Block-Structured Plan Deordering

Fazlul Hasan Siddiqui and Patrik Haslum

The Australian National University & NICTA Optimisation Research Group,
Canberra, Australia

Abstract. Partially ordered plans have several useful properties, such as exhibiting the structure of the plan more clearly which facilitates post-plan generation tasks like scheduling the plan, explaining it to a user, or breaking it into subplans for distributed execution. The standard interpretation of partial ordering implies that whenever two subplans are unordered, every interleaving of steps from the two forms a valid execution. This restricts deordering to cases where individual steps (i.e., actions) are independent. We propose a weaker notion of partial ordering that divides the plan into *blocks*, such that the steps in a block may not be interleaved with steps outside the block, but unordered blocks can be executed in any sequence. We present an algorithm to find such deorderable blocks, and show that it enables deordering plans in many cases where no deordering is possible under the standard interpretation.

1 Introduction

In AI planning, the process of *deordering* converts a sequential plan into a partially ordered plan, by removing ordering constraints between steps, such that the steps of the plan can be successfully executed in any order consistent with the partial order and still achieve the goal [1]. Partially ordered plans have two advantages: first, they afford execution flexibility, thus allowing plans to be scheduled for improved efficiency or robustness [2]; and second, they make the plan structure more accessible, which facilitates further analysis of the plan. However, current state space search planners, which produce totally ordered sequential plans, are far more efficient than the older partial order planners. Thus, deordering plays a useful role in that it enables more efficient generation of partially ordered plans.

The standard interpretation of a partially ordered plan is that it is valid if and only if every sequential plan that is a topological sort of the steps is valid (according to the semantics of sequential plan execution). This implies that for two subplans to be unordered, every interleaving of steps from the two must form a valid execution. This restricts deordering to only the cases where individual steps (i.e., actions) are independent and non-interfering. We examine a weaker notion of partial ordering: We divide a plan into *blocks*, such that the steps in a block may not be interleaved with steps outside the block, but unordered blocks can be executed in any sequence. The difference is illustrated in Figure 1. The restriction to non-interleaved executions allows “transient” dependencies

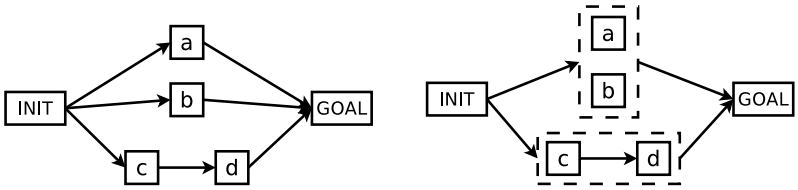

Fig. 1. A normal partially ordered plan (left) represents the set of all sequential plans that are topological sorts of the plan steps, such as *abcd*, *acbd*, *cbda*, etc. A block ordered plan (right) allows unordered blocks (shown with dashed outlines) to be executed in any order, but not steps from different blocks to be interleaved. Thus, *abcd*, *bacd*, *cdab* and *cdba* are the only linearisations of this plan.

and effects to be encapsulated within a block, and thus not cause interference with other blocks. This can enable deordering of blocks in some cases also when their constituent actions cannot be deordered under step-wise interpretation of partially ordered plans. (An example of this is shown in Figure 2.) We also present an algorithm to find block deorderings of plans, and show empirically that it deorders plans to a much greater degree than step-wise deordering.

Since unordered blocks cannot, in general, be executed concurrently, block deordering improves mainly on the second advantage of partial ordering, viz. making the structure of the plan explicit. We believe that the main benefit of this will be to facilitate post-plan generation tasks such as explaining the plan to a user, or breaking it into a set of subplans with minimal constraints between them, which is useful for reducing the coordination overhead if the plan is to be executed by a distributed team of agents [3], or for improving plan quality by local modifications [4].

