Journal of Artificial Intelligence Research 57 (2016) 113-149 Submitted 03/16; published 09/16

Optimal Partial-Order Plan Relaxation via MaxSAT

Christian Muise cjmuise@cs.toronto.edu
Department of Computer Science,
Toronto, Ontario, Canada. M5S 3G4

J. Christopher Beck jcb@mie.utoronto.ca
Department of Mechanical & Industrial Engineering
Toronto, Ontario, Canada. M5S 3G8

Sheila A. McIlraith sheila@cs.toronto.edu
Department of Computer Science,
Toronto, Ontario, Canada. M5S 3G4

Abstract

Partial-order plans (POPs) are attractive because of their least-commitment nature, which provides enhanced plan flexibility at execution time relative to sequential plans. Current research on automated plan generation focuses on producing sequential plans, despite the appeal of POPs. In this paper we examine POP generation by relaxing or modifying the action orderings of a sequential plan to optimize for plan criteria that promote flexibility. Our approach relies on a novel partial weighted MaxSAT encoding of a sequential plan that supports the minimization of deordering or reordering of actions. Using a similar technique, we further demonstrate how to remove redundant actions from the plan, and how to combine this criterion with the objective of maximizing a POP's flexibility. Our partial weighted MaxSAT encoding allows us to compute a POP from a sequential plan effectively. We compare the efficiency of our approach to previous methods for POP generation via sequential-plan relaxation. Our results show that while an existing heuristic approach consistently produces the optimal deordering of a sequential plan, our approach has greater flexibility when we consider reordering the actions in the plan while also providing a guarantee of optimality. We also investigate and confirm the accuracy of the standard *flex* metric typically used to predict the true flexibility of a POP as measured by the number of linearizations it represents.

1. Introduction

For an agent to operate effectively in a dynamic world, its behaviour must be flexible in the face of unexpected changes. In the context of AI planning, there are several approaches to increase the flexibility of an agent, including giving it the option to select from different plans (Graham, Decker, & Mersic, 2001) or expanding the applicability of existing plans through plan generalization (Anderson & Farley, 1988). One example of the former is to delay committing to the ordering of certain actions in a plan until absolutely necessary, allowing the agent to dynamically choose how the plan proceeds at execution time (Veloso, Pollack, & Cox, 1998). This flexibility is precisely what *partial-order plans* provide.

Partial-order planning reflects a least commitment strategy (Weld, 1994). Unlike a sequential plan, which specifies a set of actions and a total order over those actions, an ideal partial-order plan (POP) only specifies those action orderings necessary to achieve the goal. In doing so, a POP embodies a family of sequential plans – a set of linearizations all sharing the same actions, but differing with respect to the order of the actions. During execution, the agent is free to choose the next action to execute from the plan as long as the chosen action has no preceding actions left to execute. Increasing the number of linearizations in a plan translates directly to giving the agent more freedom at execution time. Thus, we typically use the number of linearizations a POP has as a measure of its flexibility. While the measure of a POP's linearizations is not perfect, it is quite useful as a proxy for the plan's flexibility.

The flexibility afforded by POPs makes them attractive for real-time execution, multi-agent task assignment, and a range of other applications (Veloso et al., 1998; Weld, 1994). Nevertheless, in recent years research on plan generation has shifted away from partial-order planning towards sequential planning, primarily due to the effectiveness of heuristic-based forward-search planners. To regain the least commitment nature of POPs, while leveraging fast sequential plan generation, it is compelling to examine the computation of POPs via sequential planning technology as is done, for example, in the forward-chaining partial-order planner POPF (Coles, Coles, Fox, & Long, 2010).

In this paper, we present an alternative approach that first generates a sequential plan with a state-of-the-art planner, and subsequently relaxes the plan to a minimally constrained POP. Deordering is the process of removing ordering constraints from a plan and reordering is the process of allowing any arbitrary change to the ordering constraints; both requiring that the POP remains valid. POP deordering and reordering have been theoretically investigated (Bäckström, 1998), and unfortunately optimal deordering and reordering are NP-hard to compute and difficult to approximate within a constant factor (unless $NP \in DTIME(n^{poly\ log\ n})$, Bäckström, 1998). Despite this theoretical impediment, we find that in practice we can often compute an optimal solution.

The *minimum deordering* and *minimum reordering* of a sequential plan cover a natural aspect of least commitment planning – minimizing the ordering constraints placed on a plan. Intuitively, a deordering involves removing existing ordering constraints while a reordering allows both for ordering constraints to be removed as well as new ordering constraints to be included. Both techniques naturally provide greater flexibility at execution time, as there is an inverse correlation between the number of ordering constraints and the number of linearizations in a POP. Reordering may achieve greater flexibility than deordering as the addition of a new constraint may allow a number of existing ordering constraints to be removed while still guaranteeing POP validity.

Our approach for computing an optimally relaxed POP is to use a family of novel encodings for partial weighted MaxSAT: an optimal solution to the MaxSAT problem corresponds to an optimally relaxed POP. Unlike typical SAT-based planning techniques, we represent an action instance only once, giving us a succinct representation. We empirically compare our approach to an existing polynomial-time heuristic for relaxing a sequential plan due to Kambhampati and Kedar (1994) and find that the latter is extremely proficient at computing a minimum deordering, matching the optimal solution in every problem tested. We find, however, that a minimum reordering can be substantially more flexible than a minimum deordering, having fewer ordering constraints and far more linearizations.

We also compare the efficiency of our technique with a related approach that uses a Mixed Integer Linear Programming encoding to compute the minimum reordering (Do & Kambhampati, 2003) and find that our approach consistently performs better on problems of non-trivial size. Our approach represents a practical technique for computing a guaranteed optimal deordering and reordering of a POP.

Using a modern MaxSAT solver to compute maximally flexible solutions provides two key benefits: (1) the solver can be used as an any-time procedure that computes the optimally flexible reordering of a POP given enough time (where no such technique previously existed), and (2) computing the optimal deordering of a POP allows us to evaluate the efficiency of the existing heuristic algorithm.

1.1 Removing Redundant Actions

The generality of encoding allows us to easily define alternative objectives and optimization criteria. To demonstrate a key aspect of this generality, we extend the characterization to an orthogonal metric: minimizing the total cost of actions in a plan via the removal of unnecessary actions. This is a metric commonly used as a measure of plan quality, and interestingly can be directly at odds with the task of improving a plan's flexibility. Here, we consider one option for combining the two metrics that puts a higher priority on action cost than the subsequent plan flexibility.

The majority of the encodings, theoretical foundations, and theorems presented in this paper apply to the more general class of problems that incorporates both metrics. We refer to a POP that has a minimum action cost (over the actions in the POP), and subsequently a minimum number of ordering constraints, as a *minimum cost least commitment POP* (MCLCP).[1] An MCLCP is compelling because it is free of any redundant actions as well as any redundant ordering constraints: *an MCLCP contains only what is relevant to achieve the goal*.

We present the theoretical aspects of the more general MCLCP criterion, but as our main focus is on maximizing the flexibility of POPs, we focus our experimental evaluation on deorderings and reorderings exclusively. We leave the further evaluation of our method for solving MCLCP as compared to plan repair techniques (e.g., Nebel & Koehler, 1995; Gerevini & Serina, 2000) as a matter for future work.

1.2 Contributions

The following are the main contributions of this paper:

- We introduce a practical method for computing the optimal deordering and reordering of a plan. We accomplish this through a set of novel partial-weighted MaxSAT encodings, differing by a set of clause schema to define the type of relaxation we desire. We model the encodings after standard partial-order planning concepts, causal support and threat resolution, which we then draw upon to prove the correctness of our encodings.

- We propose an extension to least commitment planning, MCLCP, that includes the total cost of a solution. The optimization focuses first on minimizing the total action cost before minimizing the number of ordering constraints included in the plan. We

1. Note that minimizing the total action cost in a uniform cost domain is equivalent to minimizing the number of actions.

further prove the correctness of our approach that uses a partial weighted MaxSAT encoding for computing an MCLCP.

- We demonstrate, somewhat surprisingly, that an existing heuristic is extremely proficient at computing optimal deorderings. The existing algorithm produces only deorderings, and it is not theoretically guaranteed to find a minimal, let alone an optimal, deordering. Nonetheless, we find empirically that the heuristic computes the optimal deordering in every instance in our suite of benchmarks.
- We demonstrate the efficiency of our approach compared to a previous method that uses a similar encoding for a different optimization framework. For problems that are relatively difficult to relax (i.e., take more than a second to compute), our approach improves on the previous work by solving 22% more of the problems within the given time bound.
- We establish the empirical connection between the number of linearizations of a POP and the standard *flex* measure, which captures a normalized measure of the number of ordering constraints.
- We demonstrate the impact that the starting solution form will have on the final relaxed plan. In particular, we consider using two types of layered plans that are produced by the Mp and POPF planners (Rintanen, 2012; Coles et al., 2010).
- We show that we can achieve greater flexibility, compared to the optimal deordering, when using the optimal reordering. This result justifies the need for an approach such as ours to compute a more flexible plan.

The work in this paper extends the conference publications by Muise, McIlraith, and Beck (2011, 2012). While we provide the full generality of using MCLCP as our base encoding, in this paper we focus our evaluation on the minimum deordering and reordering aspects. We have expanded on the theoretical framework of our approach, including proofs of correctness, and significantly expanded the empirical evaluation.

1.3 Organization

We start by providing, in Section 2, the necessary background and notation for automated planning and partial weighted MaxSAT. Next, we detail our approach in Section 3, including both our new MCLCP criterion in Section 3.1 and the family of encodings for the various optimization criteria in Section 3.2. Finally, we present our evaluation in Section 4 and conclude with a discussion of related work and summary in Section 5.

2. Preliminaries

In this section, we present the necessary background notation and concepts for our work.

2.1 Classical Planning

Planning is the task of synthesizing a solution that dictates what actions an agent must take in order to achieve some prescribed goal. In classical planning, we assume the world is fully known and deterministic (Russell & Norvig, 2009). Classical planning has many applications that range from robotics to modelling biological processes (Ghallab, Nau, & Traverso, 2004). The standard approach for synthesizing a classical plan is to perform search through the state space of a problem, using heuristics to guide the planner towards a high quality solution. Here, we describe the most common formalism used for specifying a planning problem: STRIPS (Fikes, Hart, & Nilsson, 1972).

In STRIPS, a planning problem is a tuple $\Pi = \langle F, I, G, A\rangle$ where F is a finite set of fluents, $I \subseteq F$ is the initial state, and $G \subseteq F$ is the goal state, and A is the finite set of actions. We characterize an action $a \in A$ by the following three sets:

- $PRE(a)$: The fluents that must be true in order for a to be executable.
- $ADD(a)$: The fluents that action a adds to the state.
- $DEL(a)$: The fluents that action a deletes from the state.

For our work, the actions are instantaneous and we adopt the standard model of interleaved concurrency: no two actions can occur simultaneously. As such, we make the simplifying assumption that for every action a, $ADD(a) \cap DEL(a) = \emptyset$. This can be done without loss of generality, and simplifies the theoretical results below.

We say that an action a is *executable* in state s iff $PRE(a) \subseteq s$. The resulting state after executing action a in state s is defined as:

$$\mathcal{P}(s, a) \stackrel{\text{def}}{=} \begin{cases} (s \setminus DEL(a)) \cup ADD(a) & \text{if } a \text{ is executable in } s \\ \textit{undefined} & \text{otherwise} \end{cases}$$

For a planning problem $\Pi = \langle F, I, G, A\rangle$, we associate a cost function c_Π that maps every action $a \in A$ to a non-negative real number: $c_\Pi : A \to \mathbb{R}_0^+$.

