised complexity analysis shows the problem to be polynomial when parameterised with the *treewidth* of the constraint formula.

- A number of optimality criteria were introduced that compare the flexibility of partial plans based on the relative strengths of their constraint formulae. Key to providing execution-time flexibility is the idea of a *minimal k-treewidth relaxation*, that is, a partial plan that cannot be made more flexible without its treewidth exceeding k (i.e., without its runtime cost exceeding some predetermined complexity).

- While finding a minimal k-treewidth relaxation is an intractable problem, further analysis shows it to be in FPT when the search is limited to a particular class of plans in which constraints are represented as allowable causal links. A practical technique was introduced for finding minimal k-treewidth relaxations within this class: MkTR is a greedy, policy-driven FPT algorithm that relaxes a partial plan by iteratively expanding the set of allowable causal links while ensuring its treewidth stays below k.

- An empirical evaluation over all previous IPC STRIPS domains compared MkTR with the polynomial, non-optimal but effective EOG technique. Results show that overall, MkTR can find a quadratic-time instantiatable partial plan with 163.2% more instantiations than EOG, and setting a maximum treewidth of five increases this to 302.3% with little change in instantiation time. Interestingly, there are three domains in which MkTR can relax plans that cannot be improved by either EOG or any of the de/reordering techniques studied in Chapter 4. Extending MkTR with the Multi Lex symmetry

breaking technique (introduced in Chapter 3) in order to avoid symmetrically equivalent partial plans results in little change in flexibility or execution time.

This chapter studied, from both a theoretical and practical perspective, the problem of "relaxing" a plan into a flexible partial plan, a generalised plan that allows decisions regarding both ordering and domain objects to be postponed until execution time. Despite their worst-case intractability, an empirical evaluation shows that the MkTR algorithm can find a tractable partial plan with, on average, 163.2% more instantiations than the POPs found by the EOG baseline. While this comes at significant computational cost, MkTR cannot by definition produce a less flexible plan than the polynomial EOG, and indeed there are domains in which MkTR can relax plans that EOG cannot deorder at all. Thus, whether MkTR is preferable to EOG simply depends on whether off-line preprocessing time is available.

The remainder of this chapter will compare partial plan relaxation with some other approaches to achieving least commitment, in particular the *explanation-based generalisation* approach of Kambhampati and Kedar [71], and then present some possible areas of future work.

5.7.1 Comparison with Explanation Based Generalisation

As discussed in Section 2.2.2.2, Kambhampati and Kedar [71] also study the problem of relaxing a plan into a form that allows for execution-time domain object selection. The notion of *partially ordered, partially instantiated plans* (POPI) is introduced, and a technique for relaxing their variable binding and ordering constraints, *explanation based generalisation* (EBG), is presented. A theoretical comparison of EBG and MkTR shows that while it is by far the more complex of the two, MkTR *can, unlike* EBG*, guarantee both the optimality of the partial plans it produces, and the tractability of the constraints they contain.*

For convenience, the definitions will be repeated here, albeit in a slightly modified form. A POPI is a tuple $P = \langle O, \prec, \Phi \rangle$ where O is a set of operators, $\prec$ is a strict partial order over O, and $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_n$ is a constraint formula where each Φ_i is of the form $\vec{t_1} = \vec{t_2}$ or $\vec{t_1} \neq \vec{t_2}$. A classical plan is an instantiation[6] of a POPI if it satisfies its ordering and binding constraints.

Two techniques are presented, *explanation-based order generalisation* (EOG) and *explanation-based precondition generalisation* (EPG), that relax a POPI's ordering and variable binding constraints, respectively. Both algorithms use a *validation structure* – a set of causal links that "explain" the plan's validity – as a guide for this relaxation. A validation structure links every consumer in a POPI with the unthreatened supporting producer that appears *first* in an arbitrary linearisation:

[6] While Kambhampati and Kedar [71] use the term *completions*, the term *instantiations* is used here for consistency

Definition 5.15. *Let $P = \langle O, \prec, \Phi \rangle$ be a POPI and total order $\prec'$ be an arbitrary linearisation of $\prec$. Then, a* **validation structure** *V_P is a set of causal links* s.t. $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle \in V_P$ iff*:*

1. $o_p \prec o_c$,
2. $\Phi \models \vec{u} = \vec{s}$, *and*
3. *for all o_t and $\vec{v}$* s.t. $\mathsf{thrtns}(o_t, q(\vec{v}))$, *either $o_t \prec o_p$, $o_c \prec o_t$ or $\Phi \models \vec{s} \neq \vec{v}$, and*
4. *for all $\langle o'_p, q(\vec{u})', o_c, q(\vec{s}) \rangle$* s.t. *conditions* 1–3 *hold, $o_p \prec o'_p$.*

The EOG algorithm removes all ordering constraints that are not required to protect the causal links in V_P:

Definition 5.16. *If $P = \langle O, \prec, \Phi \rangle$ is a POPI with validation structure V_P, then the* **explanation-based order generalisation** *of P is the POPI* $\mathrm{EOG}(P) = \langle O, \prec'^{+}, \Phi \rangle$ *where $o_1 \prec' o_2$* iff *there exists a $q(\vec{u})$, $q(\vec{s})$* s.t. *either:*

- $\langle o_1, q(\vec{u}), o_2, q(\vec{s}) \rangle \in V_P$,
- *there exists an o_3 and $q(\vec{v})$* s.t. $\langle o_2, q(\vec{s}), o_3, q(\vec{v}) \rangle \in V_P$, $o_1 \prec o_2$, $\mathsf{thrtns}(o_1, q(\vec{u}))$, *and* $\Phi \not\models \vec{u} \neq \vec{s}$, *or*
- *there exists an o_3 and $q(\vec{v})$* s.t. $\langle o_3, q(\vec{v}), o_1, q(\vec{u}) \rangle \in V_P$, $o_1 \prec o_2$, $\mathsf{thrtns}(o_2, q(\vec{s}))$ *and* $\Phi \not\models \vec{u} \neq \vec{s}$.

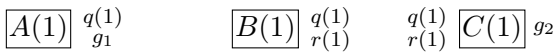
Similarly, EPG removes all binding constraints except those required by the validation structure, that is, a consumer must remain codesignated with its supporting producer, and must remain non-codesignated with any threat that can be ordered between them:

Definition 5.17. *If $P = \langle O, \prec, \Phi \rangle$ is a POPI with validation structure V_P, then the* **explanation-based precondition generalisation** *of P is the POPI* $\mathrm{EPG}(P) = \langle O, \prec, \Phi' \rangle$ s.t.*:*

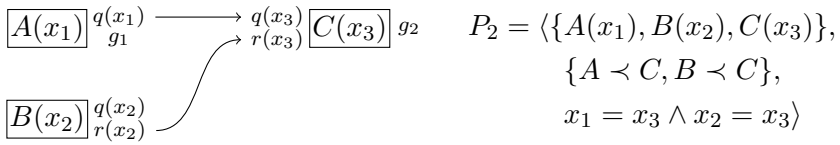
$$\Phi = \bigwedge_{\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle \in V_P} \Big(\vec{u} = \vec{s} \wedge \bigwedge_{\substack{o_t, q(\vec{v}): \\ \mathsf{thrtns}(o_t, q(\vec{v})), \\ o_t \not\prec o_p, o_c \not\prec o_t}} \vec{v} \neq \vec{s} \Big).$$

The process of *explanation based generalisation* relaxes both ordering and binding constraints by simply applying the two processes above in sequence: $\mathrm{EBG}(P) = \mathrm{EPG}(\mathrm{EOG}(P))$.

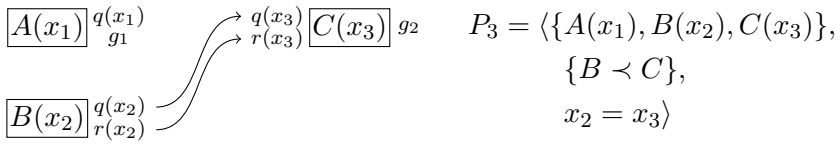
While MkTR finds a minimal CLP, there is no guarantee that EBG will produce an optimally relaxed POPI. Bäckström [8] demonstrated that when applied to POPs (special cases of POPIs), EOG does not guarantee even a *minimal deordering* (Definition 2.18) of its input, and a similar example can be used to demonstrate that EBG does not optimally relax its input. Figure 5.5a depicts the ground, totally ordered plan P_1 that achieves the goal $\{g_1, g_2\}$ from the initial state $\varnothing$, and Figure 5.5b shows the POPI, P_2, and validation structure that result from applying EBG to P_1. As a validation structure links each consumer to its earliest producer, P_2 is not a



(a) Classical plan P_1 that achieves the goal $\{g_1, g_2\}$.



(b) POPI P_2 and its validation structure, as produced by EBG from P_1.



(c) POPI P_3 and its validation structure, a more general relaxation of P_1.

Figure 5.5: Non-optimal EBG example.

minimum relaxation of P_1. Plan P_3 in Figure 5.5c uses a different validation structure, and is a more general relaxation of P_1: both its ordering and variable binding constraints are weaker than P_2's. As P_3 does not require that x_2 and x_3 be codesignated or that $A \prec C$, it allows for alternative instantiations such as $B(1), C(1), A(2)$ which would be disallowed by P_2.

While MkTR guarantees the tractability of the partial plans that it produces, EBG does not. The authors show that instantiating a POPI is a polynomial time operation under the assumption that variables have infinite domains, however, it is not unreasonable to expect to encounter domains where this assumption does not hold. If a domain involves "real" objects then they are likely to be finite, and indeed, the challenge of many IPC domains, such as *barman* and *parking* is the careful manipulation of limited resources. In such cases, despite the limited syntax for POPI variable binding constraints, determining the satisfiability of a finite-domained POPI is NP-complete:

Theorem 5.9. Finite Domain POPI Satisfiability. *Let $P = \langle O, \prec, \Phi \rangle$ be a POPI such that $\mathsf{vars}(O) = \{x_1, \ldots, x_n\}$ and let $D_{x_1}, \ldots, D_{x_n}$ be sets of constants representing variable domains. Deciding the existence of a classical plan $\vec{o}\theta$ such that* (i) *$\vec{o}\theta$ is an instantiation of P, and* (ii) *for all $x \in \mathsf{domain}(\theta)$, $\theta(x) \in D_x$ is* NP*-complete.*

This result has implications for EBG's runtime complexity. To construct a validation structure it must be determined whether the POPI's variable binding constraints entail a literal (e.g., whether $\Phi \models \vec{t} = \vec{u}$), and EOG must determine whether they do not (e.g., whether $\Phi \not\models \vec{t} \neq \vec{u}$), and these are NP and co-NP-complete problems. However, in the special case when the input plan is ground (i.e., a POP or classical plan), EBG runs in polynomial time.

Despite the above criticisms, it is not clear that MkTR is always preferable to EBG. If, for example, plan sizes are small enough, then the

intractability results above may not be a problem in practice. However, an experimental comparison of MkTR and EBG (based on model counts, treewidths and instantiation time) is left for future work.

5.7.2 Comparison with Other Least Commitment Approaches

Much of the literature on plan post-processing for flexibility (with the notable exception of explanation based generalisation, above) is on plan de/reordering with the aim of finding *minimum orderings* (Section 2.2.2.1). It thus differs from this work in a number of ways. While de/reordering aims to increase ordering flexibility, this chapter is concerned with *relaxing both action orderings and variable bindings*. Other differences stem from the ways in which POPs and partial plans represent constraints. POPs define a partial order over a set of ground actions, while a partial plan's constraints are expressed in $\mathcal{L}_C$ (Definition 5.1), a fragment of first-order logic. This results in significant differences in the theoretical properties of the two approaches.

The optimality definitions in Section 5.3.1 differ from those in the POP de/reordering literature. POPs represent ordering constraints as a binary relation (i.e., a set), resulting a distinction between, for example, a *minimal deorder* (i.e., removing a constraint will render the POP invalid) and *minimum deorder* (the smallest of all deorders). However, there is no equivalent minimal/minimum distinction for partial plans. It follows from Theorems 5.5 and 5.6 that if $P = \langle O, \phi \rangle$ and $Q = \langle O, \phi' \rangle$ are valid partial plans and P is a minimum relaxation, then P is a relaxation of Q (i.e., $\phi' \models \phi$). Thus, if a partial plan is a "minimal relaxation", in the sense that it has no strict relaxations, then it must also be a minimum relaxation.

The complexity results in Sections 5.2 and 5.3 are quite different to those found in the POP de/reordering literature. For example, while the problem of finding a minimum de/reordering of a POP is NP-hard [8], the equivalent problem in the context of partial plans is trivial: a minimum relaxation of a partial plan can be directly defined in $\mathcal{L}_C$ (Theorem 5.6). However, this expressivity comes with a computational cost. Validating or instantiating (i.e., finding a linearisation of) a POP takes polynomial time, but the equivalent problems for partial plans are DP-complete and NP-complete, respectively (Theorems 5.2 and 5.3). Thus, in POP de/reordering the challenge is to find effective techniques for the intractable relaxation problem, but in the context of partial plans, the challenge is to *identify the special cases in which instantiation is tractable* (Section 5.2.3), and to *develop techniques to find minimal but tractable partial plans* (Section 5.4).

Section 5.4 introduced the idea of a *causal link plan*, a generalised plan that represents constraints as a *disjunctive causal structure*. Such structures have previously been used in the context of partial-order planning (Section 2.2.1). Kambhampati [70] provides a definition of POPI validity *w.r.t.* to disjunctive causal structures that requires each consumer to be supported by an unthreatened causal link from the structure (i.e., POCL-validity as in Definition 2.7 applied to POPIs). A partial-order planner is introduced, which, unlike most partial-order planners, maintains *multiple* possible supporting links for each consumer. This prevents the planner from

prematurely committing to a single supporter for a consumer, and allows it to find less committed plans.

5.7.2.1 Future Work

Riddle et al. [109] break symmetries in classical planning instances by reformulating the PDDL into a "bagged" representation, where sets of interchangeable objects are replaced with resources, that is, counter variables (Section 2.3.4). This approach reduces solve times by eliminating irrelevant permutations of objects, but can also provide flexibility, as any such permutation can be selected at execution time. However, the approach only works when individual objects are interchangeable, and cannot account for interchangeable *collections* of objects (as studied extensively in Chapter 3), provide flexibility for domain objects that are not interchangeable, or provide ordering flexibility. Nevertheless, this raises the question of what proportion of the instantiations found by MkTR are simply symmetrical variations of the original plan. This analysis is left for future work.

While this chapter generalises plan de/reordering, techniques for optimising other aspects of plans have been studied (Section 2.2). Of particular note is the work of Muise et al. [91], which studies the minimisation of both ordering constraints *and plan size*. Bäckström's POP optimality definitions are extended to account for differences in POP size, and a MaxSAT-based approach to finding such optima is introduced. Future work could apply this to partial plans. An expanded constraint language *clang* that can express whether an operator must appear in a plan would allow a partial plan's instantiations to differ not just in ordering and variable bindings, but also in plan size.

CHAPTER 6

Conclusion

This thesis introduced, theoretically analysed and empirically evaluated novel techniques for representing and generating *flexible* plans. In particular, it was demonstrated that allowing an agent to reason about both action orderings and domain objects when optimising and executing plans results in significantly more execution-time flexibility.

Chapter 4 focused on reasoning about, and possibly modifying, domain objects at *optimisation time*, specifically when relaxing a classical plan into a partial-order plan (POP), and Chapter 5 studied the problem of allowing the agent to choose domain objects at *execution time*. In both cases, an experimental comparison with *explanation-based order generalisation* [71] (EOG), an effective albeit non-optimal technique for optimising orderings, showed that simultaneously optimising a plan's actions' orderings and parameters results in significantly more flexible plans than optimising orderings alone, although this additional flexibility comes at significant computational cost.

Effectively reasoning about domain objects to maximise flexibility comes with a number of interrelated computational challenges. Firstly, a *plan representation* must be found that allows the agent to *(i)* decide on both action ordering and domain objects at execution time, and *(ii)* determine which of its available actions are compatible with its plan in a time suitable for real-time decision-making. Further, the plan should be expressed in a form that lends itself to algorithmic manipulation, for the purposes of generation and optimisation. Chapter 4 used an existing representation, namely a POP, that meets all criteria except allowing domain objects to be selected at execution time, and thus focused on the problem of finding a fixed set of objects that maximises ordering flexibility. The central challenge in Chapter 5 was the development of a new plan representation, a *partial plan*, that satisfies all of the above criteria.

The second challenge is the development of *optimality criteria* that define the degree to which a plan can provide execution-time flexibility, and then determining the complexity of the problems of finding plans that satisfy those criteria. Chapters 4 and 5 introduced theoretical frameworks for comparing the flexibility of POPs with different variable bindings, and assessing the flexibility and tractability of partial plans, respectively.

Finally, algorithms that *generate* optimally flexible plans must be designed and implemented. Rather than synthesising such plans directly, both Chapters 4 and 5 take an *optimisation* approach, and make a given classical plan more flexible by *relaxing* its actions' orderings and variable bindings. The effectiveness of these approaches is dependent on their handling of the combinatorial explosion that results from allowing variable bindings to change. In both chapters, the range of possible bindings is exponentially reduced by only considering those that are relevant to the plan's *causal structure*, and by breaking symmetries with the Multi Lex technique introduced in Chapter 3.

The contributions of this thesis were as follows:

Chapter 3: Breaking generalised symmetries in matrix and graph models This chapter showed that problems related to plan relaxation and reasoning about alternative domain objects can have highly *symmetrical* search spaces (Section 2.3), that is, they can contain an exponential number of equally valid and flexible plans that are identical up to a permutation of orderings, domain objects, or both. Thus, before the plan relaxation techniques were studied directly, Chapter 3 introduced an effective, compact and domain-independent optimisation technique that accelerates the relaxation process by breaking symmetries.

A small example from a synthetic domain demonstrated that symmetries can occur in a plan's *causal structure*, that is, the mapping between its actions' effects and preconditions that determines its validity and variable bindings, and partially determines its ordering constraints. A causal structure can be naturally represented as a matrix of Boolean variables. While standard approaches to symmetry breaking in matrix and graph models break symmetries in which rows, columns or vertices are *individually* interchangeable, Chapter 3 identified a generalised form of symmetry in which they are only interchangeable in *combinations*. A series of examples showed that these symmetries can occur in many plan optimisation problems and CSP benchmark domains [1], and cannot be broken with standard techniques such as Double Lex [42] or Snake Lex [56].