2 Plans, Validity and Deordering

AI planning is model-based: a planner takes as input a description of available actions, in some formal modelling language, and the initial state and goal. Several modelling languages are in use (see, e.g., [5]). Because many details of the modelling language are not relevant for the purpose of plan deordering, we adopt Bäckström's [1] *producer-consumer-threat* model, which is general enough to encompas most common planning formalisms (e.g., STRIPS and SAS+).

A planning problem is defined over a set of *atomic propositions*. An action that makes a proposition p true is called a *producer* of p; an action that requires p to hold is called a *consumer* of p, and p is called a *precondition* of the action; and an action that makes p false is called a *threat* to p. (In the common propositional STRIPS formalism, producers are actions that add p and threats are actions that delete p.) A partially ordered plan is a set, S, of *steps*, where each step $s \in S$ is labelled by an action, act(s), and a strict (i.e., irreflexive) partial order $\prec$ over S. There is one *final step*, s_G in S, which represents the goal. If necessary,

we can also include in S an initial step, which acts as the producer of initially true propositions. We use the terms producer, consumer and threat also for plan steps, referring to their associated actions. We use $\prec^+$ to denote the transitive closure of a partial order $\prec$. An element $(s_i, s_j) \in \prec$ (also $s_i \prec s_j$) is a *basic ordering constraint* iff it is not implied by other constraints and transitivity, that is, iff $(\prec - \{s_i \prec s_j\})^+ \subset \prec^+$. A *linearisation* of $\prec$ is a strict total order that contains $\prec$ (i.e., a topological sort). A partially ordered plan $(S, \prec)$ is *valid* iff for every step $s \in S$ and for every precondition p of $\mathrm{act}(s)$, there is a producer s_p of p that precedes s (i.e., $s_\mathrm{p} \prec^+ s$) and for every step s_t that threatens p, s_t is either ordered before s_p or after s (i.e., $s_\mathrm{t} \prec^+ s_\mathrm{p}$ or $s \prec^+ s_\mathrm{t}$). This is essentially Chapman's [6] modal truth criterion (without "white knights"), and equivalent to the standard notion that a partial order plan is valid iff every linearisation of it is valid under the usual sequential execution semantics [5].

There are three possible reasons for an ordering constraint $s_i \prec s_j$: (1) Step s_i produces a proposition p that s_j consumes. This relation is usually called a *causal link* from s_i to s_j [7]. (2) s_i threatens a proposition that s_j produces, and that is consumed by some later step. Note that it is not necessary to order a producer and threat if no step that may occur after the producer in the plan depends on the produced proposition. (3) s_j threatens a proposition that s_i consumes. It is easy to see that if every precondition is supported by a causal link, and no causal link is threatened by a possibly intervening step, the plan is valid in the sense defined above. We will use the labels $\mathrm{PC}(p)$ (producer–consumer of p), $\mathrm{TP}(p)$ (threat–producer) and $\mathrm{CT}(p)$ (consumer–threat) to denote the three reasons. Note that an ordering constraint can have several associated reasons (including several reasons of the same type but referring to different propositions). We denote the set of reasons for an ordering constraint $s_i \prec s_j$ by $\mathrm{Re}(s_i \prec s_j)$.

Let $(S, \prec)$ be a valid partially ordered plan. A (step-wise) *deordering* of the plan is a valid plan $(S, \prec')$ such that $(\prec')^+ \subset \prec^+$. That is, a deordering is the result of removing some basic ordering constraints without invalidating the plan. Deordering a sequential plan is simply a special case, since a total order is a special case of a partial order. Several algorithms for plan deordering have been proposed [8–13]. The complexity of optimal deordering depends on the planning formalism and the measure of optimality. To compute a deordering with a smallest (w.r.t. cardinality) ordering relation is NP-hard for almost every planning formalism [1]. We use a combination of two simple methods: Chrpa & Bartak's algorithm [14] for computing causal links (what they call "dependency"), with the difference that our version selects the earliest unthreatened producer whereas theirs selects the latest, and the PRF algorithm [1] for computing threat–producer and consumer–threat orderings. This procedure is essentially the same as the algorithm by Kambhampati & Kedar [11]. Although it does not guarantee optimality, a recent study found that it did produce optimal deorderings of all plans on which it was tested [13].