We will make use of two further items of notation with respect to a set of actions A:

- **adders**(f): The set of actions in A that add the fluent f:

$$\{a \mid a \in A \text{ and } f \in ADD(a)\}$$

- **deleters**(f): The set of actions in A that delete the fluent f:

$$\{a \mid a \in A \text{ and } f \in DEL(a)\}$$

The most common representation of a solution to a planning problem is a sequential plan. A sequence of actions $\vec{a} = [a_1, \cdots, a_n]$ is *executable* if the preconditions of each action in the sequence are true in the corresponding state, and an executable sequence of actions is a *sequential plan* for the problem $\Pi = \langle F, I, G, A\rangle$ if executing the actions in $\vec{a}$ in sequence, when starting in state I, causes the goal to hold in the final state:

$$G \subseteq \mathcal{P}(\mathcal{P}(\cdots \mathcal{P}(I, a_1) \cdots, a_{n-1}), a_n)$$

For readability, we abbreviate the progression of a sequential plan $\vec{a}$ from state s as $\mathcal{P}^*(s, \vec{a})$. The cost of an action sequence $\vec{a} = [a_1, \ldots, a_n]$ is the sum of the individual actions costs:

$$c_\Pi(\vec{a}) = \sum_{i=1}^{n} c_\Pi(a_i)$$

Rather than impose a total order on the actions in a plan, a partial-order plan (POP) specifies a set of ordering constraints over the actions. We define a POP with respect to a planning problem Π as a tuple $\langle \mathcal{A}, \mathcal{O} \rangle$ where $\mathcal{A}$ is the set of actions in the plan and $\mathcal{O}$ is a set of ordering constraints between the actions in $\mathcal{A}$ (Russell & Norvig, 2009). While the same action may appear more than once in $\mathcal{A}$, we assume that every element of $\mathcal{A}$ is uniquely identifiable. For the actions $a_1, a_2 \in \mathcal{A}$, we denote the ordering constraint between a_1 and a_2 as $(a_1 \prec a_2) \in \mathcal{O}$ and interpret the constraint as "action a_1 appears before action a_2 in the plan". A total ordering of the actions in $\mathcal{A}$ that respects $\mathcal{O}$ is a *linearization*. A POP provides a compact representation for multiple linearizations. We assume the ordering constraints $\mathcal{O}$ are transitively closed:

$$\forall a_1, a_2, a_3 \in \mathcal{A}, (a_1 \prec a_2) \wedge (a_2 \prec a_3) \rightarrow (a_1 \prec a_3)$$

Assuming that $\mathcal{O}$ is transitively closed does not change the fundamental structure of the POP – the set of linearizations remains the same – but it allows us to effectively compare the flexibility of two POPs that share the same action set.

Similar to the cost of an action sequence, the cost of a POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$ is the sum of the action costs for the actions in P:

$$c_\Pi(P) = \sum_{a \in \mathcal{A}} c_\Pi(a)$$

To simplify the exposition that follows, we designate two actions in the POP that represent the initial state and goal state: a_I and a_G respectively. a_I is ordered before every other action, and a_G is analogously ordered after every other action. For a planning problem $\Pi = \langle F, I, G, A \rangle$, the actions have the following definition:

$$\begin{aligned} PRE(a_I) &= \emptyset & \qquad PRE(a_G) &= G \\ ADD(a_I) &= I & \qquad ADD(a_G) &= \emptyset \\ DEL(a_I) &= \emptyset & \qquad DEL(a_G) &= \emptyset \end{aligned}$$

The inclusion of a_I and a_G actions allow us to simplify the presentation of many algorithms, avoiding special checks in the procedure (e.g., we can assume that there will always be a "first" and "last" action in the POP).

Depending on how the POP was constructed, it may include a set of causal links, $\mathcal{C}$. Each causal link contains a pair of ordered actions, a_1, a_2 (a_1 may be a_I and a_2 may be a_G), and a fluent, p, such that a_1 achieves p for a_2: denoted as $(a_1 \xrightarrow{p} a_2)$. Causal links often serve as justifications for the ordering constraints in a POP.

Definition 1 (POP Validity: Notion 1)**.** A POP P is valid for a planning problem Π if and only if every linearization of P is a sequential plan for Π.[2]

2. Note that notion 1 does not rely on the set of causal links $\mathcal{C}$.

While simple and intuitive, notion 1 is rarely used to verify the validity of a POP because there may be a prohibitively large number of linearizations represented by the POP. There is, however, a tractable equivalent notion of POP validity that uses the concepts of causal links, open preconditions and threats.

For a POP $\langle \mathcal{A}, \mathcal{O} \rangle$ and a set of causal links $\mathcal{C}$, an *open precondition* is a precondition p of an action $a \in \mathcal{A}$ that does not have an associated causal link:

$$\nexists a' \in \mathcal{A} \text{ s.t. } (a' \xrightarrow{p} a) \in \mathcal{C}$$

If a precondition is not open, we say that it is *supported*, and we refer to the associated action in the causal link as the *achiever* for the precondition. In a typical valid POP there will only be one supporter for every precondition of an action included in the POP, but we do not make that restriction in our work to keep the encoding general.

A *threat* in a POP refers to an action that can invalidate a causal link between two other actions due to ordering constraints (or lack thereof). Formally, if $(a_1 \xrightarrow{p} a_2) \in \mathcal{C}$, we say that the action a_3 (distinct from a_1 and a_2) threatens the causal link $(a_1 \xrightarrow{p} a_2)$ if the following two conditions hold:

- We can order a_3 between a_1 and a_2:

$$\{(a_3 \prec a_1), (a_2 \prec a_3)\} \cap \mathcal{O} = \emptyset$$

- The action a_3 deletes p:

$$p \in DEL(a_3)$$

The existence of a threat means that a linearization exists that violates some causal link, and thus may not be executable. With the actions a_I and a_G included, we have the following definition that characterizes the second notion of POP validity.

Definition 2 (POP Validity: Notion 2)**.** Given a planning problem Π, POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and set of causal links $\mathcal{C}$, P is a valid POP for the planning problem Π if no action in $\mathcal{A}$ has an open precondition and no causal link in the set $\mathcal{C}$ has a threatening action in $\mathcal{A}$.

A causal link structure can be implicitly assessed to verify POP validity in polynomial time (Nebel & Bäckström, 1994), so the set $\mathcal{C}$ is not strictly necessary. However, implicitly or explicitly, notion 2 requires all actions to be causally supported in a threat-free manner. We subsequently have the following connection between the two notions of POP validity:

Theorem 1 (POP Validity, due to McAllester & Rosenblitt, 1991)**.** *If notion 2 of POP validity holds, then notion 1 also holds. Additionally, if notion 1 holds for $\mathcal{A}$ and $\mathcal{O}$, then a set of causal links $\mathcal{C}$ must exist such that notion 2 holds for $\langle \mathcal{A}, \mathcal{O} \rangle$ and $\mathcal{C}$.*

The final concept we use for a POP is based on the well-established metric for measuring how constrained a POP is (Nguyen & Kambhampati, 2001; Siddiqui & Haslum, 2012): *flex* is a measure of how many ordering constraints there are in the POP, normalized by the total number of potential ordering constraints. The *flex* tends to 1 as the number of ordering constraints tends to 0, and vice versa. As per usual, we assume that the set of ordering constraints is transitively closed.

Definition 3 (*flex*)**.** Given a POP $\langle \mathcal{A}, \mathcal{O} \rangle$, we define *flex* as,

$$flex(\langle \mathcal{A}, \mathcal{O} \rangle) = 1 - \frac{|\mathcal{O}|}{\sum_{i=1}^{|\mathcal{A}|-1} i}$$

We use $\sum_{i=1}^{|\mathcal{A}|-1} i$ as the denominator instead of the traditional $|\mathcal{A}|^2$ as the latter over counts the number of possible ordering constraints. With our definition of *flex*, a fully unordered POP will have a *flex* value of 1 while a sequential plan will have a *flex* 0.

We strive to minimize the number of ordering constraints in the transitive closure. Omitting the transitive closure would amount to optimizing over the transitive reduction which, as noted by Bäckström (1998), has less appeal because it leads to more long chains in the plan.

2.2 Deorderings and Reorderings

The aim of least commitment planning is to find flexible plans that allow us to defer decisions regarding the execution of the plan. Considering only the ordering constraints of a POP, two important notions for least commitment planning are the *deordering* and *reordering* of a POP. Following Bäckström (1998), we define these formally as follows:

Definition 4 (Deordering and Reordering)**.** Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a STRIPS planning problem:

1. Q is a deordering of P wrt. Π iff P and Q are valid POPs for Π, $\mathcal{A} = \mathcal{A}'$, and $\mathcal{O}' \subseteq \mathcal{O}$.
2. Q is a reordering of P wrt. Π iff P and Q are valid POPs for Π, and $\mathcal{A} = \mathcal{A}'$.

Recall that we assume the ordering constraints of a POP to be transitively closed, and every action in a POP is uniquely named (i.e., every repetition of the same action is given a unique name). A *proper* deordering is one where the ordering constraints form a proper subset (i.e., $\mathcal{O}' \subsetneq \mathcal{O}$). We define the minimum deordering / reordering as follows:

Definition 5 (Minimum Deorderings and Reorderings)**.** Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a STRIPS planning problem:

1. Q is a minimum deordering of P wrt. Π iff
   (a) Q is a deordering of P wrt. Π, and
   (b) there is no deordering $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$.
2. Q is a minimum reordering of P wrt. Π iff
   (a) Q is a reordering of P wrt. Π, and
   (b) there is no reordering $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$.
3. Q is a minimal deordering of P wrt. Π iff
   (a) Q is a deordering of P wrt. Π, and
   (b) there is no proper deordering of Q.

Note that we use cardinality rather than set containment for 1(b) and 2(b) because the orderings in $\mathcal{O}'$ and $\mathcal{O}''$ need not overlap. We will equivalently refer to a minimum deordering (resp. reordering) as an *optimal* deordering (resp. reordering). In both cases, we prefer a POP that has the smallest set of ordering constraints. In other words, no POP exists with the same actions and fewer ordering constraints while remaining valid with respect to Π. The problem of finding a minimum deordering or reordering of a POP is NP-hard, and cannot be approximated within a constant factor unless $NP \in \mathrm{DTIME}(n^{poly\ log\ n})$ (Bäckström, 1998). There may be many such optimal deorderings or reorderings, but in our work we do not distinguish them further. We can compute a minimal deordering in polynomial time by iteratively removing unnecessary ordering constraints (i.e., those that do not cause the plan to become invalid).

2.3 Previous Approaches

There are many approaches to computing a partial-order plan, and we cover some of the representative examples here.

2.3.1 Partial-Order Causal Link Algorithms

Traditional methods for producing a partial-order plan follow an approach called *partial-order causal link* (POCL) planning (Weld, 1994). In POCL planning, modifications are iteratively made to an incomplete partial-order plan that consists of a set of actions, causal links, and ordering constraints. A partial-order plan is considered complete if and only if the conditions for Definition 2 are met.

The key difference between POCL planning and the standard state-based search is that POCL planning is a search through plan space. In POCL planning search, every node in the search space constitutes a partial plan, whereas in state-based search every node is a state of the world; successor nodes are generated by applying actions to the state represented by the current search node. In contrast, possible modifications to a partial plan represent the choices available in the POCL planning search procedure. The typical partial plan modifications include:

1. Add a new action to the partial plan.
2. Order two actions in the partial plan.
3. Create a causal link between two actions in the plan.

POCL planners were popular in the late 1970s, 1980s, and 1990s, starting with Tate's NONLIN planner (Tate, 1976), until forward search techniques such as the one employed by the FF planner (Hoffmann & Nebel, 2001) led planning research in a new direction. The most recent POCL planner is VHPOP (Younes & Simmons, 2003), but unfortunately it is not competitive with the state-of-the-art forward search planners.

2.3.2 POPF

To take advantage of the flexibility afforded by a POP and the search efficiency of forward state-based planners, Coles *et al.* introduced the forward-chaining partial-order planner

POPF (Coles et al., 2010). The idea behind POPF is to restrict the modifications permitted to the partially completed plan so that a complete state can be easily computed that represents the truth of fluents after the partial plan is executed. Unlike POCL approaches that add actions to achieve open preconditions, the actions in POPF are chosen so that their preconditions are satisfied and heuristically lead to the goal (i.e., in a forward-search manner). When a new action is added to the plan, it is placed at the end of the plan – no action already in the incumbent plan can be ordered *after* the newly added action at the time of its insertion, but the new action may be left unordered with respect to actions already in the plan. Further, adding a new action requires that all of its preconditions have causal links created immediately.