Thus, a novel symmetry breaking constraint was introduced, Multi Lex, that generalises Double Lex and can break generalised symmetries in matrix and graph models, including directed graphs with self-loops. An empirical evaluation over a series of unconstrained matrix, simple graph and DAG models of different sizes and symmetries showed that *(i)* these generalised symmetries result in a factorial size increase in the search space, and *(ii)* Multi Lex is a compact and efficient symmetry breaking constraint that removes all but a constant factor of non-canonical solutions.

Chapter 4: Optimising partial-order plans by modifying orderings and variable bindings This chapter demonstrated that reasoning about domain objects while optimising plans allows POPs that are significantly more flexible to be found. A brief opening example taken from the IPC *rovers* domain showed that some plans *cannot* be made more flexible unless the optimisation process is allowed to modify the domain objects used by the plan.

A theoretical framework based on the idea of *reinstantiated deordering* and *reinstantiated reordering* – generalisations of the well-known *deordering* and *reordering* processes [8] (Section 2.2.2.1) in which variable bindings can also change – is used to compare the relative flexibility of POPs that use different domain objects. Under this framework, a *reinstantiated reorder* of a plan uses the same action *types* but with possibly different variable bindings and orderings, and amongst those, a *minimum reinstantiated reorder* has the fewest ordering constraints.

A practical method for finding such optimal plans was introduced and empirically evaluated. The technique is an optimisation and generalisation of that of Muise et al. [91], in which the plan optimisation problem is reduced to an instance of the partial weighted MaxSAT problem (Section 2.1.5).

The MaxSAT encoding has a number of features that tackle the combinatorial explosion that results from considering all ways of rebinding variables. All refer to the possible *causal structures* of the final optimised POP. The first reduces the encoding size by removing any variables or clauses that are not used to enforce validity under some possible structure. The second accelerates the search process with additional *symmetry breaking* constraints that rule out solutions with non-canonical causal structures. To detect these symmetries, the idea of a *plan description graph* was introduced, an undirected coloured graph with automorphisms that correspond to the plan's symmetries. The symmetries were broken with the Multi Lex constraint as introduced in Chapter 3.

An empirical evaluation over all previous IPC STRIPS domains showed that allowing parameters to change results in a large increase in flexibility, but at a large computational cost. In the 55.6% of cases in which a result was returned within the time limit, allowing the input plan to be reinstantiated resulted in a 21.1% *flex* increase over the baseline, as opposed to a 3.3% increase when bindings must remain static.

Chapter 5: Finding tractable and flexible partial plans This chapter demonstrated that allowing an agent to select domain objects at execution time provides a significant increase in flexibility. Indeed, a modification of the example in Chapter 4 shows that there are plans that can *only* provide flexibility if the agent can vary actions' parameters during the course of running the plan.

Much of the chapter was concerned with the problem of plan representation, in particular, how to express a plan that allows choices regarding domain object use to be made quickly at execution time. To this end, the idea of a *partial plan* was introduced, a partially specified plan comprising a set of *operators*, that is, schematised actions with parameters replaced by variables, and a *constraint formula*, that is, a set of constraints expressed in a fragment of first-order logic defining the allowable combinations of orderings and variable bindings. While the problem of finding a classical plan that is consistent with a constraint formula is NP-complete, a parameterised complexity analysis showed that the problem is polynomial, and thus suitable for real-time decision-making, when limited to formulae of bounded *treewidth* (a measure of the "cyclicity" of a formula's underlying graph).

A theoretical framework was introduced to assess the amount of flex-

ibility provided by a partial plan. Central to this framework is the notion of a *minimal k-treewidth relaxation*, that is, a partial plan with constraints that cannot be made any weaker without its treewidth exceeding some predetermined parameter.

To simplify the process of generating such a plan, a further representation was introduced in which constraints are expressed as a set of allowable causal links: a simpler representation that inherits many of the tractability properties of constraint formulae while being easier to generate and optimise algorithmically. This allowed the introduction of MkTR, an fpt-algorithm that generates a minimal k-treewidth relaxation of a plan by greedily and iteratively relaxing its causal structure while keeping its treewidth below an input parameter. As in Chapter 4, a number of optimisations were introduced, such as symmetry breaking with the Multi Lex constraint, to accelerate this process.

An empirical evaluation over all previous IPC STRIPS domains showed that with a maximum treewidth of two (i.e., searching for plans of quadratic "complexity"), MkTR can find a partial plan that represents 163.2% more plans than the EOG baseline. Increasing the maximum treewidth to five increased this to 302.3% with little change in the time required to find a classical plan that satisfies the constraints.

6.1 Future Work

Earlier chapters discussed various possible avenues for future research, such as extending Multi Lex to handle rotational symmetries (Section 3.5), finding sets of variable bindings that allow for greater reduction in POP size or makespan, or better *block deorderings* [120] (Section 4.6), and extending partial plans to allow for instantiations that differ in size (Section 5.7). This thesis will conclude with a discussion of some broader possibilities for future work.

Exploiting symmetries for plan flexibility Chapter 5 showed that a classical plan can be relaxed into a partial plan with huge numbers of instantiations, but raised the question of how many of these are simply symmetrical permutations of the original. Given the relative ease with which a plan's symmetries can be computed (Section 4.4.2.2 of Chapter 4) this suggests an area of future work in which a plan comprising a canonical classical plan and its symmetry group provides flexibility by compactly representing an exponential number of symmetrical variations.

The computational challenge would be allowing an agent to determine, in a reasonable time, which of its available actions are compatible with such a plan. More specifically, the agent must determine which actions are the first step of a classical plan with a canonical form that is identical to that held by the plan. While canonicalisation is NP-hard [83], approximate canonicalisation techniques have proved to be effective in practice, for example in the work of Pochter et al. [100] and Shleyfman et al. [118]. Indeed, Section 5.5.4 of Chapter 5 introduced an approximation scheme that extends this work with the highly effective Multi Lex symmetry breaking constraint (Section 3.4 of Chapter 3). This suggests the possibility that

determining which actions are compatible with the plan could be approximated in a time suitable for execution-time decision-making with minimal loss of flexibility. But of course, confirming this, and evaluating the effectiveness of this approach, is left for future work.

Flexibility from plan diversity An implicit assumption in this thesis is that the flexibility of a POP or a partial plan can be measured by the size of the set of classical plans that it represents. However, as it is possible that all elements of the set are nearly entirely identical, a more sophisticated approach could maximise the set's *diversity*. A benefit of maximising diversity rather than cardinality is that the likelihood of all plans in the set sharing a common point of failure is reduced. For example, a plan in a logistics domain may allow for many different shipping routes, but if they all use the Suez Canal then a single blockage will render the whole plan useless. By contrast, a plan that provides a *qualitatively distinct* range of shipping routes, or perhaps includes air or road freight options, is more likely to provide useful alternatives.

Typically, the *diversity* of a set of plans is derived from the "distance" between each pair of plans in the set, and ideally the set would evenly cover as large a proportion of the solution space as possible [94]. Since it is preferable for a plan to provide flexibility in both actions' orderings and parameters, a distance measure is required that takes both of these into account. Many measures are based on a comparison of the plans' ground actions, for example edit distance or *stability* (i.e., the proportion of shared actions [31, 46]). Unfortunately, this can obscure degrees of difference between variable bindings. For example, these measures would consider the three one-step plans $\langle \mathit{grasp}(\mathsf{LH}, \mathsf{SH}_1) \rangle$, $\langle \mathit{grasp}(\mathsf{LH}, \mathsf{SH}_2) \rangle$ and $\langle \mathit{grasp}(\mathsf{RH}, \mathsf{SH}_3) \rangle$ to be equally different from each other despite the common domain object in the first two. Distances derived from differences in the plans' state space transitions [125] suffer the same problem. Measures derived from set-wise comparisons of plans' causal structures [125] will recognise degrees of difference between variable bindings (since the causal links define the actions' parameters), but not reorderings of plans with the same causal links. Thus, future work that develops a diversity measure for sets of plans that differ in both their ordering and domain object usage will help to determine the amount of flexibility provided by that set.

A further area of future work is the development of plan relaxation algorithms that optimise the diversity of the final POP or partial plan. A challenge here is balancing the three competing optimisation objectives, namely the number of classical plans represented by the final plan, their diversity, and, in the case of partial plans, the plan's tractability. The problem of finding the most diverse k elements of a set (i.e., with the largest sum of pairwise distances) is known to be NP-hard [78] and in FPT when parameterised with k [10]. Thus, the complexity of the problem of finding an optimally diverse relaxation of a plan is likely to be at least as hard as the relaxation problems studied in this thesis.

Bibliography

[1] CSPLib: A problem library for constraints. http://www.csplib.org, 1999.

[2] Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Exploiting symmetries by planning for a descriptive quotient. In *Proc. 24th Int. Joint Conf. Artif. Intell.*, pages 1479–1486, 2015.

[3] Meysam Aghighi and Christer Bäckström. Plan reordering and parallel execution – a parameterized complexity view. In *Proc. 31st AAAI Conf. Artif. Intell.*, pages 3540–3546, 2017.

[4] Meysam Aghighi, Peter Jonsson, and Simon Ståhlberg. Tractable cost-optimal planning over restricted polytree causal graphs. In *Proc. 29th AAAI Conf. on Artif. Intell.*, pages 3225–3231, 2015.

[5] Yusra Alkhazraji, Martin Wehrle, Robert Mattmüller, and Malte Helmert. A stubborn set algorithm for optimal planning. In *20th Eur. Conf. Artif. Intell. Including Prestigious Appl. of Artif. Intell.*, pages 891–892, 2012.

[6] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(9):1117–1137, 2003.

[7] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[8] Christer Bäckström. Computational aspects of reordering plans. *J. of Artif. Intell. Research*, 9:99–137, 1998.

[9] Rosy Barruffi, Evelina Lamma, Paola Mello, and Michela Milano. Least commitment on variable binding in presence of incomplete knowledge. In *Recent Advances in AI Planning, Proc. 5th Eur. Conf. Planning*, pages 159–171, 1999.

[10] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In *Proc. 29th Int. Joint Conf. Artif. Intell.*, pages 1119–1125, 2020.

[11] Pascal Bercher and Conny Olz. POP ≡ POCL, right? Complexity results for partial order (causal link) makespan minimization. In *Proc. 34th Conf. on Artif. Intell.*, pages 9785–9793, 2020.

[12] Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *Proc. 16th Int. Conf. Integration of Constraint Program., Artif. Intell., and Operations Res.*, pages 39–56, 2019.

[13] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, Amsterdam, Netherlands, 2009. IOS Press.

[14] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1–2):281—300, 1997.

[15] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. 25th Ann. ACM Symp. Theory of Comput.*, pages 226–234, 1993.

[16] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.

[17] Blai Bonet, Gábor Loerincs, and Hector Geffner. A robust and fast action selection mechanism for planning. In *Proc. 14th Nat. Conf. Artif. Intell. and 9th Innovative Appl. of Artif. Intell. Conf.*, pages 714–719, 1997.

[18] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

[19] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Auton. Agents Multi Agent Syst.*, 19(3):297–331, 2009.

[20] Graham R. Brightwell and Peter Winkler. Counting linear extensions is #P-complete. In *Proc. 23rd Annu. ACM Symp. on Theory of Comput.*, pages 175–181, 1991.

[21] Jonathan F. Buss and Tarique Islam. Simplifying the weft hierarchy. *Theoret. Comput. Sci.*, 351(3):303–313, 2006. Parameterized and Exact Computation.

[22] David Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32(3): 333–377, 1987.

[23] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

[24] Yixin Chen and Guohui Yao. Completeness and optimality preserving reduction for planning. In *Proc. 21st Int. Jont Conf. Artif. Intell.*, pages 1659—-1664, 2009.

[25] Yixin Chen, You Xu, and Guohui Yao. Stratified planning. In *Proc. 21st Int. Joint Conf. Artif. Intell.*, pages 1665–1670, 2009.

[26] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[27] Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artif. Intell.*, 159(1-2):127–206, 2004.

[28] Michael Codish, Alice Miller, Patrick Prosser, and Peter James Stuckey. Breaking symmetries in graph representation. In *Proc. 23rd Int. Joint Conf. on Artif. Intell.*, pages 510–516, 2013.

[29] Amanda Coles and Andrew Coles. Have I been here before? State memoization in temporal planning. In *Proc. 26th Int. Conf. Automated Planning and Scheduling*, pages 97–105, 2016.

[30] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proc. 20th Int. Conf. Automated Planning and Scheduling*, pages 42–49, 2010.

[31] Alexandra Coman and Hector Muñoz-Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proc. 25th AAAI Conf. Artif. Intell.*, pages 946–951, 2011.

[32] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proc. 5th Int. Conf. Princ. of Knowledge Representation and Reasoning*, pages 148–159, 1996.

[33] Minh Binh Do and Subbarao Kambhampati. Improving temporal flexibility of position constrained metric temporal plans. In *Proc. 13th Int. Conf. Automated Planning and Scheduling*, pages 42–51, 2003.

[34] Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proc. 22nd Int. Conf. Automated Planning and Scheduling*, pages 343—-347, 2012.

[35] Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Symmetry breaking: Satisficing planning and landmark heuristics. In *Proc. 23rd Int. Conf. Automated Planning and Scheduling*, pages 298—-302, 2013.

[36] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, NY, USA, 1999.

[37] Hani Elgabou. Encoding the lexicographic ordering constraint in satisfiability modulo theories. Master's thesis, Univ. York, 2015.

[38] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1):105–131, 1996.

[39] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2 (3/4):189–208, 1971.

[40] Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Symmetry in matrix models. Technical Report 2001-022, Dept. Inf. Technol., Uppsala Univ., Uppsala, Sweden, September 2001.

[41] Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kzltan, Ian Miguel, and Toby Walsh. Matrix modelling. In *Proc. CP2001 Post Conference Workshop Modelling and Problem Formulation*, pages 1–7, 2001.

[42] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Proc. 8th Int. Conf. Princ. Pract. Constraint Program.*, pages 462–476, 2002.

[43] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kızıltan, Ian Miguel, and Toby Walsh. Matrix modelling: Exploiting common patterns in constraint programming. In *Proc. Int. Workshop on Reformulating Constraint Satisfaction Problems*, pages 27–41, 2002.

[44] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proc. 16th Int. Joint Conf. Artif. Intell.*, pages 956–961, 1999.

[45] Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In *Proc. 6th Int. Conf. Artif. Intell. Planning Systems*, pages 83–91, 2002.

[46] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proc. 16th Int. Conf on Automated Planning and Scheduling*, pages 212–221, 2006.

[47] Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. 8th Nat. Conf. Artif. Intell.*, volume 1, pages 4–9, 1990.

[48] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning.* Morgan & Claypool Publishers, 2013.

[49] Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *Proc. 28th Int. Conf. Automated Planning and Scheduling*, pages 88–96, 2018.

[50] Alfonso Gerevini and Lenhart Schubert. Accelerating partial-order planners: Some techniques for effective search control and pruning. *J. of Artif. Intell. Research*, 5(1):95–137, 1996.

[51] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann, May 2004.

[52] Daniel Gnad, Jörg Hoffmann, and Carmel Domshlak. From fork decoupling to star-topology decoupling. In *Proc. 8th Annu. Symp. on Combinatorial Search*, pages 53–61, 2015.

[53] Daniel Gnad, Martin Wehrle, and Jörg Hoffmann. Decoupled strong stubborn sets. In *Proc. Twenty-Fifth Int. Joint Conf. Artif. Intell.*, pages 3110–3116, 2016.

[54] Daniel Gnad, Álvaro Torralba, Alexander Shleyfman, and Jörg Hoffmann. Symmetry breaking in star-topology decoupled search. In *Proc. 27th Int. Conf. Automated Planning and Scheduling*, pages 125–134, 2017.

[55] Daniel Gnad, Jörg Hoffmann, and Martin Wehrle. Strong stubborn set pruning for star-topology decoupled state space search. *J. of Artif. Intell. Research*, 65:343–392, 2019.

[56] Andrew Grayland, Ian Miguel, and Colva M. Roney-Dougal. Snake Lex: An alternative to Double Lex. In *Proc. 5th Int. Conf. Princ. Pract. Constraint Program.*, pages 391–399, 2009.

[57] Frank Harary and Edgar M. Palmer. *Graphical Enumeration.* Addison-Wesley, Boston, MA, USA, 1973.

[58] M. A. Harrison. On the number of classes of binary matrices. *IEEE Trans. Comput.*, 22(12):1048—-1052, 1973.

[59] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. 5th Int. Conf. Artif. Intell.*, pages 140–149, 2000.

[60] Patrik Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. 5th Eur. Conf. Planning*, pages 308–318, 1999.

[61] Daniel S. Heller, Aurojit Panda, Meinolf Sellmann, and Justin Yip. Model restarts for structural symmetry breaking. In *Proc. 14th Int. Conf. Princ. Pract. Constraint Program.*, pages 539–544, 2008.

[62] Sarah Hickmott and Sebastian Sardiña. Optimality properties of planning via Petri net unfolding: A formal analysis. In *Proc. 19th Int. Conf. Automated Planning and Scheduling*, pages 170–177, 2009.

[63] Sarah Hickmott, Jussi Rintanen, Sylvie Thiébaux, and Lang White. Planning via Petri net unfolding. In *Proc. 20th Int. Joint Conf. on Artif. Intell.*, pages 1904–1911, 2007.

[64] Jörg Hoffmann and Ronen I. Brafman. Contingent planning via heuristic forward search witn implicit belief states. In *Proc. 15th Int. Conf. Automated Planning and Scheduling*, pages 71–80, 2005.

[65] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7): 507–541, 2006.

[66] D.F. Holt, B. Eick, and E.A. O'Brien. *Handbook of Computational Group Theory*. Discrete Mathematics and Its Applications. CRC Press, Boca Raton, FL, USA, 2005.

[67] David Joslin and Martha E. Pollack. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proc. 12th Nat. Conf. Artif. Intell.*, pages 1004–1009, 1994.

[68] David Joslin and Amitabha Roy. Exploiting symmetry in lifted CSPs. In *Proc. 14th Nat. Conf. Artif. Intell.*, pages 197–202, 1997.