3 Block Decomposition and Deordering

3.1 Blocks and their Semantics

A *block* is a part of the plan, i.e., a subset of steps, that must be executed without being interleaved with any step not in the block. Thus, there cannot be a step not in the block that is ordered in between two steps of the block.[1] The no-interleaving restriction affords us a simplified,"black box", view of blocks, in which only the preconditions and effects of executing the block as a whole are important. Thus, for the purpose of deordering we can ignore some dependencies and effects that matter only internally within the block.

A decomposition of a plan into blocks can be recursive, i.e., a block can be a strict subset of another block. However, blocks cannot be partially overlapping.

Definition 1. *Let $(S, \prec)$ be a partially ordered plan. A* block w.r.t. $\prec$ *is a subset $b \subset S$ of steps such for any two steps $s, s' \in b$, there exists no step $s'' \in (S - b)$ such that $s \prec^+ s'' \prec^+ s'$ or $s' \prec^+ s'' \prec^+ s$. A set B of subsets of S is a* block decomposition *of $(S, \prec)$ iff (1) each $b \in B$ is a block w.r.t. $\prec$ and (2) for every $b_i, b_j \in B$, either $b_i \subset b_j$, $b_j \subset b_i$, or b_i and b_j are disjoint.*

We omit the reference to the ordering relation when it is clear from context. A set consisting of a single step is always a block, but we do not consider such "trivial blocks" explicitly in the decomposition.

Formally, the semantics of a partially ordered block decomposed plan are defined by restricting its linearisations to those that respect the block decomposition, i.e., that do not interleave steps from disjoint blocks. The plan is defined to be valid iff every linearisation of it is valid, in the sense defined earlier.

Definition 2. *Let $(S, B, \prec)$ be a partially ordered and block decomposed plan. A* linarisation *of $(S, B, \prec)$ is a total order $\prec_{lin}$ on S such that (1) $\prec \subseteq \prec_{lin}$ and (2) every $b \in B$ is a block w.r.t. $\prec_{lin}$.*

The set of linearisations of a partially ordered and block decomposed plan is a subset of the linearisations under the same partial ordering without block decomposition, as illustrated by the example in Figure 1.

As mentioned above, the restriction on exections of a block decomposed plan allows us to ignore some dependencies and effects that matter only within the block. The following definition captures those preconditions and effects that are visible from outside the block, i.e., those that give rise to dependencies or interference with other parts of the plan. These are what we need to consider when deciding if two blocks can be unordered.

Definition 3. *Let $(S, B, \prec)$ be a partially ordered and block decomposed plan, and $b \in B$ a block:*

[1] Chrpa & Bartak [14] define the same concept, but call it a "subplan". We use the term "block" to emphasize the contiguous, "black box", nature of them, and to leave the term "subplan" free to refer to any part of a plan.

- b consumes p *iff there is a step* $s \in b$ *that consumes* p *such that there is no step* $s' \in b$ *with* $s' \prec^+ s$ *that produces* p *and for which either* $s'' \prec^+ s'$ *or* $s \prec^+ s''$ *holds for any* $s'' \in b$ *that threatens* p*, (In other words, there is a consumer* s *of* p *in the block, and there is no producer of* p *within the block from which we could draw a causal link to* s *that is not threatened by any step in the block.)*
- b produces p *iff there is a step* $s \in b$ *that produces* p *such that for any step* $s' \in b$ *that threatens* p*,* $s' \prec^+ s$*, and* b *does not consume* p*.*
- b threatens p *iff there is a step* $s \in b$ *that threatens* p *such that there is no step* $s' \in b$ *with* $s \prec^+ s'$ *that produces* p*.*

Whenever a block consumes, produces or threatens a proposition, there is at least one step within the block that does the same. We refer to this as the *responsible step*, and it plays an important role in the block deordering algorithm.