The approach used in POPF leverages the partial-order nature of planning domains by avoiding some of the unnecessary reasoning about the permutations of unordered actions; ordering constraints are included only as required. Sequential planners may try to complete the same partial-order plan multiple times with the only change being a different permutation of unordered actions, and POPF can avoid this situation some of the time by maintaining the partial-order structure. Further, using recently introduced techniques to detect repeated states (Coles & Coles, 2016), the planner avoids even more unnecessary permutations of the action sequences.

Finally, POPF leverages the powerful techniques of forward-search planners by maintaining the complete state of the world that will be reached by the plan. Having this state information allows for powerful heuristics to be computed efficiently.

2.3.3 Petri Net Unfolding

Predating the work of Coles et al. (2010), an alternative approach to generating partially ordered plans is via Petri net unfolding (Hickmott, 2008). The general idea is to encode the evolution of a forward planning system through the repeated unfolding of a carefully crafted Petri net: a mathematical structure used to model and analyze the dynamics of discrete distributed systems (Murata, 1989). The unfolding process naturally represents a parallel or partially ordered plan (Hickmott, Rintanen, Thiébaux, & White, 2007).

In 2009, Hickmott and Sardina detailed a theoretical property of Petri net unfolding for partial-order plans, noting that the plan resulting from Petri net unfolding is a minimal deordering or reordering that respects *strong independence* (Hickmott & Sardina, 2009). Strong independence is a restriction on the unordered actions in the partial-order plan: there can be no ambiguity with respect to which action produces a particular fluent. As a result, if two different actions each produce the same fluent f, they can only be unordered if neither is required to produce f – either in service of achieving the goal or in service of the successful execution of some other action in the plan. This restriction makes the deorderings and reorderings produced by the unfolding more restrictive than the optimal deorderings or reorderings produced by our approach, as we do not require strong independence.

Similar to the POCL and POPF approaches, Petri net unfolding is exploited to produce a partial-order plan directly, rather than finding a deordering or reordering of an existing plan, as we do in this paper.

2.3.4 Relaxer Algorithm

Due to Kambhampati and Kedar (1994), the Relaxer Algorithm[3] operates by removing ordering constraints from a sequential plan in a systematic manner. A heuristic guides the procedure and, as detailed by Bäckström (1998), the process does not provide any guarantee that the resulting POP is minimally deordered. There is an error in the counterexample used by Bäckström to demonstrate that Kambhampati and Kedar's algorithm does not necessarily produce a minimally deordered POP. However, the conclusion is correct and we provide a new counterexample in Appendix A.

The intuition behind the algorithm is to remove any ordering $(a_i \prec a_k)$ from the sequential plan where a_i is not the achiever of some precondition of a_k and removing the ordering does not lead to a threat. The algorithm heuristically attempts to choose the earliest possible action in the sequential plan as the achiever of a precondition. For example, consider the case where our sequential plan is $[a_1 \cdots, a_i, \cdots, a_k, \cdots, a_n]$ and $p \in PRE(a_k)$. The algorithm will keep the ordering $(a_i \prec a_k)$ only if leaving it out would create a threat for a precondition of one of the actions, or if a_i is the earliest action in the sequence where the following holds:

1. $p \in ADD(a_i)$: a_i is an achiever for p
2. $\forall a_j, i < j < k, p \notin DEL(a_j)$: p is not threatened.

Algorithm 1 presents this approach formally. We use **index**$(a,\vec{a})$ to refer to the index of action a in the sequence $\vec{a}$, and assume every action in the plan is uniquely named.

If $\vec{a}$ is a valid plan, line 8 will evaluate to true before either line 11 evaluates to true or the for-loop at line 6 runs out of actions. That is, we know an unthreatened achiever exists and the earliest such one is found. The achiever is then ordered before the action requiring the fluent as a precondition (line 14), and the for-loop at line 16 adds all of the necessary ordering constraints so the achiever remains unthreatened. Note that for any deleter found in this for-loop, either line 17 or 19 must evaluate to true. After going through the outer loop at line 3, every action in the newly formed POP has an unthreatened supporting action for each of its preconditions. The resulting POP will therefore be valid (cf., Kambhampati & Kedar, 1994, section 5.2).

2.3.5 SAPA Post-Processing

As part of a post-processing phase for the SAPA planner, Do and Kambhampati (2003) introduce an approach similar to ours for relaxing the ordering of a plan. In their setting, they begin with a temporal plan with the actions assigned to specific time points, and the objective is to optimize either the number of ordering constraints or some temporal aspect of the resulting plan.

The strategy Do and Kambhampati take (abbreviated as DK here), is to model the task of computing a partial-order relaxation in terms of a constraint satisfaction optimization problem (CSOP). Variables are introduced to represent the ordering of actions, the timing and duration of actions, the resource usage, etc. From the abstract CSOP formalism, a concrete mixed integer linear program (MILP) is proposed to realize the set of constraints

3. Referred to as "order generalization" originally.

Algorithm 1: Relaxer Algorithm

Input: Sequential plan, $\vec{a}$, including a_I and a_G
Output: Relaxed Partial-order plan, $\langle \mathcal{A}, \mathcal{O} \rangle$

```
A = set(a⃗);
O = ∅;
foreach a ∈ A do
    foreach f ∈ PRE(a) do
        ach = null;
        for i = (index(a, a⃗) − 1) ··· 0 do
            // See if we have an earlier achiever
            if f ∈ ADD(a⃗[i]) then
                ach = a⃗[i];
            // Stop if we find a deleter of f
            if f ∈ DEL(a⃗[i]) then
                break;
        // Add the appropriate supporting link
        O = O ∪ {(ach ≺ a)};
        // Add orderings to avoid threats
        foreach a′ ∈ deleters(f) \ {a} do
            if index(a′, a⃗) < index(ach, a⃗) then
                O = O ∪ {(a′ ≺ ach)};
            if index(a′, a⃗) > index(a, a⃗) then
                O = O ∪ {(a ≺ a′)};
return ⟨A, O⟩;
```

that model a valid temporal plan. Similar to our work, DK contains the option for enforcing adherence to the original ordering constraints which allows either a deordering or a reordering to be produced.

DK considers a number of optimization criteria including minimizing the makespan, maximizing the sum of slack in the temporal variables, maximizing the flexibility in the temporal variables, and minimizing the number of ordering constraints. While the first three are related to temporal planning domains, the final one coincides with the optimization criteria of our work. Experimental evaluation is provided for the temporal optimization criterion, but Do and Kambhampati do not empirically investigate the minimization of ordering constraints.

Differences between DK and our approach include the formalism (we do not focus on temporal aspects), the model used (unique to our encoding are variables that represent an action appearing in the plan and unique to their encoding are variables representing time points and resources), the underlying solving technology (we rely on partial weighted MaxSAT instead of MILP), and finally the MCLCP criterion. In Section 4.4 we compare the efficiency of our approach for computing a minimum reordering with an implementation of the DK approach that uses only the variables and constraints relevant to computing a minimum reordering.

2.4 Partial Weighted MaxSAT

To compute a relaxed plan, we encode the task as a *partial weighted MaxSAT* problem where a solution to the encoding corresponds to a minimally relaxed plan that optimizes our desired criteria. Here, we review the notation for partial weighted MaxSAT that we use throughout the paper.

In Boolean logic, the problem of Satisfiability (SAT) is to find a true/false setting of Boolean variables such that a logical formula referring to those variables evaluates to true (Biere, Heule, van Maaren, & Walsh, 2009). Typically, we write problems in Conjunctive Normal Form (CNF), which is made up of a conjunction of clauses, where each clause is a disjunction of literals. A literal is either a Boolean variable or its negation. A setting of the variables satisfies a CNF formula iff every clause has at least one literal that evaluates to true. For example, setting variables x and z to be true will satisfy the following theory:

$$(\boldsymbol{x} \vee y) \wedge (\neg x \vee \boldsymbol{z}) \tag{1}$$

The MaxSAT problem is the optimization variant of the SAT problem in which the goal is to maximize the number of satisfied clauses (Biere et al., 2009, ch. 19). Although we cannot satisfy every clause in the following theory, setting x, y to true and z to false satisfies five clauses:

$$(\boldsymbol{x} \vee \boldsymbol{y} \vee \boldsymbol{\neg z}) \wedge (\boldsymbol{x} \vee z) \wedge (\boldsymbol{y} \vee z) \wedge (\neg x \vee \neg y) \wedge (\boldsymbol{\neg z} \vee \neg x) \wedge (\boldsymbol{\neg z} \vee \neg y) \tag{2}$$

Adding non-uniform weights to each clause allows for a richer version of the optimization problem, and we refer to maximizing the weight of satisfied clauses as the weighted MaxSAT problem. We use the syntax $\overset{k}{(\cdots)}$ to indicate the clause has a weight of k. Generally, the weight must be a positive real number. Consider setting x to false and y, z to true in the following theory:

$$\overset{3}{(x)} \wedge \overset{1}{(\boldsymbol{\neg x} \vee \neg y)} \wedge \overset{1}{(\boldsymbol{\neg x} \vee \neg z)} \wedge \overset{1}{(\boldsymbol{y})} \wedge \overset{1}{(\boldsymbol{z})} \tag{3}$$

While the setting satisfies four clauses, it only has a total weight of 4. With the aim of maximizing the total weight of satisfied clauses, we can achieve a sum of 5 by assigning all variables to true:

$$\overset{3}{(\boldsymbol{x})} \wedge \overset{1}{(\neg x \vee \neg y)} \wedge \overset{1}{(\neg x \vee \neg z)} \wedge \overset{1}{(\boldsymbol{y})} \wedge \overset{1}{(\boldsymbol{z})} \tag{4}$$

If we wish to force the solver to find a solution that satisfies a particular subset of the clauses, we refer to clauses in this subset as *hard*, while all other clauses in the problem are *soft*. The syntax we use to indicate a hard clause is $\overset{\bullet}{(\cdots)}$. When we have a mix of hard and soft clauses, we have a partial weighted MaxSAT problem (Biere et al., 2009, ch. 19.6).

In a partial weighted MaxSAT problem, only the soft clauses are given a weight, and a feasible solution corresponds to any setting of the variables that satisfies the hard clauses in the CNF. An optimal solution to a partial weighted MaxSAT problem is any feasible solution that maximizes the sum of the weights on the satisfied soft clauses. In the following example, setting variables x, y to false and z to true satisfies every hard clause and one of the soft clauses:

$$\overset{1}{(x)} \wedge \overset{2}{(y)} \wedge \overset{3}{(\boldsymbol{z})} \wedge (\neg\boldsymbol{x} \overset{\bullet}{\vee} \neg z) \wedge (\neg\boldsymbol{y} \overset{\bullet}{\vee} \neg z) \wedge (\neg\boldsymbol{x} \overset{\bullet}{\vee} \neg\boldsymbol{y}) \tag{5}$$

Although not required for partial weighted MaxSAT in general, the encodings we create will never contain a soft clause that has more than one literal. This special form of partial weighted MaxSAT problem, referred to as a binate covering problem (Coudert, 1996), allows us to flip the optimization criterion: minimizing the sum of the satisfied soft (unit) clauses is equivalent to maximizing the sum of unit clauses that have the literal flipped (e.g., x goes to $\neg x$ and vice versa). Using this technique to solve the minimization problem with a partial weighted MaxSAT solver only works if the soft clauses contain a single literal. This property is key to our encoding, as our objective is always to minimize.