[69] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *Proc. 17th Nat. Conf. Artif. Intell. and 12th Conf. on Innovative Appl. of Artif. Intell.*, pages 157–162, 2000.

[70] Subbarao Kambhampati. Multi-contributor causal structures for planning: A formalization and evaluation. *Artif. Intell.*, 69(1-2):235–278, 1994.

[71] Subbarao Kambhampati and Smadar Kedar. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artif. Intell.*, 67(1):29–70, 1994.

[72] Subbarao Kambhampati and Dana S. Nau. On the nature and role of modal truth criteria in planning. *Artif. Intell.*, 82(1-2):129–155, 1996.

[73] Richard M. Karp. Reducibility among combinatorial problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 – From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010.

[74] George Katsirelos, Nina Narodytska, and Toby Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. In *Proc. 16th Int. Conf. Princ. Pract. Constraint Program.*, pages 305–320, 2010.

[75] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proc. 10th Eur. Conf. Artif. Intell.*, pages 359–363, 1992.

[76] Sarah Keren, Avigdor Gal, and Erez Karpas. Strong stubborn sets for efficient goal recognition design. In *Proc. 28h Int. Conf. Automated Planning and Scheduling*, pages 141–149, 2018.

[77] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In *Proc. 20th Int. Conf. Theory and Appl. of Satisfiability Testing*, pages 449–456, 2017.

[78] Ching-Chung Kuo, Fred Glover, and Krishna S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming*. *Decision Sciences*, 24(6):1171–1185, 1993.

[79] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proc. 19th Int. Conf. Theory and Appl. of Satisfiability Testing*, pages 123–140, 2016.

[80] Vladimir Lifschitz. On the semantics of STRIPS. In Michael Georgeff and Amy Lansky, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, San Mateo, CA, 1987.

[81] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proc. 31st AAAI Conf. Artif. Intell.*, pages 3590–3596, 2017.

[82] Michael L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proc. 14th Nat. Conf. Artif. Intell.*, pages 748–754, 1997.

[83] Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups And Computation, Proc. DIMACS Workshop*, volume 11 of *DIMACS Series in Discrete Math. and Theoret. Comput. Sci.*, pages 139–175, 1991.

[84] David Mcallester and David Rosenblitt. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on Artif. Intell.*, pages 634–639, 1991.

[85] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symbolic Comput.*, 60(0):94–112, 2014.

[86] Christopher Mears, Maria Garcia de la Banda, and Mark Wallace. On implementing symmetry detection. *Constraints*, 14(4):443–477, 2009.

[87] Pedro Meseguer and Carme Torras. Exploiting symmetries within constraint satisfaction search. *Artif. Intell.*, 129(1-2):133–163, 2001.

[88] Alice Miller and Patrick Prosser. Diamond-free degree sequences. *CoRR*, abs/1208.0460, 2012. URL http://arxiv.org/abs/1208.0460.

[89] Christian Muise. *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution.* PhD thesis, Univ. Toronto, 2014.

[90] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. 22nd Int. Conf. Automated Planning and Scheduling*, pages 172–180, 2012.

[91] Christian Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *J. of Artif. Intell. Research*, 57:113–149, 2016.

[92] Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proc. 28th AAAI Conf. Artif. Intell.*, pages 2322–2329, 2014.

[93] T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, 1989.

[94] Karen L. Myers and Thomas J. Lee. Generating qualitatively different plans through metatheoretic biases. In Jim Hendler and Devika Subramanian, editors, *Proc. 16th Nat. Conf. Artif. Intell.*, pages 570–576.

[95] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proc. 17th Int. Joint Conf. on Artif. Intell.*, volume 1, pages 459–464, 2001.

[96] Conny Olz and Pascal Bercher. Eliminating redundant actions in partially ordered plans – a complexity analysis. In *Proc. 29th Int. Conf. on Automated Planning and Scheduling*, pages 310–319, 2019.

[97] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. Syst. Sci.*, 28 (2):244–259, 1984.

[98] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. Princ. of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[99] Mark A. Peot and David E. Smith. Threat-removal strategies for partial-order planning. In *Proc. 11th Nat. Conf. Artif. Intell.*, pages 492–499, 1993.

[100] Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In *Proc. 25th AAAI Conf. Artif. Intell.*, 2011.

[101] G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, 68:145–254, 1937.

[102] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *J. of Artif. Intell. Research*, 4:287–339, 1996.

[103] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proc. 7th Int. Symp. on Methodologies for Intell. Syst.*, pages 350–361, 1993.

[104] Jean-Francois Puget. Breaking symmetries in all different problems. In *Proc. 19th Int. Joint Conf. Artif. Intell.*, pages 272–277, 2005.

[105] Jean-François Puget. Automatic detection of variable and value symmetries. In *Princ. Pract. Constraint Program.*, pages 475–489, 2005.

[106] Jean-Francois Puget. Breaking all value symmetries in surjection problems. In *Proc. 11th Int. Conf. Princ. Pract. Constraint Program.*, pages 490–504, 2005.

[107] Jean-François Puget. An efficient way of breaking value symmetries. In *Proc. 21st Nat. Conf. on Artif. Intell. and 18th Innovative Appl. of Artif. Intell.*, pages 117–122, 2006.

[108] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. of Artif. Intell. Research*, 39(1):127–177, 2010.

[109] Pat Riddle, Jordan Douglas, Mike Barley, and Santiago Franco. Improving performance by reformulating PDDL into a bagged representation. In *Proc. 8th Workshop on Heuristics and Search for Domain-independent Planning*, pages 28–36, 2016.

[110] Jussi Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling*, pages 32–40, 2003.

[111] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. 14th Int. Conf. Automated Planning and Scheduling*, pages 345–354, 2004.

[112] Jussi Rintanen. Heuristics for planning with SAT. In *Proc. Int. Conf. Princ. Pract. Constraint Program.*, pages 414–428, 2010.

[113] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. of Algorithms*, 7(3):309–322, 1986.

[114] Ivan D. Rodriguez, Blai Bonet, Sebastian Sardiña, and Hector Geffner. Flexible FOND planning with explicit fairness assumptions. *CoRR*, abs/2103.08391, 2021. URL https://arxiv.org/abs/2103.08391.

[115] Gabriele Röger, Silvan Sievers, and Michael Katz. Symmetry-based task reduction for relaxed reachability analysis. In *Proc. 28th Int. Conf. Automated Planning and Scheduling*, pages 208–217, 2018.

[116] Buser Say, André A. Ciré, and J. Christopher Beck. Mathematical programming models for optimizing partial-order plan flexibility. In *Proc. 22nd Eur. Conf. Artif. Intell.*, pages 1044–1052, 2016.

[117] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In *Proc. 28th Int. Joint Conf. Artif. Intell.*, pages 1169–1176, 2019.

[118] Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and symmetries in classical planning. In *Proc. 29th AAAI Conf. Artif. Intell.*, pages 3371–3377, 2015.

[119] Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Appl. Math.*, 155(12):1539–1548, 2007.

[120] Fazlul Hasan Siddiqui and Patrik Haslum. Block-structured plan deordering. In *Proc. 25th Australas. Joint Conf. Advances in Artif. Intell.*, pages 803–814, 2012.

[121] Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical foundations for structural symmetries of lifted PDDL tasks. In *Proc. 29th Int. Conf. Automated Planning and Scheduling*, pages 446–454, 2019.

[122] N. J. A. Sloane. *A Handbook of Integer Sequences.* Academic Press, London, UK, 1973.

[123] Barbara M. Smith. Sets of symmetry breaking constraints. In *Proc. SymCon 2005, the 5th Int. Workshop on Symmetry in Constraints*, 2005.

[124] David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proc. 15th Nat. Conf. Artif. Intell.*, pages 889–896, 1998.

[125] Biplav Srivastava, Tuan Anh Nguyen, Alfonso Gerevini, Subbarao Kambhampati, Minh Binh Do, and Ivan Serina. Domain independent approaches for finding diverse plans. In *Proc. 20th Int. Joint Conf. Artif. Intell.*, pages 2016–2022, 2007.

[126] Edward P. K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, UK, 1993.

[127] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990, Proc. 10th Int. Conf. Appl. and Theory of Petri Nets*, pages 491–515, 1989.

[128] Roman van der Krogt and Mathijs de Weerdt. Plan repair as an extension of planning. In *Proc. 15th Int. Conf. Automated Planning and Scheduling*, pages 161–170, 2005.

[129] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing treewidth with LibTW. Technical report, Univ. Utrecht, Utrecht, Netherlands, 2006.

[130] Vincent Vidal and Héctor Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.*, 170(3):298—-335, 2006.

[131] Toby Walsh. General symmetry breaking constraints. In *Proc. 12th Int. Conf. Princ. Pract. Constraint Program.*, pages 650–664, 2006.

[132] Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proc. 24th Int. Conf. Automated Planning and Scheduling*, 2014.

[133] Martin Wehrle, Malte Helmert, Yusra Alkhazraji, and Robert Mattmüller. The relative pruning power of strong stubborn sets and expansion core. In *Proc. 23rd Int. Conf. on Automated Planning and Scheduling*, 2013.

[134] Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In *Proc. 24th Int. Joint Conf. Artif. Intell.*, pages 1712–1718, 2015.

[135] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[136] Anna Wilhelm, Marcel Steinmetz, and Jörg Hoffmann. On stubborn sets and planning with resources. In *Proc. Twenty-Eighth Int. Conf. Automated Planning and Scheduling*, pages 288–297, 2018.

[137] Dominik Winterer, Yusra Alkhazraji, Michael Katz, and Martin Wehrle. Stubborn sets for fully observable nondeterministic planning. In *Proc. 27th Int. Conf. Automated Planning and Scheduling*, pages 330–338, 2017.

[138] You Xu, Yixin Chen, Qiang Lu, and Ruoyun Huang. Theory and algorithms for partial order based reduction in planning. *CoRR*, abs/1106.5427, 2011. URL http://arxiv.org/abs/1106.5427.

[139] Qiang Yang and Alex Y. M. Chan. Delaying variable binding commitments in planning. In *Proc. 2nd Int. Conf. on Artif. Intell. Planning Systems*, pages 182–187, 1994.

[140] Håkan L. S. Younes and Reid G. Simmons. On the role of ground actions in refinement planning. In *Proc. 6th Int. Conf. on Artif. Intell. Planning Systems*, pages 54–62, 2002.

[141] Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *J. of Artif. Intell. Research*, 20:405–430, 2003.

APPENDIX A

Publications

Parts of this thesis appear in the following publications:

Max Waters, Bernhard Nebel, Lin Padgham, and Sebastian Sardiña, Plan Relaxation via Action Debinding and Deordering. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling* (ICAPS 2018) [Chapter 5].

Max Waters, Lin Padgham, and Sebastian Sardiña, Optimising Partial-Order Plans via Action Reinstantiation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (IJCAI 2020) [Chapter 4].

APPENDIX B

Proofs of Theorems in Chapter 3

B.1 Proof of Theorem 3.1

Theorem 3.1. MULTI LEX CORRECTNESS. *Let* $\mathbf{M}$ *be a matrix with multi-row-column symmetry group* Γ, Σ *be a generating set of* Γ *as in Definition 3.7 and* θ *be a ground substitution that is complete* w.r.t. *to the variables in* $\mathbf{M}$. *If* $\mathsf{can}(\mathbf{M}, \Gamma, \theta)$ *and* $\sigma \in \Sigma$, *then* $\theta(\mathbf{M}) \upharpoonright \langle R_\sigma, C_\sigma \rangle) \preceq_L \sigma(\theta(\mathbf{M})) \upharpoonright \langle R_\sigma, C_\sigma \rangle$ *holds.*

Proof. For ease of notation, let $\vec{l} = \mathbf{M} \upharpoonright \langle R_\sigma, C_\sigma \rangle$ and $\vec{r} = \sigma(\mathbf{M}) \upharpoonright \langle R_\sigma, C_\sigma \rangle$ be the left and right sides of the Multi Lex constraint for σ as in Definition 3.12, and let $\vec{v} = \mathbf{M}[1]^\frown \cdots ^\frown \mathbf{M}[n]$ be the concatenation of all rows in $\mathbf{M}$. It suffices to prove that $\theta(\vec{l}) \preceq_{\mathrm{L}} \theta(\vec{r})$.

As θ is canonical, $\theta(\vec{v}) \preceq_{\mathrm{L}} \theta(\sigma(\vec{v}))$ (Definition 3.9). Thus, either $\theta(\vec{v}) =_{\mathrm{L}} \theta(\sigma(\vec{v}))$ or $\theta(\vec{v}) \prec_{\mathrm{L}} \theta(\sigma(\vec{v}))$. The proof now continues by cases.

Case 1 Assume $\theta(\vec{v}) =_{\mathrm{L}} \theta(\sigma(\vec{v}))$. Then, for $1 \leq i \leq n.m$, $\theta(\vec{v}[i]) = \theta(\sigma(\vec{v}[i]))$, that is, σ leaves all elements unchanged. As all elements of $\vec{l}$ are in $\vec{v}$, and $\sigma(\vec{l}) = \vec{r}$, then for all $1 \leq i \leq |\vec{l}|$, $\vec{l}[i] = \vec{r}[i]$. Therefore, $\theta(\vec{l}) =_{\mathrm{L}} \theta(\vec{r})$ as desired.

Case 2 Assume $\theta(\vec{v}) \prec_{\mathrm{L}} \theta(\sigma(\vec{v}))$. Let s be the earliest point of difference between $\theta(\vec{v})$ and $\theta(\sigma(\vec{v}))$, that is, the index *s.t.* $\theta(\vec{v}[s]) < \theta(\sigma(\vec{v}[s]))$, and for all $1 \leq i < s$, $\theta(\vec{v}[i]) = \theta(\sigma(\vec{v}[i]))$ (Definition 2.20). Assume for now that $\vec{v}[s] \in \vec{l}$, that is, there is some t *s.t.* $\vec{v}[s] = \vec{l}[t]$. As $\vec{l}$ is a *sublist* of $\vec{v}$ (Definition 3.12), that is, it may have fewer items, but they are in the same order, it follows that $\theta(\vec{l}[t]) < \theta(\vec{r}[t])$, and for all $1 \leq i < t$, $\theta(\vec{l}[i]) = \theta(\vec{r}[i])$, that is, $\theta(\vec{l}) \prec_{\mathrm{L}} \theta(\vec{r})$. Thus, it suffices to prove that $\vec{v}[s] \in \vec{l}$.

Let k be the index *s.t.* $\vec{v}[k] = \sigma(\vec{v}[s])$. As defined above, $\vec{v}[s]$ is the first point of difference between $\theta(\vec{v})$ and $\theta(\sigma(\vec{v}))$. Therefore, $\theta(\vec{v}[s]) < \theta(\sigma(\vec{v}[s]))$, and so $\theta(\vec{v}[s]) < \theta(\vec{v}[k])$. Therefore, $s \neq k$. Further, as for all $1 \leq i < s$, $\theta(\vec{v}[i]) = \theta(\sigma(\vec{v}[i]))$, it follows that $k \geq s$. And so $s < k$.

Let p and q be indexes *s.t.* $\vec{v}[s] = \mathbf{M}[p][q]$ and let r and c be indexes *s.t.* $\vec{v}[k] = \mathbf{M}[r][c]$, that is, p and q are the row and column indexes of $\vec{v}[s]$, and r and c are the row and column indexes of $\vec{v}[k]$. As $\vec{v}[s] \neq \vec{v}[k]$, it follows that $\vec{v}[s]$ is mapped by σ into either a different row, a different column, or both, that is, either $p \neq r$, $q \neq c$, or both. The proof will now continue by (sub-)cases.

Case 2.1 Assume that $r \neq p$ (i.e, $\vec{v}[s]$ is mapped to a different row, but no assumptions are made as to whether it is mapped to a different column). If $r < p$, then $\vec{v}[s]$ would be in a lower row than $\vec{v}[i]$, and as $\vec{v}$ is the concatenation of rows, $k < s$ would hold. But, as established above, $s < k$. Thus, $r > p$ must hold. Therefore, $\vec{v}[i]$ is in a row that is mapped to a higher row by σ. As $\vec{l}$ contains all variables mapped into a higher row or column by σ (Definition 3.12), it follows that $\vec{v}[s] \in \vec{l}$, as desired.

Case 2.2 Assume that $p = r$. As either $p \neq r$ or $q \neq c$, it follows that $c \neq q$ (i.e, $\vec{v}[s]$ is mapped to the same row, and a different column). If $c < q$, then $\vec{v}[k]$ would be in the same row but a lower column than $\vec{v}[s]$, meaning that $k < s$. But, as $k > s$, $c > q$ must hold. Therefore, $\vec{v}[s]$ must be in a column that is mapped to a higher column by σ, and so $\vec{v}[s] \in \vec{l}$, as desired (Definition 3.12). □

B.2 Proof of Theorem 3.2

Theorem 3.2. Multi Lex Size Complexity. *Let* $\mathbf{M}$ *be an* $r \times c$ *matrix with multi-row-column symmetry group* Γ*, and* Σ *be the generating set of* Γ *as in Definition 3.7. Then, Multi Lex posts constraints of size* $\mathcal{O}(rc)$.

Proof. Let $\mathbf{M}$ be an $r \times c$ matrix, $\vec{r}_1, \ldots, \vec{r}_n$ be disjoint lists of rows of length m_r, $\vec{c}_1, \ldots, \vec{c}_n$ be disjoint lists of columns of length m_c, and Γ be a multi-row-column symmetry with generator Σ produced from those lists as in Definition 3.7.

From Definition 3.7, $|\Sigma| = n - 1$ and so Multi Lex posts $n - 1$ constraints. Each constraint compares m_r rows and m_c columns with their image under the permutation, and is thus of size $2.(m_r c + m_c r)$. However, as all $\vec{r}$ and $\vec{c}$ are disjoint, in the worst case $m_r = r/n$ and $m_c = c/n$, thus each constraint's size is bounded by $(4rc)/n$. As $n-1$ constraints are posted, the size of all constraints combined is bounded by $4rc$, i.e., $\mathcal{O}(rc)$. □

APPENDIX C

Proofs of Theorems in Chapter 4

C.1 Proof of Theorem 4.1

Theorem 4.1 states that determining whether a POP has a minimum reinstantiated deordering or reordering with less than k ordering constraints are both NP-complete. A proof of the NP-completeness of minimum reinstantiated reordering is presented below. The proof for minimum reinstantiated deordering is a trivial modification.