Note that if a block consumes a proposition, it cannot also produce the same proposition. The reason for this is that taking the "black box" view of block execution, the proposition simply persists: it is true before execution of the block begins and remains true after it has finished. If the steps within a block are totally ordered, the preconditions and effects of a block according to Definition 3 are nearly the same as the "cumulative preconditions and effects" of an action sequence, defined by Haslum & Jonsson [15], the only difference being that a consumer block cannot also be a producer of the same proposition.

3.2 Block Deordering

The process of block deordering is more complicated than standard deordering, which only involves removing constraints from the ordering relation. A block deordering involves adding new blocks to a plan decomposition, removing ordering constraints, and possibly also adding some explicit ordering constraints that were transitively implied by the removed constraints.

Let $(S, B, \prec)$ be a valid partially ordered and block decomposed plan. Consider a basic ordering constraint $s_i \prec s_j$, and the set $\mathrm{Re}(s_i \prec s_j)$ of reasons for this constraint. (We consider only basic ordering constraints, since removing a transitively implied constraint does not lead to any de facto deordering of the plan.) To remove $s_i \prec s_j$, we create two corresponding blocks, b_i and b_j, where s_i is the unique last step in b_i (that is, every step in b_i is transitively ordered before s_i) and s_j is the unique first step in b_j (that is, every step in b_j is transitively ordered after s_j). Note that one of the two blocks can be trivial, i.e., consist of a single action. Both blocks must be consistent with the existing decomposition, i.e., $B \cup \{b_i, b_j\}$ must still be a valid block decomposition, in the sense of Definition 1. We seek blocks that allow us to remove reasons from $\mathrm{Re}(s_i \prec s_j)$. Therefore, conditions on the blocks depend on what those reasons are:

- If $\mathrm{PC}(p) \in \mathrm{Re}(s_i \prec s_j)$, b_i must not produce p. Since s_i produces p and is the last step in b_i, and thus cannot be possibly followed by a threat to p within the block, this means b_i must consume p. Since the plan is valid, there must be some (unthreatened) producer, s', that necessarily precedes the step in b_i

that consumes p. If s' can precede every step in b_i, then adding the causal link $\mathrm{PC}(p)$ to $\mathrm{Re}(s' \prec s_j)$ (adding (s', s_j) to $\prec$ if not already present) allows $\mathrm{PC}(p)$ to be removed from $\mathrm{Re}(s_i \prec s_j)$. (Note that the added ordering constraint, even if not already explicitly in $\prec$, is transitively implied by $s_i \prec s_j$ and the conditions on b_i.)

- If $\mathrm{TP}(p) \in \mathrm{Re}(s_i \prec s_j)$, then b_j must include every step s' such that $\mathrm{PC}(p) \in \mathrm{Re}(s_j \prec s')$. Then $\mathrm{TP}(p)$ can be removed from $\mathrm{Re}(s_i \prec s_j)$.
- If $\mathrm{CT}(p) \in \mathrm{Re}(s_i \prec s_j)$, then either b_i must not consume p or b_j must not threaten p. Then $\mathrm{CT}(p)$ can be removed from $\mathrm{Re}(s_i \prec s_j)$.

The ordering $s_i \prec s_j$ can exist for several reasons (including several reasons of the same type, referring to different propositions). Only if blocks b_i and b_j can be found that meet the conditions above to remove every reason in $\mathrm{Re}(s_i \prec s_j)$ can the ordering be removed. Yet, even this does not guarantee the blocks will be unordered. If b_i contains some step other than s_i that is ordered before a step in b_j (s_j or another), the two blocks will still be ordered. Note that we must also check for new threats between steps in or before b_i and steps in or after b_j that become unordered as a result of removing $s_i \prec s_j$.