3. Approach

We can view a sequential plan (also referred to as a total-order plan) as a special case of a partial-order plan where there exists an ordering constraint between every pair of actions. Quite often, many of these ordering constraints are not required: the ordering of certain actions may be switched and the goal still achieved with the new sequence of actions. With the aim of maximizing the flexibility of a POP, we strive to minimize the number of ordering constraints included in the solution. This objective motivates the need to identify precisely which ordering constraints in a POP are relevant to the POP's validity.

Definition 6 (Ordering Relevance)**.** Given a planning problem $\Pi = \langle F, I, G, A\rangle$ and valid POP $P = \langle \mathcal{A}, \mathcal{O}\rangle$ for Π, the ordering constraint $o \in \mathcal{O}$ is *relevant* with respect to Π and P iff $\langle \mathcal{A}, \mathcal{O} - \{o\}\rangle$ is not a valid POP for Π.[4]

Ordering relevance plays a central role in the definitions of minimal and minimum POP deorderings: the relevant ordering constraints are precisely those that cannot be removed without invalidating the POP (Bäckström, 1998). Additionally, the Relaxer Algorithm of Kambhampati and Kedar (1994) operates by identifying a set of ordering constraints suspected of being relevant (i.e., those selected as achievers for action preconditions).

To maximize the flexibility of a POP, we focus our encoding on retaining only the relevant orderings. While difficult to measure efficiently, we strive to maximize the *flexibility* inherent in a POP, loosely defined as the number of linearizations a POP represents. The number of unordered pairs of actions in a POP, typically referred to as *flex* (Siddiqui & Haslum, 2012), provides an approximation for the POP's flexibility. In our evaluation, we quantify the accuracy of *flex* as an approximation for a POP's flexibility.

As we have discussed earlier, verifying a POP's validity by way of the linearizations is not always practical. Similarly, we will not attempt to compute POPs that maximize

4. Note that the transitive closure of P is necessarily different from the transitive closure of $\langle \mathcal{A}, \mathcal{O} - \{o\}\rangle$ when o is relevant with respect to Π and P.

the number of linearizations, but rather we will compute POPs that adhere to one of the previously mentioned criteria for removing redundant orderings: minimum deordering or minimum reordering.

3.1 Minimum Cost Least Commitment Criterion

While the notion of a minimum deordering or reordering of a POP addresses the commitment of ordering constraints, an orthogonal objective is to commit as few resources as possible – typically measured as either the time for a plan to be executed in parallel or the sum of action costs for the actions in a plan. Historically, the latter objective takes precedence over all other metrics. To this end, we provide the extended criterion of computing a *minimum cost least commitment POP* (MCLCP).

Definition 7 (Minimum Cost Least Commitment POP)**.** Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs valid for Π. Q is a *minimum cost least commitment POP* (MCLCP) of P iff Q is a minimum reordering, $\mathcal{A}' \subseteq \mathcal{A}$, and there does not exist a valid POP $R = \langle \mathcal{A}'', \mathcal{O}'' \rangle$ for Π such that $\mathcal{A}'' \subseteq \mathcal{A}$ and the following condition holds:

$$c_\Pi(R) < c_\Pi(Q) \vee (c_\Pi(R) = c_\Pi(Q) \wedge |\mathcal{O}''| < |\mathcal{O}'|)$$

For this work, we assume that every action in Π has positive cost. It may turn out that preferring fewer actions causes us to commit to more ordering constraints, simply due to the interaction between the actions we choose. In practice, however, we usually place a much greater emphasis on minimizing the total cost of a plan. It is also worth noting that if no plan exists with a proper subset of the actions in the input plan, computing the MCLCP is equivalent to computing a minimum reordering.

Following the MCLCP criterion, we can evaluate the quality of a POP by the total action cost and number of ordering constraints it contains; these metrics give us a direct measure of the least commitment nature of a POP with the primary emphasis placed on removing the unnecessary commitments to actions.

3.2 Encoding

We encode the task of finding a minimum deordering, reordering, or MCLCP as a partial weighted MaxSAT problem given an input planning problem and corresponding initial plan. An optimal solution to the default encoding will correspond to an MCLCP. That is, no POP exists with a cheaper overall cost or with the same cost and fewer ordering constraints in the transitive closure. We present this core encoding in Section 3.2.1 and prove the soundness and completeness of the encoding in Section 3.2.2. We add further clauses to produce encodings that correspond to optimal deorderings or reorderings, and present these modifications in Section 3.2.3.

3.2.1 Basic Encoding

In contrast to the typical SAT encoding for a planning problem (e.g., Kautz & Selman, 1999), we do not require that the actions be replicated for successive plan steps. Instead, we represent each action occurrence only once and reason about the ordering between actions. The actions in the encoding come from a provided sequential or partial-order plan,

$P = \langle \mathcal{A}, \mathcal{O} \rangle$. We use $\Phi(P)$ to denote the partial weighted MaxSAT encoding corresponding to the POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$, and refer to the POP corresponding to an encoding's solution as the *target POP*. A target POP can be reconstructed from an encoding's solution by looking at only the variables set to true. We use three types of propositional variables:

- x_a: For every action a in $\mathcal{A}$, x_a indicates that action a appears in the target POP.
- $\kappa(a_1, a_2)$: For every pair of actions a_1, a_2 in $\mathcal{A}$, $\kappa(a_1, a_2)$ indicates that the ordering constraint $(a_1 \prec a_2)$ appears in the target POP.
- $\Upsilon(a_i, p, a_j)$: For every action a_j in $\mathcal{A}$, p in PRE(a_j), and a_i in **adders**(p), $\Upsilon(a_i, p, a_j)$ indicates a_i supports a_j with the fluent p in the target POP.

In a partial weighted MaxSAT encoding there is a distinction between hard and soft clauses. We first present the hard clauses of the encoding as Boolean formulae which we subsequently convert to CNF, and later describe the soft clauses with their associated weights.[5] We define the formulae that ensure that the target POP is acyclic, and the ordering constraints include the transitive closure. Here, actions are universally quantified, and for formula (9) we assume $a_I \neq a_i \neq a_G$. We must ensure that:

- There are no self-loops:

$$(\neg \kappa(a, a)) \tag{6}$$

- We include the initial and goal actions:

$$(x_{a_I}) \wedge (x_{a_G}) \tag{7}$$

- If we use an ordering variable, then we include both actions:

$$\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j} \tag{8}$$

- An action cannot appear before the initial action (or after the goal):

$$x_{a_i} \rightarrow \kappa(a_I, a_i) \wedge \kappa(a_i, a_G) \tag{9}$$

- A solution satisfies the transitive closure of ordering constraints:

$$\kappa(a_i, a_j) \wedge \kappa(a_j, a_k) \rightarrow \kappa(a_i, a_k) \tag{10}$$

Together, (6) and (10) ensure that the target POP will be acyclic (note that this implies antisymmetry as well), while the remaining formulae tie the two types of variables together and deal with the initial and goal actions. Finally, we include the formulae needed to ensure that every action has its preconditions met, and there are no threats in the solution:

5. For readability, we omit the hard clause symbol, $\overset{\bullet}{(\cdots)}$, for constraints (6)-(12).

$$\Upsilon(a_i, p, a_j) \rightarrow \bigwedge_{a_k \in \mathbf{deleters}(p)} x_{a_k} \rightarrow \kappa(a_k, a_i) \vee \kappa(a_j, a_k) \tag{11}$$

$$x_{a_j} \rightarrow \bigwedge_{p \in PRE(a_j)} \bigvee_{a_i \in \mathbf{adders}(p)} \kappa(a_i, a_j) \wedge \Upsilon(a_i, p, a_j) \tag{12}$$

Intuitively, $\Upsilon(a_i, p, a_j)$ holds if a_i is the achiever of precondition p for action a_j and no deleter of p will be allowed to occur between the actions a_i and a_j; i.e., it corresponds directly to an unthreatened causal link. Formula (11) ensures that every causal link remains unthreatened in a satisfying variable setting, and we can view the two ordering variables in the formula as a form of the common partial-order planning concepts of promotion and demotion (Weld, 1994). Formula (12) ensures that if we include action a_j in the target POP, then every precondition p of a_j must be satisfied by at least one achiever a_i. $\kappa(a_i, a_j)$ orders the achiever correctly, while $\Upsilon(a_i, p, a_j)$ removes the possibility of a threatening action.

So far, the constraints we have described capture what is required for a POP to be valid. To go further and address the notion of ordering relevance presented in Definition 6, as well as the metric of minimizing total action cost with MCLCP, we make use of soft clauses. To generate an MCLCP, we prefer solutions that first minimize the total action cost, and then minimize the number of ordering constraints. We add a soft unit clause, containing the negation of the variable, for every action and ordering variable in our encoding. A violation of any one of the unit clauses means that the solution includes the action or ordering constraint corresponding to the violated clause's variable. The weight assigned is as follows:

- $\overset{1}{(\neg\kappa(a_i, a_j))}, \;\; \forall a_i, a_j \in \mathcal{A}$
- $\overset{c_\Pi(a)+|\mathcal{A}|^2+1}{(\neg x_a)}, \;\; \forall a \in \mathcal{A} \setminus \{a_I, a_G\}$

Note that the weight of any single action clause is greater than the weight of all ordering constraint clauses combined, because there can be no more than $|\mathcal{A}|^2$ total ordering constraints. The increased weight guarantees that we generate solutions with a minimum action cost.[6] Because we enforce the transitive closure of the ordering constraints, the second type of soft clause will lead the solver to find a POP (among those with the cheapest total action cost) that minimizes the size of the transitive closure.

Richer notions, such as a weighted trade-off between the ordering constraints and action costs, are also easily modelled using an appropriate assignment of weights to the soft clauses in the encoding. As we focus primarily on the deordering and reordering aspects in this work, we leave alternative encodings as future work.

3.2.2 Theoretical Results

In this section we present theoretical properties of our core encoding.

6. If we wish to minimize the number of actions in the solution, we need only to replace $c_\Pi(a)$ with 0.

Lemma 1 (Variable Setting Implies POP)**.** *Given a planning problem* Π *and a valid POP* $P = \langle \mathcal{A}, \mathcal{O} \rangle$*, any variable setting that satisfies the formulae (6)-(12) for* $\Phi(P)$ *will correspond to a valid POP for* Π *where the ordering constraints are transitively closed.*

Proof. We have already seen that the POP induced by a solution to the hard clauses will be acyclic and transitively closed (due to formulae (6)-(10)). We can further see that there will be no open preconditions because we include a_G, and the conjunction of (12) ensures that every precondition will be satisfied when the POP includes an action. Additionally, there are no threats in the final solution because of formula (11), which will be enforced every time a precondition is met by formula (12). Because the POP corresponding to any solution to the hard clauses will have no open preconditions and no threats, Theorem 1 allows us to conclude that the target POP will be valid for Π. □

Lemma 2 (POP Implies Variable Setting)**.** *Given a planning problem* Π *and a valid POP* $P = \langle \mathcal{A}, \mathcal{O} \rangle$*, any valid POP* $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$*, where* $\mathcal{A}' \subseteq \mathcal{A}$ *and* $\mathcal{O}'$ *is transitively closed, has a corresponding feasible variable assignment that satisfies* $\Phi(P)$*.*

Proof. The lemma follows from the direct encoding of the POP Q where $x_a = \textit{true}$ iff $a \in \mathcal{A}'$ and $\kappa(a_i, a_j) = \textit{true}$ iff $(a_i \prec a_j) \in \mathcal{O}'$. If Q is a valid POP, then it will be acyclic, include a_I and a_G, have all actions ordered after a_I and before a_G, and be transitively closed (satisfying (6)-(10)). We further can see that (11) and (12) must be satisfied: if (12) did not hold, then there would be an action a in the POP with a precondition p such that *every* potential achiever of p has a threat that could be ordered between the achiever and a. Such a situation is only possible when the POP is invalid, which is a contradiction. □

Theorem 2 (Completeness)**.** *Given a planning problem* Π *and a valid POP* $P = \langle \mathcal{A}, \mathcal{O} \rangle$*, a complete partial weighted MaxSAT solver will find a solution to the soft clauses and formulae (6)-(12) for* $\Phi(P)$ *that minimizes the total cost of actions in the corresponding POP, and subsequently minimizes the number of ordering constraints.*

Proof. Given $|\mathcal{A}|$ actions, there can only be $|\mathcal{A}|^2$ ordering constraints. Because every soft clause corresponding to an ordering constraint has a weight of 1, the total sum of satisfying every ordering constraint clause will be $|\mathcal{A}|^2$. Because the weight of satisfying any action clause is greater than $|\mathcal{A}|^2$, the soft clauses corresponding to actions dominate the optimization criteria. As such, there will be no valid POP for Π which has a subset of the actions in P with a lower total action cost than a solution that satisfies formulae (6)-(12) while maximizing the weight of the satisfied soft clauses. □

Theorem 3 (Encoding Correctness)**.** *Given a planning problem* Π*, and a valid POP* P *for* Π*, a solution to our partial weighted MaxSAT encoding* $\Phi(P)$ *is an MCLCP for* P*.*

Proof. This Theorem follows directly from Lemmas 1, 2, and Theorem 2. □

3.2.3 Variations

Observe that $\Phi(P)$ does not make use of the set of ordering constraints in P. An optimal solution to the encoding will correspond to an MCLCP, but to enforce solutions that are minimum deorderings or reorderings, we introduce two additional sets of hard clauses.