Theorem 4.1. MINIMUM REINSTANTIATED REORDERING *Given a POP* $P = \langle O, \theta, \prec \rangle$ *and an integer* $k > 0$*, determining whether there exists a POP* $Q = \langle O, \theta', \prec' \rangle$ s.t. Q *is a reinstantiated reordering (or deordering) of* P *and* $| \prec' | < k$ *is* NP*-complete.*

Proof. Let $P = \langle O, \theta, \prec \rangle$ be a POP and $k > 0$ an integer. To prove membership in NP, guess a POP $Q = \langle O, \theta', \prec' \rangle$ and verify in polynomial time that Q is valid, and $| \prec' | < k$.

Proof of NP-hardness is by reduction from the decision problem MINIMUM REORDERING, which is NP-complete [8] and asks whether, for any POP $P = \langle O, \theta, \prec \rangle$ and integer $k > 0$, there exists a valid POP $Q = \langle O, \theta, \prec' \rangle$ such that $| \prec | < k$ (Definition 2.18). Let $P = \langle O, \theta, \prec \rangle$ be a POP such that $O = \{o_I, o_1, \ldots, o_n, o_G\}$. First construct, from each constant $c \in \mathsf{consts}(O)$, the predicate symbol is_c. Next, construct the POP $P' = \langle O', \theta, \prec \rangle$ such that $O' = \{o'_I, o'_1, \ldots, o'_n, o_G\}$, where $\mathsf{pre}(o'_I) = \varnothing$, $\mathsf{post}(o'_I) = \mathsf{post}(o_I) \cup \{is_c(c) : c \in \mathsf{consts}(O)\}$, and for $1 \leq i \leq n$, $\mathsf{vars}(o'_i) = \mathsf{vars}(o_i)$, $\mathsf{post}(o'_i) = \mathsf{post}(o_i)$ and $\mathsf{pre}(o'_i) = \mathsf{pre}(o_i) \cup \{is_c(x) : x \in \mathsf{vars}(o'_i), c = \theta(x)\}$. The is_c preconditions in P''s operators mean that the bindings of all variables in P' are "fixed": any valid reinstantiation of P' cannot alter θ. It follows that there exists a reorder of P with fewer than k ordering constraints *iff* there exists a reinstantiated reorder of P' with fewer than k ordering constraints. □

C.2 Proof of Theorem 4.2

Theorem 4.2 states that finding a minimal reinstantiated de/reordering of a POP cannot be approximated to within a constant factor. As above, a

proof for minimum reinstantiated reordering is presented. The proof for minimum reinstantiated deordering is a trivial modification.

Theorem 4.2. Approximate Minimum Reinstantiated Reordering *The problem of finding a minimum reinstantiated deordering (or reordering) of a POP is not in* APX *unless* NP $\subseteq$ DTIME($n^{poly\ log\ n}$).

Proof. As the polynomial time reduction in the proof of Theorem 4.1 also preserves solutions, it is therefore an approximation preserving reduction. As Minumum Reordering cannot be approximated within a constant factor unless NP $\subseteq$ DTIME($n^{\text{poly log n}}$) [8], it follows that a minimum reinstantiated reorder cannot be either. □

C.3 Proof of Theorem 4.3

Theorem 4.3 asserts that an optimal solution to the MRR/MRD MaxSAT encoding of a POP P corresponds to a minimum reinstantiated de/reorder of P, respectively. This will be proved for the case of MRR, the case for MRD is a minor modification.

The following lemma establishes that any POP corresponding to a solution to the MRR MaxSAT encoding as in Definition 4.9 has well-formed ordering constraints:

Lemma C.1. *Let $P = \langle O, \theta, \prec \rangle$ be a POP, S be a solution to the* MRR MaxSAT *encoding of P, and* $\text{POP}(S) = \langle O, \theta', \prec' \rangle$ *be the POP corresponding to S. Then, $\prec'$ is acyclic and transitively closed.*

Proof. Assume that there is an o_1, o_2 and o_3 *s.t.* $o_1 \prec' o_2$ and $o_2 \prec' o_3$. From Definition 4.9, $\text{p}_{o_1 \prec o_2}, \text{p}_{o_2 \prec o_3} \in \mathcal{P}_\prec$ and $S(\text{p}_{o_1 \prec o_2}) = S(\text{p}_{o_2 \prec o_3}) = \mathit{true}$. Therefore, $\text{p}_{o_1 \prec o_3} \in \mathcal{P}_\prec$ (Definition 4.6), and from Formula 4.9, $S(\text{p}_{o_1 \prec o_3}) = \mathit{true}$, and so $o_1 \prec' o_3$. Therefore, $\prec'$ is transitively closed.

Assume that there is an o_1, o_2 *s.t.* $o_1 \prec' o_2$. From Definition 4.9, $\text{p}_{o_1 \prec o_2} \in \mathcal{P}_\prec$, and $S(\text{p}_{o_1 \prec o_2}) = \mathit{true}$. Either $\text{p}_{o_2 \prec o_1} \in \mathcal{P}_\prec$, or not. Assume that $\text{p}_{o_2 \prec o_1} \in \mathcal{P}_\prec$. Then, from Formula 4.8, $S(\text{p}_{o_2 \prec o_1}) = \mathit{false}$, and so $o_2 \not\prec' o_1$, and thus $\prec'$ is acyclic. Assume that $\text{p}_{o_2 \prec o_1} \notin \mathcal{P}_\prec$. Then, from Definition 4.9, $o_2 \not\prec' o_1$ and so $\prec'$ is acyclic. □

The following lemma establishes that if variable x is bound to constant c in a POCL-valid reinstantiated reorder of a POP, then $\text{p}_{x=c} \in \mathcal{P}_\theta$:

Lemma C.2. *Let $P = \langle O, \theta, \prec \rangle$ be a POCL-valid POP with POCL-valid reinstantiated reorder $Q = \langle O, \theta', \prec' \rangle$. Then, for each $x \in \mathsf{vars}(O)$ and $c \in \mathsf{consts}(O)$* s.t. *$\theta'(x) = c$, $\text{p}_{x=c} \in \mathcal{P}_\theta$.*

Proof. This proof assumes that every variable appears in an operator precondition, and constants only appear in the initial state. This results in no loss of generality as this can be achieved in any POP through the addition of "dummy" preconditions.

For every $t \in \mathsf{vars}(O) \cup \mathsf{consts}(O)$, a chain of causally-connected terms $\vec{k}_t$ can be defined as follows. If $t \in \mathsf{consts}(O)$, then $\vec{k}_t = \langle t \rangle$. If $t \in \mathsf{vars}(O)$, then $\vec{k}_t = \langle t \rangle^\frown \vec{k}_{t'}$[1], where t and t' are causally connected via an

[1]Where $\vec{x}^\frown\vec{y}$ indicates the concatenation of $\vec{x}$ and $\vec{y}$.

unthreatened causal link $\langle o_p, q(\vec{u}), o_c, q_1(\vec{s})\rangle$ *s.t.* there is an i where $\vec{u}[i] = t'$ and $\vec{s}[i] = t$. As Q is POCL-valid, such a t' must exist for every $t \in \mathsf{vars}(O)$, and $\theta'(t') = \theta'(t)$.

From Definition 4.6, for each $t \in \mathsf{vars}(O)$ and $s \in \vec{k}_t$, $\mathrm{p}_{t=s} \in \mathcal{P}_\theta$, and from the definition of $\vec{k}_t$, all elements must to the same constant, that is, $\theta'(t) = \theta'(s)$. As the last element in all such chains must be some $c \in \mathsf{consts}(o_I)$, it follows that for all $x \in \mathsf{vars}(O)$ *s.t.* $\theta'(x) = c$, $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$. □

The following lemma establishes that any POP corresponding to a solution to the MRR MaxSAT encoding as in Definition 4.9 has well-formed binding constraints:

Lemma C.3. *Let $P = \langle O, \theta, \prec\rangle$ be a POP, S be a solution to the* MRR MaxSAT *encoding of P, and* $\mathrm{POP}(S) = \langle O, \theta', \prec'\rangle$ *be the POP corresponding to S. Then, θ' is ground and complete* w.r.t. $\mathsf{vars}(O)$.

Proof. It suffices to prove that *(i)* if $x \in \mathsf{vars}(O)$ then there is a c *s.t.* $S(\mathrm{p}_{x=c}) = \mathit{true}$, and *(ii)* if $x \in \mathsf{vars}(O)$, $c_1, c_2 \in \mathsf{consts}(O)$ and $S(\mathrm{p}_{x=c_1}) = S(\mathrm{p}_{x=c_2}) = \mathit{true}$, then $c_1 = c_2$.

First, *(i)* is proved. Let $x \in \mathsf{vars}(O)$. From Formula 4.13, not all propositions $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$ *s.t.* $c \in \mathsf{consts}(O)$ can be set to *false*. It suffices to show that there is at least one $c \in \mathsf{consts}(O)$ *s.t.* $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$. Let $c = \theta'(x)$. Then, from Lemma C.2, $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$. Proof of *(ii)* follows directly from Formula 4.13. □

Lemma C.4 establishes that any POP corresponding to a solution to the MRR MaxSAT encoding as in Definition 4.9 is POCL-valid:

Lemma C.4. *Let $P = \langle O, \theta, \prec\rangle$ be a POP, S be a solution to the* MRR MaxSAT *encoding of P, and* $\mathrm{POP}(S) = \langle O, \theta', \prec'\rangle$ *be the POP corresponding to S. Then,* $\mathrm{POP}(S)$ *is POCL-valid.*

Proof. Let o_c and $q(\vec{s})$ be an operator and literal *s.t.* $\mathsf{cons}(o_c, q(\vec{s}))$. To prove that $\mathrm{POP}(S)$ is POCL-valid it suffices to show that o_c and $q(\vec{s})$ are supported by an unthreatened causal link.

Formula 4.14 states that not all $\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})} \in \mathcal{P}_\rightarrow$ can be set to *false*. It first must be shown that there is at least one such $\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})} \in \mathcal{P}_\rightarrow$.

As P is POCL-valid, each o_c and $q(\vec{s})$ *s.t.* $\mathsf{cons}(o_c, q(\vec{s}))$ are supported by an unthreatened causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s})\rangle$, and so from Definition 4.6, $\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})} \in \mathcal{P}_\rightarrow$. Therefore, there is at least one $\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})} \in \mathcal{P}_\rightarrow$.

Let $\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})} \in \mathcal{P}_\rightarrow$ and $S(\mathrm{p}_{o_p,q(\vec{u}),o_c,q(\vec{s})}) = \mathit{true}$. Then, from Definition 4.6 and Formula 4.15, $\mathrm{p}_{o_p \prec o_c} \in \mathcal{P}_\prec$ and $S(\mathrm{p}_{o_p \prec o_c}) = \mathit{true}$, and for $1 \leq i \leq |\vec{s}|$, $\mathrm{p}_{\vec{u}[i]=\vec{s}[i]} \in \mathcal{P}_\theta$ and $S(\mathrm{p}_{\vec{u}[i]=\vec{s}[i]}) = \mathit{true}$. Therefore, $o_p \prec' o_c$, $\theta'(\vec{u}) = \theta'(\vec{s})$, that is, o_c and $q(\vec{s})$ are supported by a causal link.

Let o_t and $q(\vec{v})$ be a operator and literal *s.t.* $\mathsf{thrtns}(o_t, q(\vec{v}))$. Then, from Definition 4.6, $\mathrm{p}_{o_t \prec o_p}, \mathrm{p}_{o_c \prec o_t} \in \mathcal{P}_\prec$ and for $1 \leq i \leq |\vec{s}|$, $\mathrm{p}_{\vec{p}[i]=\vec{v}[i]} \in \mathcal{P}_\theta$. And, from Formula 4.15, either $S(\mathrm{p}_{o_t \prec o_p}) = \mathit{true}$, $S(\mathrm{p}_{o_c \prec o_t}) = \mathit{true}$, or there is some $1 \leq i \leq |\vec{s}|$ *s.t.* $S(\mathrm{p}_{\vec{s}[i]=\vec{v}[i]}) = \mathit{false}$. Therefore, either $o_t \prec' o_p$, $o_c \prec' o_t$ or $\theta'(\vec{v}) \neq \theta'(\vec{s})$, that is, the causal link is unthreatened. □

Lemma C.5 establishes that any POCL-valid reinstantiated reorder Q of a POP P corresponds to a solution to the MRR MaxSAT encoding of P. The exact form of this solution is defined as follows:

Definition C.1. *Let $P = \langle O, \theta, \prec \rangle$ be a POP and $Q = \langle O, \theta', \prec' \rangle$ be a reinstantiated reorder of P. Then, S_Q denotes the transformation of Q into a solution to the* MRR MaxSAT *encoding of P* s.t.*:*

- *for all $\mathrm{p}_{o_1 \prec o_2} \in \mathcal{P}_\prec$, $S_Q(\mathrm{p}_{o_1 \prec o_2}) =$ true iff $o_1 \prec' o_2$,*
- *for all $\mathrm{p}_{t=u} \in \mathcal{P}_\theta$, $S_Q(\mathrm{p}_{t=u}) =$ true iff $\theta'(t) = \theta'(u)$, and*
- *for all $\mathrm{p}_{o_p,\vec{u},o_c,\vec{s}} \in \mathcal{P}_\rightarrow$, $S_Q(\mathrm{p}_{o_p,\vec{u},o_c,\vec{s}}) =$ true iff precondition $q(\vec{s})$ of o_c is supported by postcondition $q(\vec{u})$ of o_p and the causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle$ is unthreatened in Q.*

Lemma C.5 assumes that Q is *POCL-minimal*, as defined below:

Definition C.2. *A POP $P = \langle O, \theta, \prec \rangle$ is* **POCL-minimal** iff *it is POCL-valid, and for all $Q = \langle O, \theta, \prec' \rangle$ s.t. $\prec' \subset \prec$, Q is not POCL-valid.*

This assumption results in no loss of generality as any POCL-valid POP can be made POCL-minimal by greedily removing ordering constraints.

Lemma C.5. *Let $P = \langle O, \theta, \prec \rangle$ be a POP, and $Q = \langle O, \theta', \prec' \rangle$ be a POCL-minimal reinstantiated reorder of P. Then S_Q is a solution to the* MRR MaxSAT *encoding of P.*

Proof. As Q is POCL-minimal, it follows that every ordering constraint is required to ensure that a consumer is supported by a producer, a causal link is protected from threats, or that $\prec'$ is transitively closed. Therefore, from Definitions 4.5 and 4.6, if $o_1 \prec' o_2$ then $\mathrm{p}_{o_1 \prec o_2} \in \mathcal{P}_\prec$. As Q is a well-formed POP, and $S_Q(\mathrm{p}_{o_1 \prec o_2}) = \mathit{true}$ *iff* $o_1 \prec' o_2$ (Definition C.1), S satisfies Formulae 4.8–4.10.

Formula 4.11 requires that for all $\mathrm{p}_{s=u}, \mathrm{p}_{s=u} \in \mathcal{P}_\theta$, $S_Q(\mathrm{p}_{s=u}) = S_Q(\mathrm{p}_{u=s})$, which follows directly from Definition C.1. Formula 4.12 requires that for all $\mathrm{p}_{s=u}, \mathrm{p}_{u=v}, \mathrm{p}_{s=v} \in \mathcal{P}_\theta$, if $S_Q(\mathrm{p}_{s=u}) = S_Q(\mathrm{p}_{u=v}) = \mathit{true}$ then $S_Q(\mathrm{p}_{s=v}) = \mathit{true}$. From Definition C.1, $S_Q(\mathrm{p}_{s=u}) = S_Q(\mathrm{p}_{u=v}) = \mathit{true}$ *iff* $\theta'(s) = \theta'(u) = \theta'(v)$, which implies $S_Q(\mathrm{p}_{s=v}) = \mathit{true}$, as desired. Formula 4.13 requires that for each variable $x \in \mathsf{vars}(O)$ there is exactly one constant $c \in \mathsf{consts}(O)$ *s.t.* $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$ and $S_Q(\mathrm{p}_{x=c}) = \mathit{true}$. As θ' is ground, there is exactly one c *s.t.* $\theta'(x) = c$. From Lemma C.2, $\mathrm{p}_{x=c} \in \mathcal{P}_\theta$, and from Definition C.1, $S_Q(\mathrm{p}_{x=c}) = \mathit{true}$, as desired.

Let $q(\vec{s})$ be a precondition of $o_c \in O$. As Q is POCL-valid, $q(\vec{s})$ is supported by a postcondition $q(\vec{u})$ of $o_p \in O$ and causal link $\langle o_p, q(\vec{u}), o_c, q(\vec{s}) \rangle$ is unthreatened in Q. From Definitions 4.6 and C.1, $\mathrm{p}_{o_p,\vec{u},o_c,\vec{s}} \in \mathcal{P}_\rightarrow$ and $S_Q(\mathrm{p}_{o_p,\vec{u},o_c,\vec{s}}) = \mathit{true}$, thus satisfying Formula 4.14. From Definition 4.6, $\mathrm{p}_{o_p \prec o_c} \in \mathcal{P}_\prec$, and for $1 \leq i \leq |\vec{s}|$, $\mathrm{p}_{\vec{s}[i]=\vec{u}[i]} \in \mathcal{P}_\theta$. As $o_p \prec o_c$ and $\theta'(\vec{s}) = \theta'(\vec{u})$, then from Definition C.1, $S_Q(\mathrm{p}_{o_p \prec o_c}) = \mathit{true}$ and for $1 \leq i \leq |\vec{s}|$, $S_Q(\mathrm{p}_{\vec{s}[i]=\vec{u}[i]}) = \mathit{true}$. Therefore, the first line of Formula 4.15 is satisfied. Let postcondition $\neg q(\vec{v})$ of operator o_t be a threat to the above link. Then, from Definition 4.6, $\mathrm{p}_{o_t \prec o_p}, \mathrm{p}_{o_c \prec o_t} \in \mathcal{P}_\prec$, and for $1 \leq i \leq |\vec{s}|$, $\mathrm{p}_{\vec{v}[i]=\vec{u}[i]} \in \mathcal{P}_\theta$. As either $o_t \prec o_p$, $o_c \prec o_t$ or $\theta'(\vec{s}) \neq \theta'(\vec{v})$, then from Definition C.1, either $S_Q(\mathrm{p}_{o_t \prec o_p}) = \mathit{true}$, $S_Q(\mathrm{p}_{o_c \prec o_t}) = \mathit{true}$, or there is some $1 \leq i \leq |\vec{u}|$ *s.t.* $S_Q(\mathrm{p}_{\vec{u}[i]=\vec{v}[i]}) = \mathit{false}$. Thus, the second line of Formula 4.15 is satisfied. □

Proof of Theorem 4.3 follows from the above lemmas:

Theorem 4.3. MRD/MRR Correctness *Let $P = \langle O, \theta, \prec \rangle$ be a POP and S be an optimal solution to the* MRD/MRR MaxSAT *encoding. Then,* $\text{POP}(S) = \langle O, \theta', \prec' \rangle$ *is a POCL-valid reinstantiated de/reorder of P, and there is no $R = \langle O, \theta'', \prec'' \rangle$ that is a POCL-valid reinstantiated de/reorder of P where $\prec'' \subset \prec'$.*

Proof. Let $Q = \text{POP}(S)$. From Lemmas C.1–C.4, Q is a "well-formed" POCL-valid POP. As Q has the same operator set as P, then from Definition 4.1, Q is a reinstantiated de/reorder of P.