Theorem 1. *Deordering according to the rules above preserves plan validity.*

Proof. Let $(S, B, \prec)$ be a valid partially ordered and block decomposed plan, $s_i \prec s_j$ a basic ordering constraint, and b_i, b_j blocks that meet the conditions for removing $s_i \prec s_j$, and that are not ordered for any other reason. Let $(S, B', \prec')$ be the plan that results from deordering. Any linearisation of $(S, B', \prec')$ in which b_i precedes b_j is also a linearisation of $(S, B, \prec)$, and thus valid by assumption. Consider a linearisation in which b_j precedes b_i:

$$s_1, \ldots, s_m, b_j = [s_j, \ldots, s_{k_j}], s_{k_j+1}, \ldots, b_i = [s_{k_i}, \ldots, s_i], \ldots, s_n.$$

We examine each of the possible reasons for $s_i \prec s_j$: If $\mathrm{PC}(p) \in \mathrm{Re}(s_i \prec s_j)$, then the precondition p of step s_j is now supplied by the step s' (which is one among $s_1, \ldots, s_m$). b_j cannot threaten the causal link for p from s' to the step s'' in b_i that consumes p, since for it to do so, there must be some step $s \in b_j$ that threatens p, which would imply $\mathrm{CT}(p) \in \mathrm{Re}(s'' \prec s)$, making this linearisation inconsistent. Neither can any of the steps between b_j and b_i threaten p, since they can appear between s' and s'' also in a linearisation of the original plan.

If $\mathrm{TP}(p) \in \mathrm{Re}(s_i \prec s_j)$, then b_j includes every step s' such that $\mathrm{PC}(p) \in \mathrm{Re}(s_j \prec s')$. Thus, s_i does not threaten any causal link originating in s_j.

If $\mathrm{CT}(p) \in \mathrm{Re}(s_i \prec s_j)$, there are two possibilities: either b_i does not consume p or b_j does not threaten p. In the first case, this means that b_i constains a step s' that produces p, such that s' precedes s_i and the causal link is not threatened by any step in b_i; this means that the causal link is also unthreatened in the above linearisation, since no steps not in b_i appear between s' and s_i. In the second case, since b_j includes s_j, which does threaten p, b_j must also include a step s' that produces p and that is ordered after any step in b_j that threatens p. Since the original plan is valid, there is a step s'' that supplies an unthreatened causal

link for p to s_i. Again, there are two cases: If s'' is one of the steps $s_1, \ldots, s_m$, then none of the steps between b_j and b_i can threaten p, since these steps can appear between s'' and s_i also in the original plan. Thus, we can now form an unthreatened causal link from the step s' in b_j to s_i. Otherwise, s'' is one of the steps between b_j and b_i, in which case the causal link from the original plan remains unthreatened. □

3.3 The Block Deordering Algorithm

The previous subsection described the conditions under which block deordering is correct, in the sense that it preserves plan validity. Next, we describe the algorithm that we use to efficiently find block deordering possibilities in a plan.

We extend ordering to blocks: two blocks are ordered $b_i \prec b_j$ if there exist steps $s_i \in b_i$ and $s_j \in b_j$ such that $s_i \prec s_j$ and neither block is contained in the other (i.e., $b_i \not\subset b_j$ and $b_j \not\subset b_i$). In this case, all steps in b_i must precede all steps in b_j in any linerarisation of the block decomposed plan. We also extend the reasons for ordering (PC, TP and CT) to ordering constraints between blocks, with the set of propositions produced, consumed and threatened by a block given by Definition 3. Recall that a responsible step is a step in a block that causes it to produce, consume or threaten a proposition. For example, if b produces p, there must be a step $s \in b$ that produces p, such that no step in the block not ordered before s threatens p; we say step s is "responsible" for b producing p.