All Actions: For optimal deorderings and reorderings, we require every action to be a part of the target POP. We consider a formula that ensures that we use every action (and so the optimization works only on the ordering constraints). To achieve this, we simply need to add each action as a hard clause:

$$(\overset{\bullet}{x_a}), \ \ \forall a \in \mathcal{A} \tag{13}$$

The soft unit clauses will all be trivially unsatisfiable, and are removed in the preprocessing phase of the MaxSAT solving process. An optimal solution to the soft constraints and formulae (6)-(13), referred to as $\Phi^{MR}(P)$, corresponds to a minimum reordering of P.

Deordering: For a deordering we must forbid any explicit ordering that contradicts the input plan. Assuming our input plan is $P = \langle \mathcal{A}, \mathcal{O} \rangle$, we ensure that the computed solution is a deordering by adding the following family of hard unit clauses:

$$(\neg \kappa(\overset{\bullet}{a_i}, a_j)), \ \ \forall (a_i \prec a_j) \notin \mathcal{O} \tag{14}$$

Similar to the introduction of hard unit clauses for action inclusion, using the clauses from (14) will eliminate a number of ordering constraint soft clauses from the encoding during the preprocessing phase of the MaxSAT solver. An optimal solution to the soft constraints and formulae (6)-(14), referred to as $\Phi^{MD}(P)$, corresponds to a minimal deordering of P. We additionally could use (14) and forgo the use of (13), but this variation is not one typically studied, nor does it provide a benefit over computing an MCLCP.

4. Evaluation

We evaluate the ability and effectiveness of the state-of-the-art partial weighted MaxSAT solver, Sat4j (Le Berre & Parrain, 2010), to optimally relax a plan using our proposed encodings.[7] We use the MD and MR encodings, which ensure that all actions are always included in the solution (i.e., using the "All Actions" constraint (13)). We also investigate the effectiveness of the Relaxer Algorithm (RX) to produce a minimally constrained deordering. To measure the quality of a POP, we use either its *flex* value (cf. Section 2), or the number of linearizations (whenever feasible to compute).

For our analysis, we considered every STRIPS domain from the previous International Planning Competitions (IPC, Hoffmann, 2016). We discarded two domains (childsnack and tidybot) due to the difficulty that planners had in generating an initial solution. A further 18 were discarded due to their constrained nature; the form of which offers little or no flexibility (any domain with an average flex value of less than 10% was removed).[8] Using them in the evaluation would be uninformative since they are already as relaxed as

7. Additionally, we evaluated the 2013 winner of the partial weighted MaxSAT contest for crafted instances, MaxHS (Davies & Bacchus, 2013), however, we found that Sat4j outperformed MaxHS slightly in both coverage and time.
8. The 18 overly constrained domains are visitall, blocksworld, sokoban, pegsol, ged, parking, barman, gripper, cybersec, psr-small, storage, nomystery, mystery, mprime, freecell, hiking, floortile, and thoughtful.

possible, and the solver determined this trivially. We evaluate using only the most recent version of a domain where multiple problem sets exist, and Table 1 shows the set of 15 domains that we considered throughout our evaluation.

We conducted all experiments on a Linux desktop with a 3.4GHz processor, and each run of Sat4j was limited to 30 minutes and 4GB of memory. To generate an initial sequential plan, we used the Mercury planner (Domshlak, Hoffmann, & Katz, 2015); the best performing non-portfolio planner from the most recent satisficing IPC competition. Additionally, for some of the evaluation, we computed initial solutions using a state-of-the-art SAT-based planner, Mp (Rintanen, 2012), and a state-of-the-art partial-order planner, POPF (Coles et al., 2010). Both offer alternative methods that generate an initial partially ordered plan, and we investigate the impact that the plan's structure has on the relaxation process.

We assess various aspects of our approach through four separate experiments. First, we evaluate the difficulty of computing a *feasible* solution in addition to the optimal one (we obtain solutions of increasing quality by using Sat4j in an any-time fashion). Next, we look at the quality of the POP produced by our encodings as well as the POP produced by the Relaxer Algorithm. Here, we measure quality both as the *flex* of the plan in the transitive closure, and as the number of linearizations in the plan wherever feasible to compute. We also demonstrate empirically the accuracy of the *flex* measure as an indicator of the number of linearizations. Next, we consider the impact that the initial plan form has on the relaxation, taking into account the starting solution of the three planners. Finally, we compare our approach for computing a minimum reordering with that of a similar approach by Do and Kambhampati (2003).

4.1 Solving to Completion

We begin with a brief discussion of the various configurations of our approach and their coverage, as well as the weaknesses of some of the methods. We report on only the problems where the planner was able to find a plan within the resource limits.[9] Table 1 shows the following information for every domain:

- The number of problems in the domain is shown in brackets next to the domain name.
- The number of problems solved by each planner is under the **Plans** column.
- The **Solved** column indicates the number of plans successfully encoded and solved. Every problem that could be encoded for MR also could be encoded for MD, and the MaxSAT solver produced at least one solution for every encoded problem. Further, every encoded MD problem was solved to completion within the resource limits.
- The **MR** column indicates the number of encoded MR problems solved to completion.

We must emphasize that it is not the purpose of this evaluation to compare the efficiency of the three planners (as each have their own strengths and weaknesses). Rather, we consider the type of plan that each produces as related to relaxing the ordering constraints on the plan. Consequently, the purpose of Table 1 is to provide insight into which problems are included in our further analysis, and to bring to light some of the challenges of encoding and solving the problems to completion.

9. Providing twice the amount of time and memory to the planners did not lead to more problems solved.

Domain	Mercury			POPF			Mp		
	Plans	Solved	MR	Plans	Solved	MR	Plans	Solved	MR
airport (50)	32	29	29	24	21	21	32	29	29
depot (22)	21	21	19	10	10	10	20	20	14
driverlog (20)	20	20	16	15	15	15	17	15	10
elevators (20)	20	10	6	1	1	0	0	-	-
logistics (42)	35	9	5	5	5	5	12	7	2
parcprinter (20)	20	20	20	15	15	15	20	20	20
pipesworld (50)	42	42	42	23	23	22	14	14	14
rovers (40)	40	33	22	24	24	21	39	33	27
satellite (36)	35	29	29	13	13	13	26	25	17
scanalyzer (20)	20	17	12	10	7	7	15	15	14
tetris (20)	19	19	19	0	-	-	1	1	1
tpp (30)	30	28	11	13	13	8	20	20	11
transport (20)	20	6	5	0	-	-	0	-	-
woodwork (20)	20	20	20	5	4	4	20	20	20
zenotravel (20)	20	20	20	16	16	16	20	20	16
ALL (460)	394	323	275	174	167	157	256	239	195

Table 1: Per domain solver and relaxation coverage. Values in brackets indicate the benchmark size. The **Plans** column indicates how many problems the respective planner solved. The **Solved** column indicates how many of the solved problems were successfully encoded and solved: every encoded problem was solvable by both MD and MR, and every MD encoding was solvable to completion. The **MR** column indicates the number of problems that were successfully encoded, and MR solved to completion.

When a problem could not be encoded, this was due to the large number of actions in the plan; typically plans with more than 200 actions caused an issue. In domains where this is problematic (e.g., elevators, logistics, and transport), we can see that the initial coverage for the non-sequential planners suffers as well. The problem with encoding plans that contain many actions is due to the number of transitivity clauses included for formula (10), which are cubic in the number of actions.

The tetris domain proved extremely difficult for POPF and Mp to solve, although the number of actions in the plans for Mercury were small enough to encode. Finally, we found that proving the optimality of the MR encoding for tpp and rovers was the most difficult, but there is no clear indication as to why: rovers has high *flex*, but not as high as other domains, and the opposite is true for tpp. In both domains, however, good initial plans were produced quickly by Sat4j, and the solver devoted the remaining time to making small improvements and proving optimality.

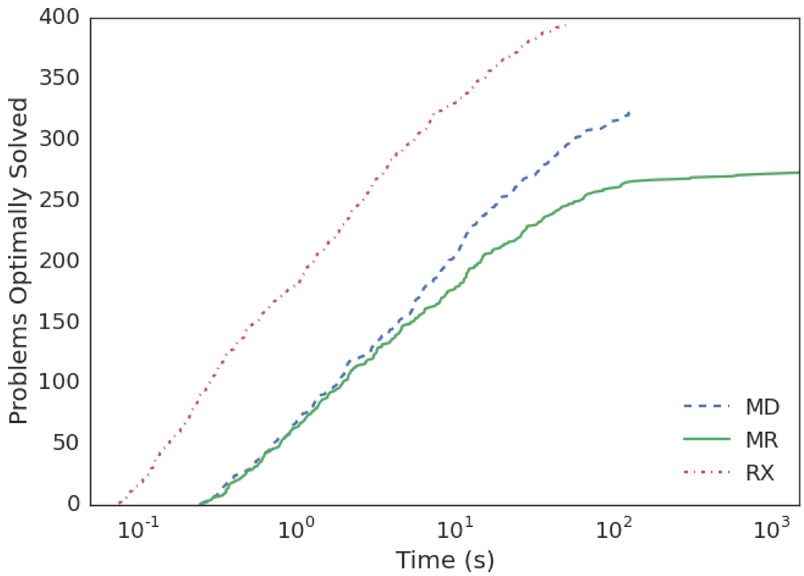


Figure 1: The number of problems solved to completion by Sat4j if given a limited amount of time per problem, as well as the number of problems solved by the RX algorithm. Every MD encoding was solved completely by Sat4j, and RX is a polynomial sub-optimal technique shown only for comparison of solve time.

Mercury solved a strict superset of the problems solved by POPF and Mp. As such, we use the sequential plans produced by Mercury as input for the majority of our evaluation (Section 4.3 is the one exception). Figure 1 provides a view on how long it took for Sat4j to optimally solve the MD and MR encodings from the initial plans Mercury produced: we show the number of problems solved optimally as a function of time (including the encoding phase). For comparison, we include the aggregate time for RX as well. The strong run-time performance of RX is to be expected given that it is a polynomial time algorithm without optimality guarantees.

4.2 Plan Quality

To begin, we discuss a surprising result for the Relaxer algorithm on the planning benchmarks. In every one of the 323 problems where Sat4j solved the MD encoding optimally, the POP that was produced with the Relaxer algorithm contained the same number of ordering constraints. Even though theoretically the Relaxer algorithm is not guaranteed to find a minimal POP, *it nonetheless computes a minimum deordering in every tested problem.* RX can produce only deorderings, and so this is the best RX could hope to achieve. Note that there may be many candidates for a minimum deordering, and RX does not necessarily find the same one that the MD encoding finds.