Proof of minimality is by *reductio*. Let $R = \langle O, \theta'', \prec'' \rangle$ be a POCL-valid POP *s.t.* $\prec'' \subset \prec'$. From Lemma C.5, there is some solution S' *s.t.* $\text{POP}(S') = R$. As $\prec'' \subset \prec'$, then from Definition 4.9 $\{\text{p} : \text{p} \in \mathcal{P}_{\prec}, S'(\text{p}) = \mathit{true}\} \subset \{\text{p} : \text{p} \in \mathcal{P}_{\prec}, S(\text{p}) = \mathit{true}\}$. As this contradicts the assumption that S is optimal (Definition 4.9), either R is not POCL-valid, or $\prec'' \not\subset \prec'$. Therefore, Q is minimal. □

C.4 Proof of Theorem 4.4

Theorem 4.4 states that the automorphisms of a POP's plan description graph (PDG) are symmetries of the POP, that is, if P is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}$ is a symmetry of P.

The proof will be split into two sections. The first section will show that the automorphisms of a POP's PDG are also automorphisms of the POP's problem description. Lemmas C.7–C.9 show that $\hat{\sigma}$ leaves the initial state unchanged, Lemma C.10 shows that it leaves the goal unchanged, and Lemmas C.11–C.13 show that it leaves the other operators unchanged. Lemma C.14 follows directly from these, and shows that the entire problem description is left unchanged.

The second section will show that the automorphisms preserve the validity and well-formed-ness of the POP. Lemma C.18 shows that after applying $\hat{\sigma}$, the POP's variable binding constraints remain ground and complete *w.r.t.* to the operators, and Lemma C.19 shows that the transitivity and irreflexivity of the POP's precedence relation is preserved. Lemma C.20 shows that $\hat{\sigma}$ preserves validity.

The following observation follows directly from the definition of an automorphism (Definition 2.16), and will be used throughout the proofs:

Observation C.1. *Let $G = V, E$ be a graph, and $\sigma \in \mathsf{Aut}(G)$. If $(v_1, v_2) \in E$ and $\sigma(v_1) = v_3$, then $(v_3, \sigma(v_2)) \in E$.*

The following lemma, which will also be used throughout the proofs, shows that if a function f that maps every $x \in X$ both to and from another element in X, then f is a permutation over X:

Lemma C.6. *Let $f : S \times S$ be a function and $X \subset S$ be set* s.t. *if $x \in X$ then* (i) *$f(x) \in X$, and* (ii) *there is a $y \in X$* s.t. *$f(y) = x$. Then f is a permutation over X.*

Proof. It suffices to show that f is surjective and injective over X. Surjectivity requires that for all $x \in X$, there is a $y \in X$ *s.t.* $f(y) = x$,

which follows directly from *(ii)*. Injectivity requires that if $x, y \in X$ and $f(x) = f(y)$ then $x = y$. This follows from surjectivity and the pigeonhole principle, and can be shown by *reductio*. Assume there is an $x, y \in X$ *s.t.* $f(x) = f(y)$ and $x \neq y$. As every element of X is mapped to another element in X (assumption *(i)*), and x and y map to the same element, then there must be a $z \in X$ for which there is no $z' \in X$ *s.t.* $f(z') = z$. As this contradicts surjectivity, f must be injective over X. □

The next three lemmas show that the initial state is left unchanged. Lemma C.7 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ maps every $q(\vec{c}) \in \mathsf{post}(o_I)$ to some $q(\vec{d}) \in \mathsf{post}(o_I)$:

Lemma C.7. *Let P be a POP, $q(\vec{c}) \in \mathsf{post}(o_I)$ and $\sigma \in \mathsf{Aut}(G_P)$. Then, there is some $q(\vec{d}) \in \mathsf{post}(o_I)$* s.t. *$\hat{\sigma}(q(\vec{c})) = q(\vec{d})$.*

Proof. Let $\vec{c} = \langle c_1, \ldots, c_n \rangle$. From Definition 4.14, it suffices to show that there is some $\vec{d} = \langle d_1, \ldots, d_n \rangle$ *s.t.* $q(\vec{d}) \in \mathsf{post}(o_I)$ and for $1 \leq i \leq n$, $\sigma(\mathrm{v}_{c_i}) = \mathrm{v}_{d_i}$.

From Definition 4.13, the chain of vertices $\mathrm{v}^1_{q(\vec{c})}, \ldots, \mathrm{v}^n_{q(\vec{c})}$ represents the parameters of $q(\vec{c})$. First, it is shown that there is some $q(\vec{d}) \in \mathsf{post}(o_I)$ *s.t.* for $1 \leq i \leq n$, $\sigma(\mathrm{v}^i_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$. The proof is inductive. The base cases show that there is some $q(\vec{d}) \in \mathsf{post}(o_I)$ such that $\sigma(\mathrm{v}^1_{q(\vec{c})}) = \mathrm{v}^1_{q(\vec{d})}$ and $\sigma(\mathrm{v}^2_{q(\vec{c})}) = \mathrm{v}^2_{q(\vec{d})}$. The inductive case shows that as if $\sigma(\mathrm{v}^{i-1}_{q(\vec{c})}) = \mathrm{v}^{i-1}_{q(\vec{d})}$ and $\sigma(\mathrm{v}^i_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$ hold, then so must $\sigma(\mathrm{v}^{i+1}_{q(\vec{c})}) = \mathrm{v}^{i+1}_{q(\vec{d})}$.

First, the base cases are shown for $i = 1$ and $i = 2$. Let $\mathrm{v}^1_{q(\vec{c})} \in V$ be a vertex representing the first parameter of $q(\vec{c})$. From Definition 2.16, σ must permute $\mathrm{v}^1_{q(\vec{c})}$ to a vertex of the same colour, so from Definition 4.13, $\sigma(\mathrm{v}^1_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$, where $\mathrm{v}^i_{q(\vec{d})}$ represents the ith parameter of some $q(\vec{d}) \in \mathsf{post}(o_I) \cup \mathsf{pre}(o_G)$.

Since $\sigma(\mathrm{v}^1_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$ and $(\mathrm{v}^1_{q(\vec{c})}, \mathrm{v}^{post}_{o_I}) \in E$, it follows from Observation C.1 that $(\mathrm{v}^i_{q(\vec{d})}, \sigma(\mathrm{v}^{post}_{o_I})) \in E$. As $\mathrm{v}^{post}_{o_I}$ has a unique colour, $\sigma(\mathrm{v}^{post}_{o_I}) = \mathrm{v}^{post}_{o_I}$ and so $(\mathrm{v}^i_{q(\vec{d})}, \mathrm{v}^{post}_{o_I}) \in E$. So, from Definition 4.13 it follows that $i = 1$ and $q(\vec{d}) \in \mathsf{post}(o_I)$. Since $\sigma(\mathrm{v}^1_{q(\vec{c})}) = \mathrm{v}^1_{q(\vec{d})}$ and $(\mathrm{v}^1_{q(\vec{c})}, \mathrm{v}^2_{q(\vec{c})}) \in E$, it follows from Observation C.1 that $(\mathrm{v}^1_{q(\vec{d})}, \sigma(\mathrm{v}^2_{q(\vec{c})})) \in E$. As $\mathrm{v}^1_{q(\vec{d})}$ is only linked to $\mathrm{v}^{post}_{o_I}$ and $\mathrm{v}^2_{q(\vec{d})}$, and $C(\mathrm{v}^2_{q(\vec{c})}) \neq C(\mathrm{v}^{post}_{o_I})$, it follows that $\sigma(\mathrm{v}^2_{q(\vec{c})}) = \mathrm{v}^2_{q(\vec{d})}$.

Next, the inductive case is shown for $i > 2$. Let $\mathrm{v}^1_{q(\vec{c})}, \ldots, \mathrm{v}^n_{q(\vec{c})}$ and $\mathrm{v}^1_{q(\vec{d})}, \ldots, \mathrm{v}^n_{q(\vec{d})}$ be two chains of vertices representing $q(\vec{c})$ and $q(\vec{d})$, respectively, let $2 \leq i \leq n$, and let $\sigma(\mathrm{v}^{i-1}_{q(\vec{c})}) = \mathrm{v}^{i-1}_{q(\vec{d})}$ and $\sigma(\mathrm{v}^i_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$. Since $\sigma(\mathrm{v}^i_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$ and $(\mathrm{v}^i_{q(\vec{c})}, \mathrm{v}^{i+1}_{q(\vec{c})}) \in E$, then from Observation C.1, $(\mathrm{v}^i_{q(\vec{d})}, \sigma(\mathrm{v}^{i+1}_{q(\vec{c})})) \in E$. As $i > 2$, from Definition 4.13 $\mathrm{v}^i_{q(\vec{d})}$ is only connected to $\mathrm{v}^{i-1}_{q(\vec{d})}$ and $\mathrm{v}^{i+1}_{q(\vec{d})}$. However, as $\sigma(\mathrm{v}^{i-1}_{q(\vec{c})}) = \mathrm{v}^{i-1}_{q(\vec{d})}$ and $\mathrm{v}^{i-1}_{q(\vec{c})} \neq \mathrm{v}^{i+1}_{q(\vec{c})}$, it follows that $\sigma(\mathrm{v}^{i+1}_{q(\vec{c})}) = \mathrm{v}^{i+1}_{q(\vec{d})}$.

From Definition 4.13, for $1 \leq i \leq n$, each parameter vertex $\mathrm{v}^i_{q(\vec{c})}$ is connected to the vertex representing that parameter's binding, that is,

$(\mathrm{v}^i_{q(\vec{c})}, \mathrm{v}_{c_i}) \in E$. Let $\vec{d} = \langle d_1, \ldots, d_n \rangle$. As $q(\vec{d}) \in \mathsf{post}(o_I)$, then similarly, for each $1 \leq i \leq n$, $(\mathrm{v}^i_{q(\vec{d})}, \mathrm{v}_{d_i}) \in E$. From Definition 2.16, σ must map each v_{c_i} to a vertex v *s.t.* $C(v) = C(\mathrm{v}_{c_i})$. And, because $(\mathrm{v}^i_{q(\vec{c})}, \mathrm{v}_{c_i}) \in E$ and $\sigma(\mathrm{v}^i_{q(\vec{c})}) = \mathrm{v}^i_{q(\vec{d})}$, it must also hold that $(\mathrm{v}^i_{q(\vec{d})}, \mathrm{v}) \in E$. From Definition 4.13, the only vertex that both has the same colour as v_{c_i} and is connected to $\mathrm{v}^i_{q(\vec{d})}$ is v_{d_i}, and so it follows that for $\sigma(\mathrm{v}_{c_i}) = \mathrm{v}_{d_i}$, as desired. □

Lemma C.8 shows that if $\sigma \in \mathsf{Aut}(G_P)$ and $q(\vec{c}) \in \mathsf{post}(o_I)$, then there is some $q(\vec{d}) \in \mathsf{post}(o_I)$ that $\hat{\sigma}$ maps to $q(\vec{c})$:

Lemma C.8. *Let P be a POP, $q(\vec{c}) \in \mathsf{post}(o_I)$ and $\sigma \in \mathsf{Aut}(G_P)$. Then, there is some $q(\vec{d}) \in \mathsf{post}(o_I)$* s.t. *$\hat{\sigma}(q(\vec{d})) = q(\vec{q})$.*

Proof. The proof is a trivial modification of the proof of Lemma C.7. Let $\vec{c} = \langle c_1, \ldots, c_n \rangle$. From Definition 4.14, it suffices to show that there is some $\vec{d} = \langle d_1, \ldots, d_n \rangle$ *s.t.* $q(\vec{d}) \in \mathsf{post}(o_I)$ and for $1 \leq i \leq n$, $\sigma(\mathrm{v}_{d_i}) = \mathrm{v}_{c_i}$. First it is shown inductively that σ maps the parameter vertices $\mathrm{v}^1_{q(\vec{d})}, \ldots, \mathrm{v}^n_{q(\vec{d})}$ to $\mathrm{v}^1_{q(\vec{c})}, \ldots, \mathrm{v}^n_{q(\vec{c})}$. The base cases show that for $i = 1$ and $i = 2$, there is some $q(\vec{d}) \in \mathsf{post}(O)$ such that $\sigma(\mathrm{v}^1_{q(\vec{d})}) = \mathrm{v}^1_{q(\vec{c})}$ and $\sigma(\mathrm{v}^2_{q(\vec{d})}) = \mathrm{v}^2_{q(\vec{c})}$. The inductive case shows that for $i > 2$, if there is some $q(\vec{d})) \in \mathsf{post}(O)$ such that $\sigma(\mathrm{v}^{i-1}_{q(\vec{d})}) = \mathrm{v}^{i-1}_{q(\vec{c})}$ and $\sigma(\mathrm{v}^i_{q(\vec{d})}) = \mathrm{v}^i_{q(\vec{c})}$ then $\sigma(\mathrm{v}^{i+1}_{q(\vec{d})}) = \mathrm{v}^{i+1}_{q(\vec{c})}$. Finally, it is shown that for $1 \leq i \leq n$, as $(\mathrm{v}^i_{q(\vec{d})}, \mathrm{v}_{d_i}), (\mathrm{v}^i_{q(\vec{c})}, \mathrm{v}_{c_i}) \in E$ then $\sigma(\mathrm{v}_{d_i}) = \mathrm{v}_{c_i}$, as desired. □

Lemma C.9 uses the above two results to show that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ leaves o_I unchanged:

Lemma C.9. *If P is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}(o_I) = o_I$.*

Proof. It suffices to show that $\hat{\sigma}$ leaves $\mathsf{post}(o_I)$ unchanged. Lemmas C.7 and C.8 show that if $q(\vec{c}) \in \mathsf{post}(o_I)$ then $\hat{\sigma}(q(\vec{c})) \in \mathsf{post}(o_I)$ and there is a $q(\vec{d}) \in \mathsf{post}(o_I)$ *s.t.* $\hat{\sigma}(q(\vec{d})) = q(\vec{c})$. So, from Lemma C.6, $\hat{\sigma}$ is a permutation over $\mathsf{post}(o_I)$, and as $\mathsf{post}(o_I)$ is a set, it is unchanged by $\hat{\sigma}$. □

The proofs for Lemmas C.7, C.8 and C.9 above can be modified to show that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ leaves o_G unchanged:

Lemma C.10. *If P is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}(o_G) = o_G$.*

Proof. A trivial modification of the proofs of Lemmas C.7, C.8 and C.9. □

Lemma C.11 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ maps every $o(\vec{x}) \in O$ to some $o(\vec{y}) \in O$:

Lemma C.11. *Let $P = \langle O, \theta, \prec \rangle$ be a POP, $o(\vec{x}) \in O \setminus \{o_I, o_G\}$ and $\sigma \in \mathsf{Aut}(G_P)$. Then, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$* s.t. *$\mathsf{name}(o(\vec{x})) = \mathsf{name}(o(\vec{y}))$ and $\hat{\sigma}(o(\vec{x})) = o(\vec{y})$.*

Proof. From Definition 4.14, it suffices to show that for all $o(\vec{x}) \in O \setminus \{o_I, o_G\}$, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$ such that for $1 \leq i \leq |\vec{x}|$, $\sigma(\mathrm{v}_{x_i}) = \mathrm{v}_{y_i}$. The proof is inductive. The base cases show that for all $o(\vec{x}) \in O \setminus \{o_I, o_G\}$, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$ such that $\sigma(\mathrm{v}_{x_1}) = \mathrm{v}_{y_1}$ and $\sigma(\mathrm{v}_{x_2}) = \mathrm{v}_{y_2}$. The inductive case shows that if $\sigma(\mathrm{v}_{x_{i-1}}) = \mathrm{v}_{y_{i-1}}$ and $\sigma(\mathrm{v}_{x_i}) = \mathrm{v}_{y_i}$ hold, then so must $\sigma(\mathrm{v}_{x_{i+1}}) = \mathrm{v}_{y_{i+1}}$.