The core of the algorithm is the RESOLVE procedure (Algorithm 1). It takes as input two blocks, b_i and b_j, that are ordered (one or both blocks may consist of a single step), and tries to break the ordering by extending them to larger blocks, b'_i and b'_j. The procedure examines each reason for the ordering constraint and extends one of the blocks to remove that reason, following the rules given in the previous subsection. After this, the sets of propositions produced, consumed and threatened by the new blocks (b'_i and b'_j) are recomputed (following Definition 3) and any new reasons for the ordering constraint that have arisen because of steps that have been included are added to $\mathrm{Re}(b'_i \prec b'_j)$. This is repeated until either no reason for the ordering remains, in which case the new blocks returned by the procedure can safely be unordered, or some reason cannot be removed, in which case deordering is not possible (signalled by returning NULL). The function INTERMEDIATE(b_i, b_j) returns the set of steps ordered between b_i and b_j, i.e., $\{s \mid b_i \prec^+ s \prec^+ b_j\}$. Where Algorithm 1 refers to a "nearest" step s' preceding or following another step s, it means a step with a smallest number of basic ordering constraints between s' and s.

Applying the RESOLVE procedure to each basic ordering constraint we obtain a collection of blocks with which we can break some orderings. But this collection is not necessarily a valid decomposition, since some of the blocks may have partial overlap. To find a valid decomposition, we use a greedy procedure with some heuristic enhancements. We repeatedly examine each basic ordering constraint $b_i \prec b_j$ and call RESOLVE to find two extended blocks $b'_i \supseteq b_i$ and $b'_j \supseteq b_j$ that allow the ordering to be removed. In each iteration, constraints are checked in order from the beginning of the plan. If such blocks are found, and they are

Algorithm 1 Resolve ordering constraints between a pair of blocks.

```
1: procedure RESOLVE(b_i, b_j)
2:     Initialise b'_i = b_i, b'_j = b_j.
3:     while Re(b'_i ≺ b'_j) ≠ ∅ do
4:         for each r ∈ Re(b'_i ≺ b'_j) do
5:             if r = PC(p) then
6:                 Find a responsible step s ∈ b'_i and a nearest s' ∉ b'_i that consumes
                   p such that s' ≺+ s.
7:                 if such s' exists then
8:                     Set b'_i = b'_i ∪ {s'} ∪ INTERMEDIATE(s', b'_i).
9:                 else
10:                    return NULL
11:            else if r = TP(p) then
12:                Find a responsible step s ∈ b'_j and a nearest s' ∉ b'_j that threatens p
                   such that s ≺+ s'.
13:                if such s' exists then
14:                    Set b'_j = b'_j ∪ {s'} ∪ INTERMEDIATE(b'_j, s').
15:                else
16:                    return NULL
17:            else if r = CT(p) then
18:                Find a responsible step s ∈ b'_j and a nearest s' ∉ b'_j that produces p,
                   such that s ≺+ s'.
19:                if such s' exists then
20:                    Set b'_j = b'_j ∪ {s'} ∪ INTERMEDIATE(b'_j, s').
21:                else
22:                    Find a responsible step s ∈ b'_i and a nearest s' ∉ b'_i that produces
                       p, such that s' ≺+ s.
23:                    if such s' exists then
24:                        Set b'_i = b'_i ∪ {s'} ∪ INTERMEDIATE(s', b'_i).
25:                    else
26:                        return NULL.
27:        Recompute Re(b'_i ≺ b'_j).
28:    return (b'_i, b'_j).
```

consistent with the current decomposition, b_i and b_j are replaced. If b'_i or b'_j cannot be added to the decomposition (because one or both of them partially overlaps with an existing block), we consider all blocks ordered immediately after b_i, and check if all these orderings can be broken simultaneously, using the union of the blocks returned by RESOLVE for each ordering constraint. (Symmetrically, we also check the set of blocks immediately before b_j, though this is only very rarely useful.) As an additional heuristic, we discard the two blocks if there is a basic ordering constraint between a step that is internal to one of the blocks (i.e., that has both preceding and following steps within the block) and a step outside the block. If either possibility leads to a valid new decomposition, the ordering is removed. The inner loop then exits and the ordering relation is updated with any new constraints between b'_i and blocks ordered after b_j and between b'_j and