Next, we consider the difference in quality between the minimum deordering and minimum reordering. Quality is measured by the *flex* of the transitive closure of the generated POP, and we include only those problems where both the MD and MR encodings can be solved to completion by Sat4j (275 in total from the plans generated by Mercury). Table 2 shows the average *flex* for MD and MR in all domains, and Figure 2 shows the *flex* comparison on a per-problem basis over all domains.

Domain	MD	MR
airport	0.28	0.37
depot	0.31	0.36
driverlog	0.33	0.34
elevators	0.31	0.32
logistics	0.56	0.58
parcprinter	0.76	0.76
pipesworld	0.16	0.16
rovers	0.68	0.69
satellite	0.39	0.39
scanalyzer	0.31	0.31
tetris	0.49	0.50
tpp	0.37	0.38
transport	0.51	0.51
woodwork	0.96	0.96
zenotravel	0.32	0.32

Table 2: Average *flex*

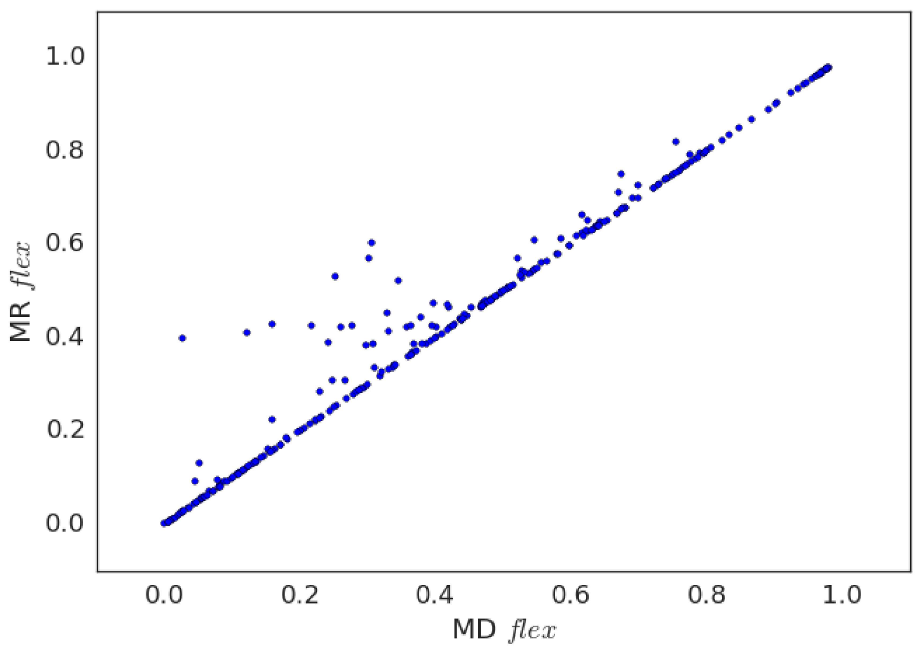


Figure 2: MD versus MR *flex* Comparison

Domains where we see substantial improvement include airport and depot. Domains that saw zero gain in terms of flexibility include parcprinter, transport, and woodwork. In total, almost one third (76/275) of the problems showed an improvement in *flex* for the MR over the MD by varying degrees.[10]

The *flex* value fails to convey the extreme amount of execution flexibility introduced by the relaxations. To investigate this further, we computed the number of linearizations for the plans wherever feasible. Determining the number of total orders for a partially ordered graph is #P-Complete (Brightwell & Winkler, 1991), and in practice it is difficult to compute precisely for many graphs. We were able to compute the number of linearizations for both the MD and MR solutions in a total of 203 problems where a solution to both encodings was computed. We found that approximately one quarter (51/203) showed a difference in the number of linearizations, and we plot the ratio #Linears(MR) / #Linears(MD) for these 51 problems in Figure 3.

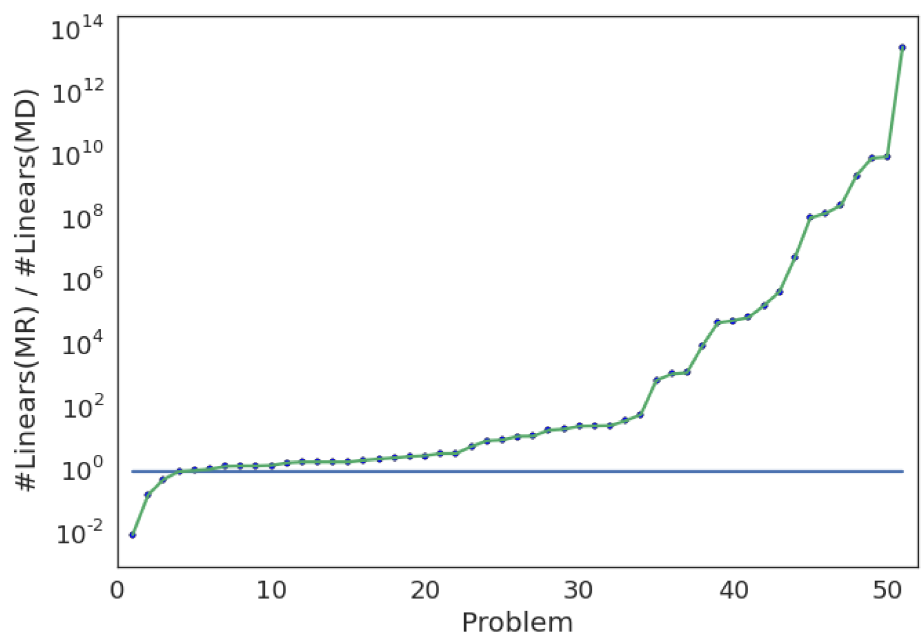


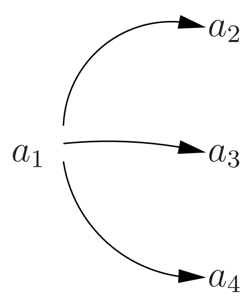
Figure 3: Ratio of Linearizations. The y-axis represents the number of linearizations induced by the POP for the optimal reordering divided by the number of linearizations induced by the POP for the optimal deordering. The x-axis ranges over all problems where the number of linearizations differed (∼25%), and is sorted based on the y-axis.

At its most extreme, the improvement in the number of linearizations can be massive; over 13 orders of magnitude in one airport problem. Conversely, we see an interesting artefact resulting from optimizing a metric which acts as a proxy for the number of linearizations: while the *flex* value of MR will never be lower than that of MD, the POPs produced by each approach using *flex* as an optimization criterion can have the opposite effect in the number of linearizations.

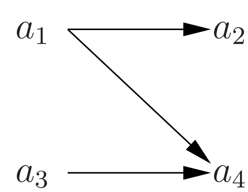
10. Many of the smaller improvements do not show up in the scatter plot.

In three problems (one from tetris and two from depot), we found that the number of linearizations in the POP produced from the MR encoding was *fewer* than the number of linearizations in the POP produced from the MD encoding. While the number of ordering constraints in a POP for a given number of actions is usually indicative of the number of linearizations for that POP, these three problems indicate that this is not a universal rule.

For a concrete example, consider two POPs on four actions $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$. Ignoring causal links, Figure 4 shows the structure of the POPs P_1 and P_2. Both POPs have the same number of actions and ordering constraints, but the number of linearizations differ: P_1 has 6 linearizations while P_2 only has 5. These POPs serve as a basic example of how the *flex* criterion does not capture fully the notion of POP flexibility that we use in our work. There may be similar notions that do take such differences into account, and they are left for future investigation (see Say, Cire, and Beck (2016) for some recent work in this direction).



(a) P_1



(b) P_2

Figure 4: Two POPs with the same number of actions and ordering constraints, but a different number of linearizations.

To demonstrate the correlation between a POP's *flex* and the number of linearizations, we focused on random partial orders for a POP with 20 actions (not including the special actionis a_I and a_G).[11] We constructed 10,000 random partial orders (8,959 of them unique) with a spread of *flex* value from 0.0 to 1.0, and subsequently we computed the corresponding number of linearizations in every POP. 100 POPs were constructed for each target *flex* value (taken in 0.01 increments), and the method of construction was to iteratively add new edges not present in the transitive closure until the POP reached the target *flex* value.

Qualitatively, the POPs resembled those found using planning techniques. The reason we use randomly generated plans is due to the number of examples required for the trend to present itself (comparing plans with a varying number of actions was uninformative). Figure 5 shows the *flex* as a function of the number of linearizations normalized by the total number of linearizations possible (in all POPs, this equals 20! which is roughly 2.4×10^{18}).

The Pearson correlation coefficient between the log of the normalized linearization count and the *flex* value is 0.991, and this clear trend ties together the *flex* of a POP and the number of linearizations. The red line in Figure 5 is the line-of-best-fit when using a log scole for the $linflex$ values (as the plot x-axis does). Interestingly, if we were to use this line as a

11. Similar results hold for random POPs with a different number of actions.

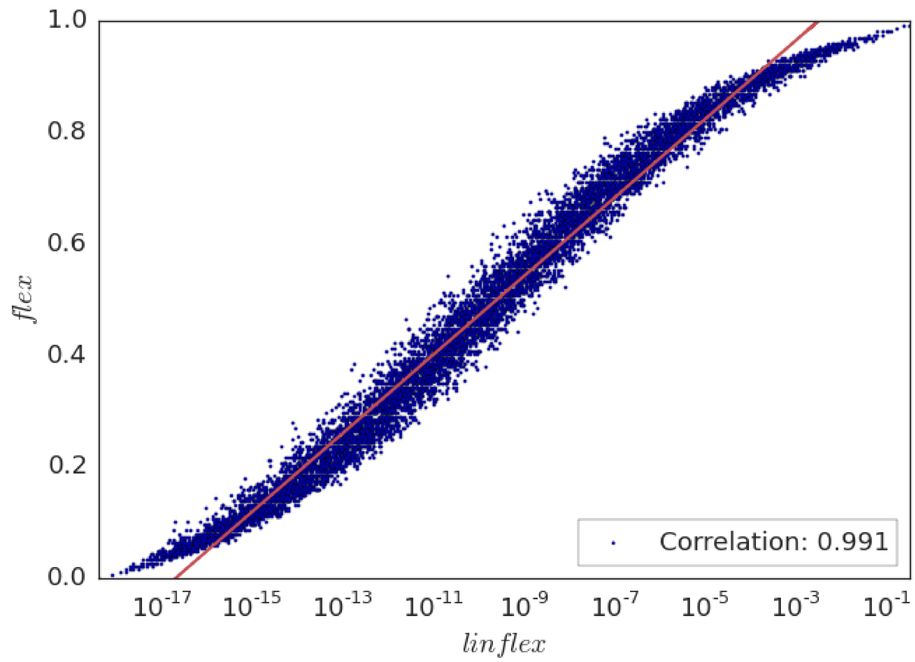


Figure 5: Comparison of the normalized number of linearizations and the *flex* value of approximately 10,000 random POPs with 20 actions. Every point represents a unique POP with 20 actions. The $linflex$ value is computed by normalizing the total number of linearizations by all those possible (20!), and note that the x-axis uses a log scale. The red line is the line-of-best-fit when using the log of $linflex$.

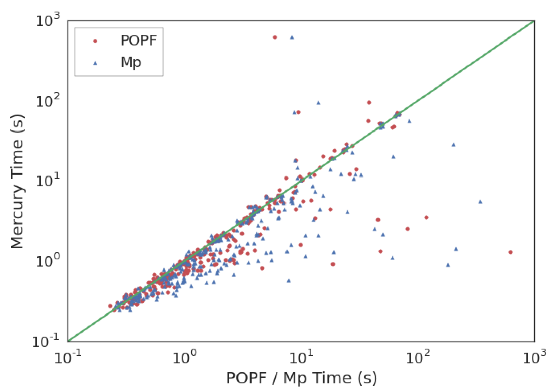
predictor for the number of linearizations, *flex* overestimates the number of linearizations of highly constrained plans and underestimates the number of linearizations for unconstrained plans. Though minimizing the number of ordering constraints in the transitive closure of a POP is not what we want to optimize directly, it does serve as a highly informative proxy for maximizing the number of linearizations for the POP.