First, the base cases are shown for $i = 1$ and $i = 2$. Let $\mathrm{v}^{pre}_{o(\vec{x})}$ and $\mathrm{v}^{post}_{o(\vec{x})}$ be vertices representing operator $o(\vec{x})$'s pre/postcondition sets. From Definition 4.13, $\mathrm{v}^{pre}_{o(\vec{x})}$ and $\mathrm{v}^{post}_{o(\vec{x})}$ must be mapped to vertices $\mathrm{v}^{pre}_{o(\vec{y})}$ and $\mathrm{v}^{post}_{o(\vec{y})}$ representing the pre/postcondition sets of an operator $o(\vec{y})$ *s.t.* $\mathsf{name}(o(\vec{x})) = \mathsf{name}(o(\vec{y}))$. Since $\sigma(\mathrm{v}^{pre}_{o(\vec{x})}) = \mathrm{v}^{pre}_{o(\vec{y})}$ and $(\mathrm{v}^{pre}_{o(\vec{x})}, \mathrm{v}_{x_1}) \in E$, then from Observation C.1 it follows that $(\mathrm{v}^{pre}_{o(\vec{y})}, \sigma(\mathrm{v}_{x_1})) \in E$. Further, since $\sigma(\mathrm{v}^{post}_{o(\vec{x})}) = \mathrm{v}^{post}_{o(\vec{y})}$ and $(\mathrm{v}^{post}_{o(\vec{x})}, \mathrm{v}_{x_1}) \in E$, then from Observation C.1 it follows that $(\mathrm{v}^{post}_{o(\vec{y})}, \sigma(\mathrm{v}_{x_1})) \in E$. Because $\mathrm{v}^{pre}_{o(\vec{y})}$ and $\mathrm{v}^{post}_{o(\vec{y})}$ are only connected to one vertex, v_{y_1}, it follows that $\sigma(\mathrm{v}_{x_1}) = \mathrm{v}_{y_1}$. Since $\sigma(\mathrm{v}_{x_1}) = \mathrm{v}_{y_1}$ and $(\mathrm{v}_{x_1}, \mathrm{v}_{x_2})$, then from Observation C.1 it follows that $(\mathrm{v}_{y_1}, \sigma(\mathrm{v}_{x_2})) \in E$. As v_{y_1} is only linked to $\mathrm{v}^{pre}_{o(\vec{y})}$, $\mathrm{v}^{post}_{o(\vec{y})}$ and v_{y_2}, and $C(\mathrm{v}^{pre}_{o(\vec{y})}) \neq C(\mathrm{v}_{x_2})$ and $C(\mathrm{v}^{post}_{o(\vec{y})}) \neq C(\mathrm{v}_{x_2})$, it follows that $\sigma(\mathrm{v}_{x_2}) = \mathrm{v}_{y_2}$.

Next, the inductive case is shown for $i > 2$. Let $\mathrm{v}_{x_1}, \ldots, \mathrm{v}_{x_n}$ and $\mathrm{v}_{y_1}, \ldots, \mathrm{v}_{y_n}$ be two chains of vertices representing $\vec{x}$ and $\vec{y}$, respectively, let $2 \leq i \leq n$, and let $\sigma(\mathrm{v}_{x_{i-1}}) = \mathrm{v}_{y_{i-1}}$ and $\sigma(\mathrm{v}_{x_i}) = \mathrm{v}_{y_i}$. Since $\sigma(\mathrm{v}_{x_i}) = \mathrm{v}_{y_i}$ and $(\mathrm{v}_{x_i}, \mathrm{v}_{x_{i+1}}) \in E$, then from Observation C.1, $(\mathrm{v}_{y_i}, \sigma(\mathrm{v}_{x_{i+1}})) \in E$. As $i > 2$, from Definition 4.13 v_{y_i} is only connected to $\mathrm{v}_{y_{i-1}}$ and $\mathrm{v}_{y_{i+1}}$. However, as $\sigma(\mathrm{v}_{x_{i-1}}) = \mathrm{v}_{y_{i-1}}$ and $\mathrm{v}_{x_{i-1}} \neq \mathrm{v}_{x_{i-1}}$, it follows that $\sigma(\mathrm{v}_{x_{i+1}}) = \mathrm{v}_{y_{i+1}}$. □

Lemma C.12 shows that if $\sigma \in \mathsf{Aut}(G_P)$, and $o(\vec{x}) \in O$, then there is some $o(\vec{y}) \in O$ *s.t.* $\mathsf{name}(o(\vec{x})) = \mathsf{name}(o(\vec{y}))$, and $\hat{\sigma}(o(\vec{y})) = o(\vec{x})$:

Lemma C.12. *Let $P = \langle O, \theta, \prec \rangle$ be a POP, $o(\vec{x}) \in O \setminus \{o_I, o_G\}$ and $\sigma \in \mathsf{Aut}(G_P)$. Then, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$* s.t. *$\hat{\sigma}(o(\vec{y})) = o(\vec{x})$.*

Proof. The proof is a trivial modification of the proof of Lemma C.11. From Definition 4.14, it suffices to show that for all $o(\vec{x}) \in O \setminus \{o_I, o_G\}$, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$ such that $\mathsf{name}(o(\vec{x})) = \mathsf{name}(o(\vec{x}))$, and for $1 \leq i \leq |\vec{x}|$, $\sigma(\mathrm{v}_{y_i}) = \mathrm{v}_{x_i}$. The base cases show that for all $o(\vec{x}) \in O \setminus \{o_I, o_G\}$, there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$ such that $\sigma(\mathrm{v}_{y_1}) = \mathrm{v}_{x_1}$ and $\sigma(\mathrm{v}_{y_2}) = \mathrm{v}_{x_2}$. The inductive case shows that if $\sigma(\mathrm{v}_{y_{i-1}}) = \mathrm{v}_{x_{i-1}}$ and $\sigma(\mathrm{v}_{y_i}) = \mathrm{v}_{x_i}$ hold, then so must $\sigma(\mathrm{v}_{y_{i+1}}) = \mathrm{v}_{x_{i+1}}$. □

Lemma C.13 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ is an automorphism of $O \setminus \{o_I, o_G\}$:

Lemma C.13. *Let $P = \langle O, \theta, \prec \rangle$ be a POP and $\sigma \in \mathsf{Aut}(G_P)$. Then, $\hat{\sigma}(O \setminus \{o_I, o_G\}) = O \setminus \{o_I, o_G\}$.*

Proof. Lemmas C.11 and C.12 show that if $o(\vec{x}) \in O \setminus \{o_I, o_G\}$ then $\hat{\sigma}(o(\vec{x})) \in O \setminus \{o_I, o_G\}$, and there is some $o(\vec{y}) \in O \setminus \{o_I, o_G\}$ *s.t.* $\hat{\sigma}(o(\vec{y})) = o(\vec{x})$. Then, from Lemma C.6, $\hat{\sigma}$ is a permutation over $O \setminus \{o_I, o_G\}$. Thus, as $O \setminus \{o_I, o_G\}$ is a set, $\hat{\sigma}(O \setminus \{o_I, o_G\}) = O \setminus \{o_I, o_G\}$. □

Lemma C.14 establishes that if $\sigma \in \mathsf{Aut}(G_P)$, then $\hat{\sigma}$ is an automorphism of O:

Lemma C.14. *If $P = \langle O, \theta, \prec \rangle$ is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}(O) = O$.*

Proof. As $\hat{\sigma}(o_I) = o_I$ (Lemma C.9), $\hat{\sigma}(o_G) = o_G$ (Lemma C.10) and $\hat{\sigma}(O \setminus \{o_I, o_G\}) = O \setminus \{o_I, o_G\}$ (Lemma C.13), it follows that $\hat{\sigma}(O) = O$. □

The next section of the proof shows that if P is a POP and $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to any reinstantiated de/reorder of P preserves its validity and optimality. The proofs make use of several lemmas.

Lemma C.15 shows that if $\sigma \in \mathsf{Aut}(P)$ then $\hat{\sigma}$ is a permutation over the variables and constants in O:

Lemma C.15. *Let $P = \langle O, \theta, \prec \rangle$ be a POP and $\sigma \in \mathsf{Aut}(G_P)$. Then, $\hat{\sigma}(\mathsf{vars}(O)) = \mathsf{vars}(O)$ and $\hat{\sigma}(\mathsf{consts}(O)) = \mathsf{consts}(O)$.*

Proof. From Definition 4.13, it follows that if $x \in \mathsf{vars}(O)$, then there is some $y \in \mathsf{vars}(O)$ *s.t.* $\sigma(\mathrm{v}_x) = \mathrm{v}_y$. Therefore, from Definition 4.14, if $x \in \mathsf{vars}(O)$ then $\hat{\sigma}(x) \in \mathsf{vars}(O)$, and there is some $y \in \mathsf{vars}(O)$ such that $\hat{\sigma}(y) = x$. Thus, from Lemma C.6, $\hat{\sigma}$ is a permutation over $\mathsf{vars}(O)$. The above proof can be trivially modified to show the same for $\mathsf{consts}(O)$. □

Lemma C.16 follows from Definition 4.14, and shows that if $\sigma \in \mathsf{Aut}(P)$ then $\hat{\sigma}$ preserves the equality relation over the POP's variables:

Lemma C.16. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = \langle O, \theta', \prec' \rangle$. Then for all $x, y \in \mathsf{vars}(O)$, $\theta(x) = \theta(y)$ iff $\theta'(\hat{\sigma}(x)) = \theta'(\hat{\sigma}(y))$.*

Proof. First, it is shown that if $\theta(x) = \theta(y)$ then $\theta'(\hat{\sigma}(x)) = \theta'(\hat{\sigma}(y))$. If $\theta(x) = \theta(y)$ then there is a c such that $\{x\backslash c, y\backslash c\} \subseteq \theta$. As $\hat{\sigma}(\theta) = \theta'$, it follows from the definition of a permutation in Section 2.1.7.2 that $\{\hat{\sigma}(x)\backslash\hat{\sigma}(c), \hat{\sigma}(y)\backslash\hat{\sigma}(c)\} \subseteq \theta'$, and so $\theta'(\hat{\sigma}(x)) = \theta'(\hat{\sigma}(y))$. Next it is shown that if $\theta(x) \neq \theta(y)$ then $\theta'(\hat{\sigma}(x)) \neq \theta'(\hat{\sigma}(y))$. If $\theta(x) \neq \theta(y)$ then there is a $c \neq d$ such that $\{x\backslash c, y\backslash d\} \subseteq \theta$. As $\hat{\sigma}(\theta) = \theta'$, it follows that $\{\hat{\sigma}(x)\backslash\hat{\sigma}(c), \hat{\sigma}(y)\backslash\hat{\sigma}(d)\} \subseteq \theta'$, and so $\theta'(\hat{\sigma}(x)) \neq \theta'(\hat{\sigma}(y))$. □

Lemma C.17 follows directly from the definition of a permutation (Section 2.1.7.2), and makes clear some consequences of applying $\hat{\sigma}$ to $\prec$:

Lemma C.17. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = \langle O, \theta', \prec' \rangle$. Then:*

- *if $o_1 \prec o_2$ iff $\hat{\sigma}(o_1) \prec' \hat{\sigma}(o_2)$, and*
- *if $o'_1 \prec' o'_2$ then there is some o_1, o_2 such that $\hat{\sigma}(o_1) = o'_1$, $\hat{\sigma}(o_2) = o'_2$ and $o_1 \prec o_2$.*

Proof. Follows directly from the definition of a permutation (Section 2.1.7.2), which states that a permutation σ can be applied to a logical structure η by simultaneously replacing every term $t \in \eta$ with $\sigma(t)$ while maintaining the structure, or type, of η. Thus, if $\prec = \{(o_1, o_2), \ldots, (o_{n-1}, o_n)\}$ then $\prec' = \{(\hat{\sigma}(o_1), \hat{\sigma}(o_2)), \ldots, (\hat{\sigma}(o_{n-1}), \hat{\sigma}(o_n))\}$. □

The next two lemmas show that if $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to a reinstantiated reorder of P results in a well-formed POP. First, Lemma C.18 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to θ results in a ground substitution that is complete *w.r.t.* to O:

Lemma C.18. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = \langle O, \theta', \prec' \rangle$. Then, $\mathsf{domain}(\theta') = \mathsf{vars}(O)$ and for all $x \in \mathsf{domain}(\theta')$, $\theta'(x) \in \mathsf{consts}(O)$.*

Proof. Let $\theta = \{x_1 \backslash c_1, \ldots, x_n \backslash c_n\}$. As θ is complete *w.r.t.* to O, $\mathsf{vars}(O) = \{x_1, \ldots, x_n\}$. As $\theta' = \hat{\sigma}(\theta)$, $\theta' = \{\hat{\sigma}(x_1) \backslash \hat{\sigma}(c_1), \ldots, \hat{\sigma}(x_n) \backslash \hat{\sigma}(c_n)\}$. As $\hat{\sigma}$ is a permutation over $\mathsf{vars}(O)$ (Lemma C.15), it follows that $\{\sigma(x_1), \ldots, \sigma(x_n)\} = \{x_1, \ldots, x_n\} = \mathsf{vars}(O)$, and so $\mathsf{domain}(\theta') = \mathsf{vars}(O)$.

Since $\{c_1, \ldots, c_n\} \subseteq \mathsf{consts}(O)$ and $\hat{\sigma}$ is a permutation over $\mathsf{consts}(O)$ (Lemma C.15), it follows that $\{\sigma(c_1), \ldots, \sigma(c_n)\} \subseteq \mathsf{consts}(O)$. Thus, for all $x \in \mathsf{vars}(\theta')$, $\theta'(x) \in \mathsf{consts}(O)$. □

Lemma C.19 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to $\prec$ results in a set of transitively closed, acyclic ordering constraints:

Lemma C.19. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = \langle O, \theta', \prec' \rangle$. Then, $\prec' = \prec'^{+}$ and for all $o \in O$, $o \not\prec' o$.*

Proof. Irreflexivity is proved by *reductio*. Assume there is an $o' \in O$ *s.t.* $o' \prec' o'$. It follows that there must be some o_1 and o_2 such that $o_1 \prec o_2$ and $\hat{\sigma}(o_1) = \hat{\sigma}(o_2) = o'$ (Lemma C.17). As $\hat{\sigma}$ is a permutation over O (Lemma C.14), it follows that $o_1 = o_2$, and so $o_1 \prec o_1$ which contradicts the assumption that $\prec$ is irreflexive. So, $\prec'$ must be irreflexive.

Next, transitivity is shown. Let $o'_1 \prec' o'_2$, $o'_2 \prec' o'_3$. Then there must be an $o_1, o_2, o_3 \in O$ such that $\hat{\sigma}(o_1) = o'_1$, $\hat{\sigma}(o_2) = o'_2$, $\hat{\sigma}(o_3) = o'_3$, $o_1 \prec o_2$ and $o_2 \prec o_3$ (Lemma C.17). As $\prec$ is transitive, it follows that $o_1 \prec o_3$, and so $o'_1 \prec' o'_3$ (Lemma C.17), and so $\prec'$ is transitive. □

Lemma C.20 shows if $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to a reinstantiated reorder of P preserves the validity of the POP. For the sake of brevity, the proof only demonstrates that POCL-validity is preserved. Proving full MTC-validity is a trivial but verbose extension that incorporates the possibility of a causal link being re-established by a "white knight".

Lemma C.20. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = Q' = \langle O, \theta', \prec' \rangle$ be the image of Q under $\hat{\sigma}$. Then, Q is valid* iff *Q' is valid.*

Proof. First it is shown that if Q is POCL-valid, then Q' is POCL-valid. Assume Q is POCL-valid. To show that Q' is POCL-valid, it suffices to show that under $\prec'$ and θ', every precondition of every operator is supported by an unthreatened causal link. Let o'_c and $p(\vec{s}')$ be an operator and a literal such that $\mathsf{cons}(o'_c, p(\vec{s}'))$. There is therefore an o_c such that $\hat{\sigma}(o_c) = o'_c$ (Lemma C.12), and therefore a $p(\vec{s}) \in \mathsf{pre}(o_c)$ such that $\hat{\sigma}(\vec{s}) = \vec{s}'$. As Q is POCL-valid, there is an o_p and $p(\vec{u})$ such that $\mathsf{prods}(o_p, p(\vec{u}))$, $o_p \prec o_c$ and $\theta(\vec{s}) = \theta(\vec{u})$ (Definition 2.7). Let $\hat{\sigma}(o_p) = o'_p$. From Lemma C.13, there is a $p(\vec{u}') \in \mathsf{post}(o'_p)$. As $o_p \prec o_c$, it follows that $o'_p \prec' o'_c$ (Lemma C.17), and

as $\theta(\vec{s}) = \theta(\vec{u})$ it follows that $\theta'(\vec{s}') = \theta'(\vec{u}')$ (Lemma C.16). Thus, o'_c and $p(\vec{s}')$ are supported by a causal link to o'_p and $p(\vec{u}')$.

Let o'_t and $p(\vec{v}')$ be an operator and a literal such that $\mathsf{thrtns}(o'_t, p(\vec{v}'))$. There must be an o_t such that $\hat{\sigma}(o_t) = o'_t$ (Lemma C.12), and therefore a $p(\vec{v}) \in \mathsf{post}(o_t)$ such that $\mathsf{thrtns}(o_t, p(\vec{v}))$. As Q is POCL-valid, it follows that either $o_t \prec o_p$, $o_c \prec o_t$ or $\theta(\vec{s}) \neq \theta(\vec{v})$ (Definition 2.7). If $o_t \prec o_p$ then $o'_t \prec' o'_p$, if $o_c \prec o_t$ then $o'_c \prec' o'_t$ (Lemma C.17) and if $\theta(\vec{s}) \neq \theta(\vec{v})$, then $\theta'(\vec{s}') \neq \theta'(\vec{v}')$ (Lemma C.16). And so the causal link is unthreatened, and so Q' is POCL-valid.

Proof that if Q' is POCL-valid then Q is POCL-valid is a trivial modification of the above. If o_c and $p(\vec{s})$ are an operator and literal *s.t.* $\mathsf{cons}(o_c, p(\vec{s}))$, then, from Lemma C.11, they are mapped by $\hat{\sigma}$ to some o'_c and $p(\vec{s}')$. As Q' is POCL-valid, o'_c and $p(\vec{s}')$ are supported in Q' by an unthreatened causal link to some o'_p and $p(\vec{u}')$ *s.t.* $\mathsf{prods}(o'_p, p(\vec{u}'))$. Also, from Lemma C.11, $\hat{\sigma}$ must map some o_p and $p(\vec{u})$ to o'_p and $p(\vec{u}')$ *s.t.* $\mathsf{prods}(o_p, p(\vec{u}))$. And, from Lemmas C.17 and C.16, o_c and $p(\vec{s})$ must, in Q, be supported by an unthreatened causal link to o_p and $p(\vec{u})$. □

Lemma C.21 shows that if $\sigma \in \mathsf{Aut}(G_P)$, then applying $\hat{\sigma}$ to P results in an equally optimal POP, that is, the size of $\prec$ is unaffected by $\hat{\sigma}$:

Lemma C.21. *Let P be a POP, $Q = \langle O, \theta, \prec \rangle$ be a reinstantiated reorder of P, $\sigma \in \mathsf{Aut}(G_P)$ and $\hat{\sigma}(Q) = \langle O, \theta', \prec' \rangle$. Then, $| \prec | = | \prec' |$.*

Proof. As $\hat{\sigma}(\prec) = \prec'$, for every $(o_1, o_2) \in \prec'$ there is some $(o_3, o_4) \in \prec$ *s.t.* $\hat{\sigma}(o_3) = o_1$ and $\hat{\sigma}(o_4) = o_2$. Therefore, $\prec'$ cannot have more elements than $\prec$, that is, $| \prec | \leq | \prec' |$.