4.3 Initial Plan Impact

Different planning techniques generate solutions of varying forms. While sequential planners are by far the most widely used, there are other planners that create inherently partially ordered solutions. For example, the POPF planner uses a forward-chaining approach that results in a partial-order plan that is represented as layered sets of unordered actions. Similarly, SAT-based planners such as Mp produce solutions that contain layers of unordered actions. The two approaches fundamentally differ in how they search for a solution, and are again fundamentally different from how a sequential planner searches. One question that arises from these differences is whether or not they lead to fundamentally different solutions; amenable to relaxing in different ways. We investigate the impact that the starting solution will have on the relaxed solution quality and the ability to compute an optimal solution.

The encoding for a minimum reordering does not take into account the original sequence of actions. Therefore, this encoding can be used without modification for the plans produced by POPF and Mp. In a similar sense, the plans produced by POPF and Mp can be encoded for a minimum deordering by carefully applying equation 14: $\mathcal{O}$ will include a link between every pair of actions that do not share the same layer. We obtained the layered plan representation from POPF and Mp directly using the appropriate planner settings.

Across all domains, 287 problems were mutually solved to completion by Sat4j using the solutions produced by all three planners and either the MD or MR encoding. Only 78 of those contained the same number of actions. Figure 6 shows the time that Sat4j required to solve the problem to completion for Mercury's plans measured against the plans for the other two planners. The first plot shows all 287 problems mutually solved, and we see a performance improvement for solutions coming from the Mercury solver.[12] However, when we limit ourselves to just the 78 problems that contain the same number of actions in all solutions, we find that the Sat4j solve-time is much more comparable to that of the other solvers. Thus, there appears to be little effect on the solving efficiency based on the input solution format. The primary factor in Sat4j solve time is the number of actions represented in the encoding.



(a) All problems mutually solved by Sat4j using a source plan from each planner.

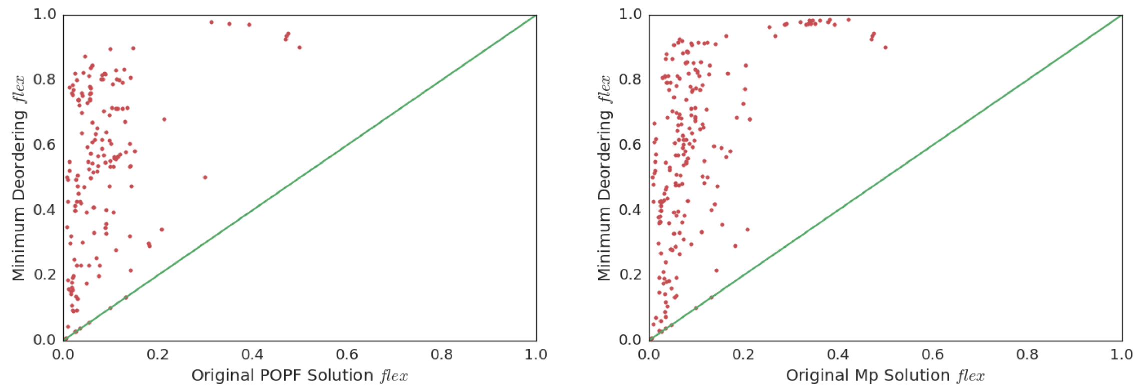
(b) The subset of mutually solved problems that contain the same number of actions.

Figure 6: Comparison of the time to relax a Mercury plan versus the time to relax a POPF or Mp plan. Both MD and MR encodings are included in the data.

In addition, we investigated the resulting *flex* of the produced POPs. Of the 78 problems mutually solved with the same number of actions, only two (from the scanalyzer domain) contained a different set of actions – these resulted in slightly higher *flex* values for the minimum deordering and reordering of Mercury's solution compared to the other planners. On the other hand, six problems from the airport domain had a lower *flex* value for the minimum deordering of Mp solutions despite having the same number of actions. This indicates that under some conditions, the initial layered plan produced by Mp may not allow for as much relaxation compared to a forward search planner such as Mercury or POPF. We should note, however, that in the vast majority of problems the *flex* from the minimum deordering or reordering coincided across all initial plan types.

12. Note that the time does *not* include initial planner computation; only the time to encode and solve the MaxSAT encoding.

Finally, we investigated the improvement in *flex* for each planner compared to its initial solution. For Mercury, the initial *flex* value is always 0, as it is a sequential planner. Because a reordering is allowed to ignore all of the original ordering constraints, we consider only the improvement in *flex* for the minimum deordering of plans coming from POPF and Mp. Figure 7 shows the relative *flex* comparison between the original plan and the minimum deordering that was computed. In these plots, we include every problem solved successfully to completion by Sat4j for plans produced by POPF (167) and Mp (219).



(a) *flex* improvement for POPF

(b) *flex* improvement for Mp

Figure 7: Comparison of the original plan *flex* versus the *flex* of the minimum deordering.

We found that the majority of initial *flex* values for both Mp and POPF solutions fell within the range of 0 to 0.2, and the difference in *flex* between the solutions from each solver was minimal. There was a moderate correlation between the original and final *flex* value: Mp and POPF solutions had a Pearson correlation coefficient of 0.57 and 0.42 respectively. However, we observed no distinction between relaxing Mp solutions versus those of POPF in either the time to compute the relaxation, or the *flex* of the final POP.

4.4 Comparison to MILP Encoding

The model for relaxing the ordering of a plan that is presented by Do and Kambhampati (2003) involves temporal constraints and resources – both are aspects beyond what we consider here. Nevertheless, a fragment of the model is capable of computing either the minimum deordering or reordering of the plan, and so it is worthwhile to see how effective it can be in finding an optimal reordering. We forgo testing the previous work for computing the optimal deordering, as the Relaxer Algorithm is so effective in doing so. We should note that Do and Kambhampati only considered using the model to heuristically guide the solver to a reasonable solution instead of an optimal one.

The optimization framework Do and Kambhampati use to model the problem of relaxing the ordering of a plan is Mixed Integer Linear Programming (MILP). A MILP consists of a set of linear constraints that are defined over variables that can take on integer or real values. The optimization criterion is specified as a weighted linear combination over a subset of the variables in the problem that should either be maximized or minimized. We do not need to go into further detail, as the MILP model presented in this section is quite basic and uses only integer variables for the encoding.

Here, we present a version of the MILP model introduced by Do and Kambhampati for comparison to our partial weighted MaxSAT model. The modifications fall under three categories: (1) fixes for bugs in the original formulation, (2) removal of variables and constraints not relevant to our setting (i.e., the temporal and resource related portions of the model), and (3) adding constraints to enforce that a solution is the transitive closure. The variables we use for the model include the following:

$$X^f_{a_j,a_i} = \begin{cases} 1 & \text{when } a_i \text{ supports } a_j \text{ with fluent } f \\ 0 & \text{otherwise} \end{cases}$$

$$Y^f_{a_i,a_j} = \begin{cases} 1 & \text{when } a_i \text{ is ordered before } a_j \text{ due to interference on fluent } f \\ 0 & \text{otherwise} \end{cases}$$

$$O_{a_i,a_j} = \begin{cases} 1 & \text{when } a_i \text{ is ordered before } a_j \\ 0 & \text{otherwise} \end{cases}$$

Note that $X^f_{a_j,a_i}$ and O_{a_i,a_j} are analogous to $\Upsilon(a_i, f, a_j)$ and $\kappa(a_i, a_j)$ respectively. The interference variables $Y^f_{a_i,a_j}$ are defined only for those cases where a_j can conflict with the execution of a_i on fluent f: either $f \in (PRE(a_i) \cup ADD(a_i)) \cap DEL(a_j)$ or $f \in (PRE(a_j) \cup ADD(a_j)) \cap DEL(a_i)$ holds. The constraints for the MILP model are as follows (unbound variables are assumed to be universally quantified).

- Interfering actions must be ordered (defined only for pairs of actions that interfere):

$$Y^f_{a_i,a_j} + Y^f_{a_j,a_i} = 1$$

- Every precondition is supported exactly one way:[13]

$$\forall f \in PRE(a_j), \quad \sum_{a_i \in adders(f)} X^f_{a_j,a_i} = 1$$

- Every support is threat free:

$$\forall a_d \in deleters(f), \quad (1 - X^f_{a_j,a_i}) + (Y^f_{a_d,a_i} + Y^f_{a_j,a_d}) \geq 1$$

- Support implies ordering:

$$O_{a_i,a_j} - X^f_{a_j,a_i} \geq 0$$

13. The original paper had this constraint erroneously listed as $\sum_{a_i \in adders(f)} X^f_{a_i,a_j} = 1$.

- Interference implies ordering:

$$O_{a_i,a_j} - Y^f_{a_i,a_j} \geq 0$$

- Enforce the transitive closure of ordering constraints:

$$(1 - O_{a_i,a_j}) + (1 - O_{a_j,a_k}) + O_{a_i,a_k} \geq 1$$

- Forbid self loops in the ordering:

$$O_{a,a} = 0$$

- Order everything after the initial state action and before the goal action:

$$O_{a_I,a} = 1$$
$$O_{a,a_G} = 1$$

The final three constraints do not appear in the original model. The last one replaces constraints that referenced temporal variables to achieve the same effect, and the first two ensure that a solution is transitively closed. As mentioned earlier, optimizing the transitive closure is preferred over optimizing the transitive reduction. Finally, the optimization criterion for the MILP model is as follows.

$$Minimize \sum_{a_1,a_2 \in \mathcal{A}} O_{a_1,a_2}$$

The above model will produce reorderings of the input plan as feasible solutions, and will find a minimum reordering if solved to completion. We implemented the MILP model using the state-of-the-art MILP solver Gurobi (version 5.6.2) (Gurobi Optimization, Inc., 2015), and measured the coverage over all domains as a function of time. Figure 8 contains the results.

We found that using the MILP model was effective for the easier problems (those solved in under 2 seconds), but for anything more difficult, solving the partial weighted MaxSAT encoding with Sat4j proved more efficient. Overall, 275 problems were solved using Sat4j on the partial weighted MaxSAT encoding while only 226 problems were solved using Gurobi on the MILP model.

We additionally tested a MILP model that mirrors the partial weighted MaxSAT encoding presented above. However, the results were very similar to those shown in Figure 8, with the MILP encoding being consistently outperformed for problems that take more than a second to solve.

5. Discussion

In this paper, we proposed a practical method for computing the optimal deordering and reordering of a sequential or partial-order plan. Despite the theoretical complexity of computing the optimal deordering or reordering being NP-hard, we are able to compute the

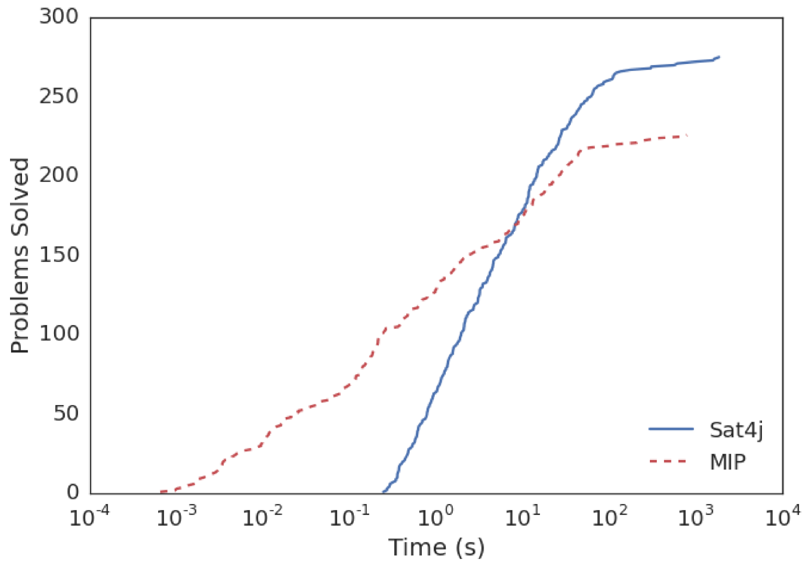


Figure 8: For a given timeout (x-axis), the number of problems solved to completion within that timeout bound (y-axis) by (1) Sat4j using the MR encoding and (2) Gurobi using the MILP encoding described in the text.

optimal solution by leveraging the power of modern MaxSAT solvers. We further proposed an extension to the classical least commitment criteria of minimal deordering and reordering: a minimum cost least commitment POP (MCLCP). An MCLCP considers the total cost of actions in a solution before minimizing the number of ordering constraints. Central to the encodings we propose is a notion of ordering relevance: we designed the optimization criteria to minimize the ordering constraints in the resulting plan, leaving only those that are relevant for plan validity.