Proof that $| \prec | \not< | \prec' |$ is by *reductio*. If $| \prec | < | \prec' |$, then $\hat{\sigma}$ must map at least two distinct elements of $\prec$ to the same element of $\prec'$. Assume that there is an $(o_1, o_2) \in \prec'$ and $(o_3, o_4) \neq (o_5, o_6) \in \prec$ *s.t.* $\hat{\sigma}(o_3) = \hat{\sigma}(o_5) = o_1$ and $\hat{\sigma}(o_4) = \hat{\sigma}(o_6) = o_2$. However, as $\hat{\sigma}$ is a permutation over O (Lemma C.14), $\hat{\sigma}(o_3) = \hat{\sigma}(o_5)$ *iff* $o_3 = o_5$ and $\hat{\sigma}(o_4) = \hat{\sigma}(o_6)$ *iff* $o_4 = o_6$, which contradicts the assumption that $(o_3, o_4) \neq (o_5, o_6)$. Therefore, $| \prec | \not< | \prec' |$, so $| \prec | = | \prec' |$. □

Theorem 4.4. Plan Description Graph *If P is a POP and $\sigma \in \mathsf{Aut}(G_P)$ then $\hat{\sigma}$ is a symmetry of P.*

Proof. Follows from Lemmas C.14, C.20 and C.21. □

APPENDIX D

Proofs of Theorems in Chapter 5

Proofs of Theorems 5.1 and 5.2 will make use of the definition below, which translates any given formula of propositional logic into an equisatisfiable constraint formula expressed in language $\mathcal{L}_C$ (Definition 5.1). The first line creates a constraint formula by substituting every proposition p appearing in φ with the constraint atom $x_p = 1$. The second line adds the requirement that all variables must be bound to either 0 or 1:

Definition D.1. *For any Boolean formula φ constructed from n propositional variables $\mathsf{vars}(\varphi) = \{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$ and the usual logical connectives ($\wedge$, $\vee$ and $\neg$), $\mathcal{L}_{\mathrm{C}}(\varphi)$ denotes its translation into a constraint formula:*

$$\mathcal{L}_{\mathrm{C}}(\varphi) \stackrel{\text{def}}{=} \varphi\{\mathrm{p}_i \backslash (x_i = 1) : \mathrm{p}_i \in \mathsf{vars}(\varphi)\} \wedge \bigwedge_{\mathrm{p}_i \in \mathsf{vars}(\varphi)} (x_i = 0 \vee x_i = 1).$$

For example, if φ is $(\mathrm{p}_1 \vee \mathrm{p}_2) \wedge (\neg\, \mathrm{p}_1 \vee \neg\, \mathrm{p}_2)$ then $\mathcal{L}_{\mathrm{C}}(\varphi)$ is $(x_1 = 1 \vee x_2 = 1) \wedge (x_1 \neq 1 \vee x_2 \neq 1) \wedge (x_1 = 0 \vee x_1 = 1) \wedge (x_2 = 0 \vee x_2 = 1)$. It follows from this definition that φ is satisfiable *iff* there exists some substitution θ such that $\mathcal{L}_{\mathrm{C}}(\varphi)\theta$ is ground and evaluates to *true*.

Proofs of Theorems 5.1 and 5.2 also refer to the following lemma:

Lemma D.1. *Let $P = \langle O, \phi \rangle$ be a partial plan and $\vec{a} = \vec{o}\theta$ be a classical plan. Whether or not $\vec{a}$ is an instantiation of P can be determined in polynomial time.*

Proof. First check, in linear time, that $o \in O$ *iff* $o \in \vec{o}$. Whether $\vec{a} \models \phi$ can be determined by recursively applying the rules in Definition 5.2. As atoms are base cases, it follows that if ϕ contains n atoms this will take at most $2n - 1$ steps. □

D.1 Proof of Theorem 5.1

Theorem 5.1. Partial Plan Soundness. *Determining the soundness of a partial plan is* co-NP*-complete.*

Proof. A partial plan P is sound *iff* all of its instantiations are valid. To show membership in co-NP, guess a classical plan $\vec{a}$, then verify in polynomial time that it is not valid and is an instantiation of P (Lemma D.1).

Proof of co-NP-hardness is by reduction from the co-NP-complete problem UNSAT, which asks whether a given Boolean formula φ is unsatisfiable. First, construct the constraint formula $\phi = \mathcal{L}_{\mathrm{C}}(\varphi)$ (Definition D.1). Let $\mathsf{vars}(\phi) = \{x_1, \ldots, x_n\}$. Next, construct the partial plan $P = \langle O, \phi \rangle$, where the operator set O comprises three operators o_I, $o_1(x_1, \ldots, x_n)$ and o_G, none of which have any postconditions, and only o_G has preconditions – a single atom, g. Because neither o_I nor o_1 have any postconditions, no instantiation of P can be valid, as the preconditions of o_G will never hold. Therefore, P is sound *iff* it has no instantiations, that is, *iff* ϕ is unsatisfiable. As ϕ and φ are equisatisfiable, it follows that P is sound *iff* φ is unsatisfiable, as desired. □

D.2 Proof of Theorem 5.2

Theorem 5.2. PARTIAL PLAN SATISFIABILITY. *Determining the satisfiability of a partial plan is* NP*-complete.*

Proof. A partial plan $P = \langle O, \phi \rangle$, is satisfiable *iff* it admits at least one instantiation. To prove membership in NP, guess a classical plan $\vec{a}$ and verify in polynomial time that it is an instantiation of P (Lemma D.1).

Proof of NP-hardness is by reduction from SAT, which asks whether a given a Boolean formula φ is satisfiable. First, construct the constraint formula $\phi = \mathcal{L}_{\mathrm{C}}(\varphi)$ (Definition D.1). Let $\mathsf{vars}(\phi) = \{x_1, \ldots, x_n\}$. Next, construct the partial plan $P = \langle O, \phi \rangle$, where O comprises the three operators $\{o_I, o_1(x_1, \ldots, x_n), o_G\}$, none of which have any preconditions or postconditions. A classical plan will be an instantiation of P *iff* it is of the form $\langle o_I, o_1, o_G \rangle \theta$ where the formula $\phi\theta$ is ground and evaluates to *true*. Thus, P is satisfiable *iff* ϕ is satisfiable. As ϕ and φ are equisatisfiable, it follows that P is satisfiable *iff* φ is satisfiable. □

D.3 Proof of Theorem 5.3

Theorem 5.3. PARTIAL PLAN VALIDITY. *Determining the validity of a partial plan is* DP*-complete.*

Proof. A language L is in DP *iff* there are two languages $L_1 \in$ NP and $L_2 \in$ co-NP such that $L = L_1 \cap L_2$. Let language $L_{SAT} = \{P : P \text{ is satisfiable}\}$ and $L_{SND} = \{P : P \text{ is sound}\}$, where P is a partial plan. From Definition 5.5, the language of partial plan validity is $L_{VAL} = L_{SAT} \cap L_{SND}$. Since $L_{SND} \in$ co-NP (Theorem 5.1) and $L_{SAT} \in$ NP (Theorem 5.2), it follows that $L_{VAL} \in$ DP.

DP-hardness shown by reduction from the DP-complete problem SAT-UNSAT [97], which asks, given two Boolean formulae φ_1 and φ_2, whether φ_1 is satisfiable and φ_2 is unsatisfiable.

From φ_1 and φ_2 construct the constraint formulae $\phi_1 = \mathcal{L}_{\mathrm{C}}(\varphi_1)$ and $\phi_2 = \mathcal{L}_{\mathrm{C}}(\varphi_2)$, where $\mathsf{vars}(\phi_1) = \{x_1, \ldots, x_n\}$ and $\mathsf{vars}(\phi_2) = \{y_1, \ldots, y_m\}$. Next, construct the partial plan $P = \langle O, \phi \rangle$. Operator set O contains four

elements: o_I, which has no preconditions or postconditions, $o_1(x_1, \ldots, x_n)$, which has no preconditions and the postcondition $\neg g$, $o_2(y_1, \ldots, y_m)$, which has no preconditions and the postcondition g, and o_G, which has the precondition g and no postconditions. Constraint formula $\phi = \phi_1 \wedge (o_1 \prec o_2 \vee \phi_2)$.

Partial plan P will be valid *iff* it is satisfiable and sound, that is *iff* ϕ is satisfiable and $o_1 \prec o_2$ in all instantiations. Clearly if ϕ_1 is satisfiable and ϕ_2 is unsatisfiable, then P is valid. However, if ϕ_1 is unsatisfiable, then so is ϕ, and if ϕ_1 and ϕ_2 are both satisfiable then $o_1 \prec o_2$ need not hold and P is not sound. Therefore, P is valid *iff* ϕ_1 is satisfiable and ϕ_2 is unsatisfiable. As φ_1 and ϕ_1 are equisatisfiable, and φ_2 and ϕ_2 are equisatisfiable, it follows that P is valid *iff* φ_1 is satisfiable and φ_2 is unsatisfiable, as desired. □

D.4 Proof of Theorem 5.4

Theorem 5.4 says that determining the satisfiability of a partial plan with treewidth k is W[1]-hard and in XP when parameterised with k. A problem is in XP *iff* it is solvable in time $\mathcal{O}(n^{f(k)})$ [36] (Section 2.1.3). The first part of the proof will show membership in XP by demonstrating that any partial plan with treewidth k can, in time $\mathcal{O}(|P|^k)$, be translated into an equivalent constraint satisfaction problem (CSP, Definition 2.10) with the same treewidth and then solved.

A CSP is a triple $\langle X, D, C\rangle$ where X is a finite set of variables, D defines the variables' domains, and $C = \{C_1, \ldots, C_m\}$ defines the constraints, where each $C_j \in C$ is of the form $\langle t_j, R_j\rangle$ where $t_j \subseteq X$, the constraint scope, is a subset of k variables and R_j is a k-ary relation over the corresponding domains.

The *primal graph* of a CSP is an undirected graph $\langle V, E\rangle$ where $V = X$ and $(x_1, x_2) \in E$ *iff* there exists a constraint $\langle t_j, R_j\rangle \in C$ such that $x_1, x_2 \in t_j$. A CSP can be solved in time $\mathcal{O}(n^k)$ where k is equal to the treewidth (Section 2.1.4) of the primal graph of the CSP [47].

The definition below transforms partial plan $P = \{O, \phi\}$ into CSP S_P:

Definition D.2. *Let $P = \langle O, \phi\rangle$ be a partial plan where $O = \{o_1, \ldots, o_n\}$, $\phi = \phi_1 \wedge \cdots \wedge \phi_m$, and for each ϕ_i, $\vec{t_i} = \langle t_1^i, \ldots, t_k^i\rangle$ is an ordering of all variables and constants appearing in ϕ_i. Then, $S_P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ defines the* CSP s.t.*:*

- $X = \{v_x : x \in \mathsf{vars}(O)\} \cup \{v_o : o \in O\}$,
- $D = \{D_1, \ldots, D_{|X|}\}$, *where* $D_{v_{t_i}} = \mathsf{consts}(O)$ *if* $t_i \in \mathsf{vars}(O)$, *or* $\{1, \ldots, n\}$ *if* $t_i \in O$, *and*
- $C = \{\langle s_1, R_1\rangle, \ldots, \langle s_m, R_m\rangle\}$ *where each* $s_i = \langle v_{t_i^i}, \ldots, v_{t_k^i}\rangle$ *and* $\vec{c} = \langle c_1, \ldots, c_k\rangle \in R_i$ iff $\phi_i\theta$, *where θ is a substitution* s.t. $\theta(\vec{t_i}) = \vec{c}$, *evaluates to* true.

Variable set X contains a v_x for each variable in P, and a v_o for each operator. The domain of each v_x is $\mathsf{consts}(O)$, and the domain of each v_o is $\{1, \ldots, |O|\}$ (i.e., numeric plan step indexes). A constraint is constructed from each conjunct of the constraint formula ϕ. For each conjunct ϕ_i, an ordering over the variables and operators in ϕ_i is produced, $\vec{t_i}$. Then,

a constraint $C_i = \langle s_i, R_i \rangle$ is constructed, where s_i is a list containing all variables in X associated elements of $\vec{t}_i$, and R_i contains a tuple of constants $\vec{c}$ *iff* substituting each element of $\vec{t}_i$ with the corresponding element of $\vec{c}$ in ϕ_i results in a ground formula that evaluates to *true* (with precedence over indexes interpreted as expected).

A solution to S_P assigns a domain object to each variable in P, and an index (or time step) to each operator. While it is clear that if P is satisfiable then S_P is also, the constraints in S_P are *weaker* than ϕ. The problem is that S_P allows more than one operator to be assigned to the same index. For example, a constraint such as $o_p \not\prec o_q \wedge o_q \not\prec o_p$, is, due to the implicit constraint that a classical plan be totally ordered, unsatisfiable. However, S_P could satisfy it by setting the two operators to the same index.

An obvious fix, namely adding constraints to S_P to enforce a total order, can change S_P's treewidth and so is unsatisfactory. Instead, ϕ is strengthened with constraints that enforce a total order over *selected* operators. The constraints do not modify ϕ's treewidth or satisfiability. However, they do allow any operators bound to the same index to be separated with an arbitrary "tie-breaker". The strengthening is denoted $\hat{\phi}$ and is defined as follows:

Definition D.3. *If $\phi = \phi_1 \wedge \cdots \wedge \phi_n$ is a constraint formula then $\hat{\phi}$ denotes the following extension:*

$$\hat{\phi} \stackrel{\text{def}}{=} \phi \wedge \bigwedge_{1 \leq i \leq n} \bigwedge_{\substack{o_j, o_k \in \mathsf{vars}(\phi_i), \\ o_j \neq o_k}} o_j \prec o_k \vee o_k \prec o_j.$$

If two operators appear in the same clause in ϕ, then $\hat{\phi}$ has a total order constraint over them. Lemma D.2 shows that the additional clauses introduced by $\hat{\phi}$ do not change the formula's satisfiability or treewidth:

Lemma D.2. *Let $P = \langle O, \phi \rangle$ and $Q = \langle O, \hat{\phi} \rangle$ be partial plans. Then* (i) $\mathsf{treewidth}(P) = \mathsf{treewidth}(Q)$*, and* (ii) *$\vec{a}$ is an instantiation of P* iff *it is an instantiation of Q.*

Proof. A partial plan's primal graph connects two vertices *iff* their associated terms appear in the same clause (Definition 5.7). As $\hat{\phi}$ has an additional clause of the form $o_j \prec o_k \vee o_k \prec o_j$ *iff* o_j and o_k appeared in some clause in ϕ, it follows that the additional clauses will not modify the primal graph of ϕ. Therefore, $\mathsf{treewidth}(P) = \mathsf{treewidth}(Q)$.

As $\hat{\phi}$ has additional clauses enforcing a total order over selected pairs of operators, and classical plans are totally ordered, it is trivially true that if $\vec{a} \models \phi$ then $\vec{a} \models \hat{\phi}$, and *vice versa.* □

The following lemma shows that if $P = \langle O, \hat{\phi} \rangle$ is a partial plan with a strengthened constraint formula, then any solution to CSP S_P (Definition D.2) that sets operators to the same index can be linearised with an arbitrary "tie-breaker". It does this by demonstrating that if θ is a solution to S_P that sets o_p and o_q to the same index, then there is another solution, θ', that binds all variables to the same domain objects, and maintains the same relative ordering of operators with the exception that o_p is instead set to the index immediately before o_q:

Lemma D.3. *Let $P = \langle O, \hat{\phi} \rangle$ be a partial plan, $o_p, o_q \in O$ be two operators* s.t. *$o_p \neq o_q$, and θ be a solution to S_P* s.t. *$\theta(v_{o_p}) = \theta(v_{o_q})$. Then, there is a solution to S_P, θ'* s.t.

- *if $x \in \mathsf{vars}(O)$ then $\theta(v_x) = \theta'(v_x)$,*
- *if $\{o_1, o_2\} \subseteq O$ and $\{o_1, o_2\} \neq \{o_p, o_q\}$, then $\theta'(o_1) < \theta'(o_2)$* iff *$\theta(o_1) < \theta(o_2)$, and*
- *$\theta'(v_{o_p}) = \theta'(v_{o_q}) - 1$.*

Proof. First, construct the constraint formula ψ from $\hat{\phi}$ by replacing every $x \in \mathsf{vars}(O)$ with $\theta(x)$, and every atom of the form $o_1 \prec o_2$ *s.t.* $\{o_1, o_2\} \neq \{o_p, o_q\}$ with $(o_1 \prec o_2 \vee \bot)$ if $\theta(o_1) < \theta(o_2)$ or $(o_1 \prec o_2 \wedge \bot)$ otherwise. The formula strengthens $\hat{\phi}$ by fixing all variable bindings, and all ordering constraints except for those directly between o_p and o_q, to those in θ. Note that $\psi \models \hat{\phi}$, and if a clause in $\hat{\phi}$ contained a precedence atom, then that atom remains in the corresponding clause in ψ. It suffices to prove that these constraints do not rule out the possibility of ordering o_p directly before o_q, that is, that $\psi \not\models o_p \not\prec o_q$.

Proof is by *reductio*. Assume that $\psi \models o_p \not\prec o_q$. Therefore, there is a list of operators $\langle o_1, \ldots, o_k \rangle$ *s.t.* $o_p = o_1$, $o_q = o_k$ and for $1 \leq i < k$, $\psi \models o_i \not\prec o_{i+1}$, and there is no o' *s.t.* $\psi \models o_i \not\prec o' \wedge o' \not\prec o_{i+1}$. Because $\psi \models o_i \not\prec o_{i+1}$ directly, that is, without any transitive links, ψ must contain a precedence atom containing o_i and o_{i+1}. Therefore, so does $\hat{\phi}$, meaning that $\hat{\phi} \models (o_i \prec o_{i+1} \vee o_{i+1} \prec o_i)$ (Definition D.3). Because $\psi \models \hat{\phi}$, it follows that $\psi \models (o_i \prec o_{i+1} \vee o_{i+1} \prec o_i)$ also. And, as $\psi \models o_i \not\prec o_{i+1}$, it follows that $\psi \models o_{i+1} \prec o_i$. As this applies to all $1 \leq i < k$, it follows that $\psi \models o_k \prec o_1$, that is, $\psi \models o_q \prec o_p$, which contradicts the assumption that $\theta(o_p) = o(o_q)$. Therefore, $\psi \not\models o_p \not\prec o_q$, as desired. □

Proof of membership in XP uses the above definitions and lemmas:

Lemma D.4. *Determining the satisfiability of a partial plan P is in* XP *when parameterised with* $\mathsf{treewidth}(P)$.

Proof. Let $P = \langle O, \phi \rangle$ be a partial plan and $k = \mathsf{treewidth}(P)$ (Definition 5.7). From P, construct the partial plan $Q = \langle O, \hat{\phi} \rangle$ (with $\hat{\phi}$ defined as in Definition D.3). As $\hat{\phi}$ contains, at most, an extra clause for each clause in ϕ, each of size $2|O|$, Q is of size $\mathcal{O}(|P|^2)$. Next, construct the CSP S_Q (Definition D.2). Because Q has a treewidth of k (Lemma D.2), the number of variables or operators appearing in a clause is bounded by $k+1$, and so each constraint in S_Q contains up to $(|\mathsf{consts}(O)| + |O|)^{k+1}$ tuples. Thus, S_Q can be constructed in time $|Q|^{k+1}$, that is, $\mathcal{O}(|P|^k)$. As variables in S_Q appear in the same constraint scope *iff* they appear in the same clause of $\hat{\phi}$, the primal graph of S_Q has treewidth k, meaning that S_Q can be solved in $\mathcal{O}(|S_Q|^k)$. Because S_Q is of size $\mathcal{O}(|P|^k)$, S_Q can be constructed and solved in time $\mathcal{O}(|P|^{k^2})$.

A solution θ to S_Q will assign a constant to each variable, and an index, or time-step to each operator. First, any operators bound to the same index are separated with an arbitrary tie-breaker to produce valid solution θ' (Lemma D.3). Next, θ' is transformed into a classical plan $\vec{a}$, which,

from Definition D.2, is an instantiation of Q and therefore instantiation of P (Lemma D.2). Therefore, P can be transformed into a CSP and solved in time $\mathcal{O}(|P|^{k^2})$, and is in XP. □

The second part of the proof shows W[1]-hardness with an *fpt-reduction* (Definition 2.8) from k-Clique to partial plan satisfiability. Let an instance of k-Clique be a tuple (G, k) where G is a graph and $k > 0$, and let a partial plan satisfiability instance be a tuple (P, k') where P is a partial plan and k' is the treewidth of P. The fpt-reduction requires two functions, $f(k)$ and $g(k)$, and a constant c, such that any (G, k) can be transformed into a (P, k'), such that $k' \leq f(k)$ and (P, k') is computable from (G, k) in time $g(k)|G|^c$.

Lemma D.5. *Determining the satisfiability of a partial plan P is* W[1]*-hard when parameterised with* treewidth(P).

Proof. Proof is by fpt-reduction from k-Clique parameterised by k, which is W[1]-complete. Let $G = \langle V, E\rangle$ be an undirected graph such that $V = \{v_1, \ldots, v_n\}$ and $k > 0$. k-Clique asks if G contains a clique of k vertices.

Construct the partial plan $P = \langle O, \phi\rangle$ where the operator set $O = \{o_I, o_1(x_1, \ldots, x_k), o_G\}$, none of which have any preconditions or postconditions, and the constraint formula ϕ is defined such that:

$$\phi \stackrel{\text{def}}{=} \bigwedge_{i,j:1\leq i<j\leq k} \left(x_i \neq x_j \wedge \left(\bigvee_{\langle v,u\rangle\in E} x_i = v \wedge x_j = u\right)\right).$$

Constraint formula ϕ contains k variables. Every pair of variables must be bound to a different vertex, and moreover, every pair of variables must be bound to vertices that are connected in G. It is thus satisfiable *iff* G contains a clique of size k. Because every pair of variables in vars(ϕ) appear together in some clause, P's primal graph forms a clique of k vertices. Thus, if $k = 1$, P has a treewidth of 1, otherwise its treewidth is $k - 1$.

Let $f(k) = k$ and $g(k) = 1$. Therefore, from the k-Clique instance (G, k), the partial plan satisfiability instance (P, k'), where $k' \leq f(k)$, can be constructed in time $g(k).|G|^2$, meaning that partial plan satisfiability is W[1]-hard. □

Theorem 5.4. Parameterised Partial Plan Satisfiability. *Determining the satisfiability of a partial plan P is in* XP *and is* W[1]*-hard when parameterised with* treewidth(P).

Proof. Follows directly from Lemmas D.4 and D.5. □

D.5 Proof of Theorem 5.5

Theorem 5.5 states that partial plan is sound *iff* it satisfies the *modal truth criterion* (MTC). The MTC for a partial plan $P = \langle O, \phi\rangle$ requires that $\phi \models \text{MTC}(O)$ (Definition 5.9). Since it is established that a classical plan is valid *iff* the MTC holds (Theorem 7.3 in [8]), and a classical plan is a special case of a partial plan (Definition 5.6) the following observation holds:

Observation D.1. *A classical plan $\vec{a} = \langle o_1, \ldots, o_n\rangle\theta$ is valid* iff $\vec{a} \models \text{MTC}(\{o_1, \ldots, o_n\})$.

Theorem 5.5. PARTIAL PLAN MTC SOUNDNESS. *A partial plan* $P = \langle O, \phi \rangle$ *is sound* iff $\phi \models \text{MTC}(O)$.

Proof. Left to right is proved through a *reductio*. Assume that $P = \langle O, \phi \rangle$ is a sound partial plan that does not satisfy the MTC, that is, $\phi \not\models \text{MTC}(O)$. As $\phi \not\models \text{MTC}(O)$, there must be a classical plan $\vec{a} = \langle o_1, \ldots, o_n \rangle \theta$ such that $\vec{a} \models \phi$ but $\vec{a} \not\models \text{MTC}(\{o_1, \ldots, o_n\})$. Therefore, $\vec{a}$ is an instantiation of P, but is not valid (Observation D.1). As P has an invalid ground instantiation, it cannot be sound, contradicting the assumption. Therefore, if $P = \langle O, \phi \rangle$ is sound then $\phi \models \text{MTC}(O)$.

To prove right to left, let $P = \langle O, \phi \rangle$ be a partial plan such that $\phi \models \text{MTC}(O)$. Either P does, or does not, have any instantiations. If P has no instantiations, it is trivially sound. Let $\vec{a} = \langle o_1, \ldots, o_n \rangle \theta$ be an instantiation of P. Then it follows from Definition 5.4 that $\vec{a} \models \phi$. Thus, $\vec{a} \models \text{MTC}(\{o_1, \ldots, o_n\})$, and it follows from Observation D.1 that $\vec{a}$ is valid. Therefore, P is sound. □

D.6 Proof of Theorem 5.6

Theorem 5.6. MINIMUM RELAXATION. *A minimum relaxation of a valid partial plan can be found in polynomial time.*

Proof. From the input $P = \langle O, \phi \rangle$, produce in polynomial time the partial plan $Q = \langle O, \text{MTC}(O) \rangle$. The proof first establishes that Q is a relaxation of P. It follows from Theorem 5.5 that Q is sound. Because P is valid, and therefore sound, $\phi \models \text{MTC}(O)$ (Theorem 5.5). Since P is valid, ϕ must be satisfiable, and as $\phi \models \text{MTC}(O)$, $\text{MTC}(O)$ must be satisfiable also. As Q is satisfiable and sound, it is valid. As Q is valid and $\phi \models \text{MTC}(O)$, Q is a relaxation of P (Definition 5.8).

Proof that Q is minimally constrained is by *reductio*. Assume Q is not minimal, that is, that there exists an $R = \langle O, \phi' \rangle$ that is a proper relaxation of Q. As R is a proper relaxation of Q, it is valid (Definition 5.8), and $\text{MTC}(O) \models \phi'$ but $\phi' \not\models \text{MTC}(O)$. However, it follows from Theorem 5.5 that if $\phi' \not\models \text{MTC}(O)$ then R is not sound, and therefore not valid, contradicting the assumption. Therefore, no such R can exist, and Q is a minimally constrained relaxation of P. □

D.7 Proof of Theorem 5.7

Theorem 5.7. PROPER K-TREEWIDTH RELAXATION. *For any partial plan* P *and integer* $k > 0$*, deciding the existence of a proper* k*-treewidth relaxation of* P *is* W[1]*-hard when parameterised with* k*.*

Proof. Proof is by fpt-reduction from K-CLIQUE parameterised by k, which is W[1]-complete, to the problem of determining if a partial plan has a $(k+4)$-treewidth relaxation. Let $G = \langle V, E \rangle$ be an undirected graph such that $V = \{v_1, \ldots, v_n\}$ and $k > 0$. K-CLIQUE asks if G contains a clique of k vertices.

Construct the partial plan $P = \langle O, \phi \rangle$ where the operator set $O = \{o_I, o_1(x_1, \ldots, x_k), o_G\}$, none of which have any preconditions or postconditions, and the constraint formula $\phi = \neg\psi$, where ψ defined as follows:

$$\psi \stackrel{\text{def}}{=} \bigwedge_{i,j:1 \leq i < j \leq k} \left(x_i \neq x_j \wedge \left(\bigvee_{\langle v,u \rangle \in E} x_i = v \wedge x_j = u\right)\right).$$

Constraint formula ψ contains k variables. Every variable must be bound to a different vertex, and moreover, every pair of variables must be bound to vertices that are connected in G. Thus, ψ is satisfiable *iff* G contains a clique of size k. And, as $\phi = \neg\psi$, it follows that ϕ is a tautology *iff* G does not contain a clique of size k.

Let $k' = k+4$. As $|\mathsf{vars}(O)| = k$ and $|O| = 3$, $k' = |\mathsf{vars}(O)|+|O|+1$. A graph with n vertices cannot have a treewidth greater than $n+1$ (i.e., if $n \leq 1$, the treewidth is 1, otherwise a connected graph has treewidth $n-1$). Since a partial plan's primal graph has a vertex for each variable and operator, it follows that a partial plan's treewidth is bound by $|\mathsf{vars}(O)| + |O| + 1$. Therefore, all proper relaxations of P will have a treewidth $\leq k'$, and so a proper k'-treewidth relaxation of P exists *iff* a proper relaxation of P exists. The proof will continue by cases, and show that P has a proper relaxation *iff* G has a clique of k vertices.

First, assume that G does not have a clique of k vertices, that is, $\phi \equiv \top$. It follows that P has no proper relaxations, as there can be no $Q = \langle O, \phi' \rangle$ *s.t.* $\phi' \not\models \phi$.

Next, assume that G has a clique of k vertices. Construct the partial plan $Q = \langle O, \text{MTC}(O) \rangle$. From Theorem 5.5, Q is sound. Because there are no preconditions to any actions, the modal truth criterion for O is empty, that is, $\text{MTC}(O) \equiv \top$. Therefore, Q is satisfiable. As it is sound and satisfiable, it is valid. As $\phi \models \text{MTC}(O)$, Q is a relaxation of P. Q is proper relaxation of P *iff* $\top \not\models \phi$, that is, if ϕ is not a tautology. As it is assumed that G has a clique of k vertices, it follows that $\phi \not\equiv \top$ and so Q is proper relaxation of P.

From the cases above, it follows that P has a proper relaxation *iff* G has a clique of k vertices. And as all proper relaxations have treewidth $\leq k'$, it follows that P has a proper k'-treewidth relaxation *iff* G has a clique of k vertices, as desired.

Let $f(k) = k + 4$ and $g(k) = 1$. Therefore, from the k-Clique instance (G, k), the k'-treewidth partial plan relaxation instance (P, k'), where $k' \leq f(k)$, can be constructed in time $g(k).|G|^2$, meaning that k'-treewidth partial plan relaxation is W[1]-hard. □

D.8 Proof of Theorem 5.8

Theorem 5.8 shows that MkTR produces a minimal k-treewidth relaxation of its input in fpt-time. The proof refers to the F encoding in Definition 5.12, which transforms a CLP into partial plan by encoding its causal structure into a constraint formula. The F encoding is a strengthening of the MTC. The MTC requires that each consumer in a CLP $P = \langle O, L \rangle$ be supported by a causal link which is either unthreatened, or re-established by a "white knight", while F requires that the link also be selected from

L. Thus, $F(L) \models \text{MTC}(O)$. Further, if $Q = \langle O, L' \rangle$ is a CLP *s.t.* $L \subseteq L'$, then $F(L) \models F(L')$.

The proof also makes use of the following lemma, that uses Definition 5.12 to compare the treewidths of CLPs:

Lemma D.6. *If $P = \langle O, L \rangle$ and $Q = \langle O, L' \rangle$ are CLPs* s.t. *$L \subseteq L'$, then the treewidth of P is no greater than that of Q.*

Proof. Let $\phi_P = F(P)$ and $\phi_Q = F(Q)$. The constraint formulae created by the F encoding in Definition 5.12 have a clause for each consumer in the plan, and each clause contains a disjunct for each causal link that can support the consumer. As $L \subseteq L'$, it follows that each clause in ϕ_Q contains all the disjuncts in the corresponding clause in ϕ_P. As a primal graph links vertices *iff* their corresponding terms appear in the same clause (Definition 5.7), the primal graph of P is a subgraph of that of Q. And so, as shown by Bodlaender [16], P's treewidth is bounded by that of Q. □

Theorem 5.8. Minimal k-treewidth CS Relaxation. *Finding a minimal k-treewidth CS relaxation of a CLP is in* FPT *for parameter k.*

Proof. Consider the MkTR algorithm in Figure 5.3. Let $P = \langle O, L \rangle$ and integer $k > 0$ be the input to MkTR, and $Q = \langle O, L' \rangle$ be the output.

The proof first shows that Q is a k-treewidth relaxation of P. It is clear that at every stage of the MkTR loop, the treewidth of Q never exceeds k. Since $F(L) \models \text{MTC}(O)$, Q is sound. Since MkTR never removes links from L, it follows that $L \subseteq L'$, and from Definition 5.12, it follows that $F(L) \models F(L')$. As P is valid, it follows that $F(L)$ is satisfiable, and as $F(L) \models F(L')$ it follows that Q is also satisfiable. As Q is sound and satisfiable, it is valid. And as Q is valid and $F(L) \models F(L')$, it follows that Q is a k-treewidth relaxation of P.

Next, the minimality of Q is shown. Let A be the set of all causal links tested by MkTR, and $A' = A \setminus L'$ be the causal links that were rejected as they resulted in a treewidth greater than k. Q is minimal *iff* for all $S \subseteq A'$ the CLP $\langle O, L' \cup S \rangle$ has a treewidth greater than k. As expanding a causal structure cannot decrease its treewidth (Lemma D.6), it suffices to show that for $l \in A'$, the CLP $\langle O, L' \cup \{l\} \rangle$ has a treewidth greater than k.

Let $l_1, \ldots, l_n$ be the order in which links are selected from A, and $L_1, \ldots, L_n$ be the states of L before each iteration of the MkTR loop. Let $l_i \in A'$, and $P_i = \langle O, L_i \rangle$. As $l_i \in A'$, that is, l_i was not added, it follows that the treewidth of partial plan of $P'_i = \langle O, L_i \cup \{l_i\} \rangle$ is greater than k. Let $Q' = \langle O, L' \cup \{l_i\} \rangle$. As $L_i \cup \{l_i\} \subseteq L' \cup \{l_i\}$, it follows that the treewidth of Q' is also greater than k (Lemma D.6). There is therefore no relaxation of Q a treewidth $\leq k$, and so Q is minimal as desired.

□

D.9 Proof of Theorem 5.9

Theorem 5.9. Finite Domain POPI Satisfiability. *Let $P = \langle O, \prec, \Phi \rangle$ be a POPI such that* vars$(O) = \{x_1, \ldots, x_n\}$ *and let $D_{x_1}, \ldots, D_{x_n}$ be sets of constants representing variable domains. Deciding the existence of*

a classical plan $\vec{o}\theta$ *such that* (i) $\vec{o}\theta$ *is an instantiation of* P*, and* (ii) *for all* $x \in \mathsf{domain}(\theta)$*,* $\theta(x) \in D_x$ *is* NP*-complete.*

Proof. To show membership in NP, guess a classical plan $\vec{a} = \langle o_1, \ldots, o_n \rangle \theta$ and verify in polynomial time that *(i)* $O = \{o_1, \ldots, o_n\}$, *(ii)* $\prec^+ \subseteq \{o_i \prec o_j : 1 \leq i < j \leq n\}$, *(iii)* if $x \in \mathsf{domain}(\theta)$ then $\theta(x) \in D_x$, and *(iv)* $\vec{a} \models \Phi$.

Proof of NP-hardness is by reduction from 3-SAT, which asks whether a given 3-CNF Boolean formula φ constructed from propositional variables $\mathrm{p}_1, \ldots, \mathrm{p}_n$ is satisfiable. Assume that φ is of the form $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \cdots \wedge (l_m^1 \vee l_m^2 \vee l_m^3)$, where each literal l is either a propositional variable p or its negation, $\neg\, \mathrm{p}$.

First, construct the constraint formula Φ by replacing every instance of p_i in φ with $x_i \neq 0$ and every instance of $\neg\, \mathrm{p}_i$ with $x_i \neq 1$. As Φ is of the form $\Phi_1 \wedge \cdots \wedge \Phi_m$, where each Φ_i is a disjunction of negated constraint atoms, Φ is a syntactically valid POPI constraint formula.

Next, construct the POPI $P = \langle O, \Phi, \prec \rangle$ where operator set $O = \{o_I, o_1(x_1, \ldots, x_n), o_G\}$, none of which have any preconditions or postconditions, and $\prec = \{o_I \prec o_1, o_1 \prec o_G\}$. Finally, for each $x \in \mathsf{vars}(\Phi)$, construct the domain constraint $D_x = \{0, 1\}$. It follows that φ is satisfiable *iff* there exists a classical plan $\vec{a} = \vec{o}\theta$ such that *(i)* $\vec{a}$ is an instantiation of P, and *(ii)* for all $x \in \mathsf{domain}(\theta)$, $\theta(x) \in D_x$. □