Our approach uses a family of novel encodings for partial weighted MaxSAT, where a solution corresponds to an optimal POP satisfying one of the three least commitment criteria we investigate: minimum deordering, minimum reordering, and our proposed minimum cost least commitment POP. We solve the former two encodings with a state-of-the-art partial weighted MaxSAT solver, Sat4j, and find that the majority of problems are readily handled by the MaxSAT solver in a reasonable amount of time.

We considered various input plan formats, as well as a similar encoding for optimizing plan flexibility, and found that using a sequential plan as input to our encodings was the most effective solution for computing a reordering; perhaps surprisingly, there was no benefit observed when using a planner that naturally generates partial orders (Mp).

We also investigated an existing polynomial algorithm for deordering sequential plans and discovered that it successfully computes the optimal deordering in every problem we tested, despite its lack of theoretical guarantee. Because the algorithm is fast in practice, it is well suited for relaxing a POP if we require a deordering. Finally, we also established a strong empirical correspondence between the commonly used *flex* metric and the number of linearizations represented by a POP.

Here, we discuss related work and conclude with a discussion of potential future work.

5.1 Related Work

In Section 2.3, we detailed a variety of approaches that naturally produce partial-order plans. Here, we review other work related to aspects of our approach.

The standard SAT-based planning encodings also produce a POP (Kautz, McAllester, & Selman, 1996), but a significant difference between the standard encodings and our work is that we avoid encoding an action in every layer in a planning graph by appealing to the fact that we already know the (superset of) actions in the solution. Intuitively, we can view the encoding as using MaxSAT to find the implicit layers for the actions in our plan by way of computing the relevant ordering constraints. An additional difference is that choosing a layer for every action unnecessarily restricts the timing of that action when it can potentially appear in multiple adjacent layers.

The notion of MCLCP is related to that of plan repair (Nebel & Koehler, 1995; Gerevini & Serina, 2000). A key difference, however, is that we do not consider the *addition* of new actions – the cost of a plan is only improved by removing actions for MCLCP. As the focus of this paper is on improving the flexibility of POPs, we forgo a full theoretical and empirical comparison of the MCLCP criterion and the existing plan repair techniques. Preliminary results on the effect of MCLCP as an action removal technique can be found in our previous work on the subject (Muise et al., 2012; Muise, 2014).

Our core encoding is similar to the causal encodings of Kautz et al. (1996) and Variant-II of Robinson, Gretton, Pham, and Sattar (2010). We similarly encode the ordering between any pair of actions as a variable ($\kappa(a_i, a_j)$ in our case), but rather than encoding every potential action occurrence or modelling a relaxed planning graph, we encode the formulae that must hold for a valid POP on the specific set of actions provided as part of the input. As mentioned in Section 2.3, there are also similarities between our work and that of Do and Kambhampati (2003). In particular, the optimization criterion of minimizing the number of ordering constraints coincide, as does the optional use of constraints to force a deordering. While Do and Kambhampati focus on temporal relaxation in the context of action ordering, we take the orthogonal view of minimizing the total action cost.

5.2 Conclusion

The use of our method for computing optimally relaxed plans provides two key advantages: (1) if maximizing flexibility is paramount, then solving the MR encoding can lead to far more flexible solutions than the MD encoding or Relaxer Algorithm can achieve, and (2) the optimal deordering provides a useful baseline for demonstrating the effectiveness of the Relaxer Algorithm. Our work leaves open the possibility for a heuristic approach similar to the Relaxer Algorithm that is capable of producing reorderings of a partial-order plan.

One extension of our work is to consider alternative forms of optimization criteria. For example, one may change the soft clauses so as to minimize the number of fluents from the initial state that are required for plan validity. Doing so has the potential to improve planning formalisms that attempt to minimize the reliance on information about the initial state, such as assumption-based planning (Davis-Mendelow, Baier, & McIlraith, 2013). Alternatively, the initial set of actions need not correspond directly to a plan. As long as a subset of actions can achieve the goal, then we will compute a plan. This opens the door to techniques for optimizing plans by adding more actions to select from, using techniques such as those introduced by Davies, Pearce, Stuckey, and Søndergaard (2014).

Acknowledgments

The authors gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada (NSERC). Thanks also go to the anonymous reviewers for their thoughtful feedback during the review process.

Appendix A. Relaxer Counterexample

The Relaxer Algorithm presented in Section 2.3.4 deorders an input plan, but as pointed out by Bäckström (1998), the resulting POP may not be a minimal deordering. The counterexample provided by Bäckström, however, incorrectly states that the resulting POP is not minimally deordered (Bäckström, 1998, Figure 14), when in fact Figure 14(b) is not a deordering of 14(a), and thus 14(a) is a minimal deordering (although not a minimum reordering). Here, we present a new counterexample that supports the claim that the Relaxer Algorithm may not produce a minimum deordering.

Both the domain theory and problem specification are shown in Figure 9. The input plan is the sequence of actions $[a_1, a_2, a_3]$. Because the Relaxer Algorithm seeks out the earliest achiever for every precondition, the algorithm results in a deordering of the plan that has two ordering constraints: $(a_1 \prec a_3)$ and $(a_2 \prec a_3)$. The problem with the deordering is that a_1 is chosen as the achiever for the fluent p, when in fact a_2 can be used as the achiever for both p and q (note that a_2 is already required for fluent q).

The weakness of the Relaxer Algorithm is that it uses the earliest achiever. This weakness surfaces when an action later in the plan can be used as an achiever is already ordered appropriately. Using this insight, there may be a modification of the Relaxer Algorithm that finds achievers already ordered appropriately, as opposed to finding the earliest achiever.

```
(define (domain counterexample)
  (:requirements :strips)
  (:predicates  (p) (q) (g1) (g2) (g3) )

(:action a1
  :parameters()
  :precondition ()
  :effect (and (g1) (p)))

(:action a2
  :parameters()
  :precondition ()
  :effect (and (g2) (p) (q)))

(:action a3
  :parameters()
  :precondition (and (p) (q))
  :effect (and (g3))))

(define (problem counterexample-problem)
   (:domain counterexample)
   (:init ())
   (:goal (and (g1) (g2) (g3) )))
```

Figure 9: Counterexample Domain and Problem Description

References

Anderson, J. S., & Farley, A. M. (1988). Plan abstraction based on operator generalization. In *7th International Conference on Artificial Intelligence*, pp. 100–104.

Bäckström, C. (1998). Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, *9*(1), 99–137.

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (2009). *Handbook of satisfiability, frontiers in artificial intelligence and applications*. IOS Press.

Brightwell, G., & Winkler, P. (1991). Counting Linear Extensions is #P-Complete. *23rd Annual ACM Symposium on Theory of Computing*, *8*(3), 175–181.

Coles, A., & Coles, A. (2016). Have I Been Here Before? State Memoization in Temporal Planning. In *26th International Conference on Automated Planning and Scheduling*, pp. 97–105.

Coles, A., Coles, A., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning. In *20th International Conference on Automated Planning and Scheduling*, pp. 42–49.

Coudert, O. (1996). On solving covering problems. In *33rd Annual Design Automation Conference*, pp. 197–202.

Davies, J., & Bacchus, F. (2013). Postponing optimization to speed up MAXSAT solving. In *19th International Conference on Principles and Practice of Constraint Programming*, pp. 247–262.

Davies, T. O., Pearce, A. R., Stuckey, P. J., & Søndergaard, H. (2014). Fragment-based planning using column generation. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, pp. 83–91.

Davis-Mendelow, S., Baier, J. A., & McIlraith, S. A. (2013). Assumption-based planning: Generating plans and explanations under incomplete knowledge. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pp. 209–216.

Do, M. B., & Kambhampati, S. (2003). Improving the temporal flexibility of position constrained metric temporal plans. In *AIPS Workshop on Planning in Temporal Domains*.

Domshlak, C., Hoffmann, J., & Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, *221*, 73–114.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, *3*(1), 251–288.

Gerevini, A., & Serina, I. (2000). Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*, pp. 112–121.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers.

Graham, J. R., Decker, K. S., & Mersic, M. (2001). DECAF - A Flexible Multi Agent System Architecture. *Autonomous Agents and Multi-Agent Systems*, *7*(1), 7–27.

Gurobi Optimization, Inc. (2015). Gurobi optimizer reference manual..

Hickmott, S., Rintanen, J., Thiébaux, S., & White, L. B. (2007). Planning via petri net unfolding. In *20th International Joint Conference on Artificial Intelligence*, pp. 1904–1911.

Hickmott, S., & Sardina, S. (2009). Optimality properties of planning via Petri net unfolding: A formal analysis. In *19th International Conference on Automated Planning and Scheduling*, pp. 170–177.

Hickmott, S. L. (2008). *Directed unfolding: reachability analysis of concurrent systems & applications to automated planning.* Ph.D. thesis, University of Adelaide.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*(1), 253–302.

Hoffmann, J. (2016). ICAPS competition page. `http://ipc.icaps-conference.org/`. Accessed: 2016-09-06.

Kambhampati, S., & Kedar, S. (1994). A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence*, *67*(1), 29–70.

Kautz, H. A., McAllester, D. A., & Selman, B. (1996). Encoding plans in propositional logic. In *5th International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 374–384.

Kautz, H. A., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In *16th International Joint Conference on Artificial Intelligence*, pp. 318–325.

Le Berre, D., & Parrain, A. (2010). The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, *7*, 59–64.

McAllester, D. A., & Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pp. 634–639.

Muise, C. (2014). *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution*. Ph.D. thesis, University of Toronto.

Muise, C., Mcilraith, S. A., & Beck, J. C. (2011). Optimization of partial-order plans via MaxSAT. In *ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, COPLAS.

Muise, C., McIlraith, S. A., & Beck, J. C. (2012). Optimally relaxing partial-order plans with MaxSAT. In *22nd International Conference on Automated Planning and Scheduling*, pp. 358–362.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Nebel, B., & Bäckström, C. (1994). On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence*, *66*(1), 125–160.

Nebel, B., & Koehler, J. (1995). Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, *76*(1-2), 427–454.

Nguyen, X., & Kambhampati, S. (2001). Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pp. 459–466.

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

Robinson, N., Gretton, C., Pham, D. N., & Sattar, A. (2010). Partial weighted MaxSAT for optimal planning. In *11th Pacific Rim International Conference on Artificial Intelligence*, pp. 231–243.

Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: a modern approach*. Prentice hall.

Say, B., Cire, A. A., & Beck, J. C. (2016). Mathematical programming models for optimizing partial-order plan flexibility. In *22nd European Conference of Artificial Intelligence (In Press)*.

Siddiqui, F. H., & Haslum, P. (2012). Block-structured plan deordering. In *Australasian Conference on Artificial Intelligence*, pp. 803–814.

Tate, A. (1976). Project planning using a hierarchic non-linear planner. In *D.A.I. Research Report No. 25*. Department of Artificial Intelligence, University of Edinburgh.

Veloso, M. M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for planning in dynamic environments. In *4th International Conference on Artificial Intelligence Planning Systems*, pp. 171–180.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Younes, H. L. S., & Simmons, R. G. (2003). VHPOP: versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, *20*, 405–430.