blocks ordered before b_i. This is done by checking for the three reasons (PC, TP and CT) based on the sets of propositions produced, consumed and threatened by b'_i and b'_j. The inner loop is then restarted, with ordering constraints that previously could not be broken checked again. This is done because removing ordering constraints can make possible the resolution of other constraints, since removal of orderings can change the set of steps intermediate between two steps.

The main loop repeats until no further deordering consistent with the current decomposition is found. It is easy to verify that each iteration runs in polynomial time, but we currently do not have an upper bound on the number of iterations. Note, however, that the procedure is "any time", in the sense that if interrupted before running to completion, the result at the end of the last completed iteration is still a block deordering of the plan. Since the choice of deordering to apply is greedy, the result is not guaranteed to be optimal.

4 Results

We tested block deordering on a large set of plans generated by planners participating in past editions of the International Planning Competition (IPC). To measure the effect of deordering, we compare the *flexibility* [2], or "flex", of plans, after standard, step-wise, deordering (as described in Section 2) and after block deordering. The flex of a partially ordered plan is defined as the fraction of pairs of steps that are not (transitively) ordered. Thus, a higher flex value indicates a less strictly ordered plan, with a fully sequential plan having a flex of zero. We apply the same definition to block deordered plans. Recall that in a block deordered plan, all steps belonging to two ordered blocks are ordered by the requirement that blocks not be interleaved, even when there is no ordering between an individual pair of steps. This is taken into account when calculating the flex of a block deordered plan. Because of this, it is in fact possible for block deordering of a partially ordered plan to decrease its flex value. For example, assume the plan on the left in Figure 1 also had the ordering constraints $a \prec d$ and $b \prec d$, and that the block decomposition on the right removed only the first of these: it would then only have the linearisations *abcd* and *bacd*, and have lower flex than the original plan (which has c unordered w.r.t. both a and b). However, we did not observe this happening in any of the plans analysed.

Results are summarised by domain in Table 1. For each domain, we report the number of plans analysed, the number of plans for which block deordering led to an increase in flex, and the average (over all analysed plans) flex values after step-based deordering and after block deordering. We imposed a 600 second time limit on the block deordering procedure; where the limit was reached ($\sim$ 8% of all plans), we take the flex of the plan after the last completed iteration.

Note that several domains are purely sequential, and do not permit any deordering of individual steps. (These have an average flex of zero after step de-

[2] The term "flexibility" is used with different meanings by different authors. Nguyen & Kambhampati's [16] definition is equivalent to ours. Muise et al. [13] use it for the number of linearisations of a partially ordered plan.

Domain	#plans	#inc.	Average flex after deordering	
			step	block
Airport (IPC4)	563	262	0.1492	0.1562
Blocks (IPC2)	381	284	0.0	0.0479
Cybersec (IPC6)	109	109	0.0142	0.2476
Depots (IPC3)	234	183	0.1589	0.1951
Elevators (IPC6)	276	216	0.1543	0.1965
FreeCell (IPC2)	358	214	0.0516	0.0596
Logistics (IPC2)	534	437	0.2435	0.2629
Openstacks (ADL, IPC5)	109	109	0.0	0.0733
Openstacks (STRIPS, IPC5)	15	15	0.0	0.0353
Openstacks (ADL, IPC6)	312	312	0.0453	0.1298
Openstacks (STRIPS, IPC6)	488	487	0.0469	0.1347
ParcPrinter (IPC6)	252	218	0.0820	0.2747
Pathways (STRIPS)	113	85	0.2455	0.2683
PegSol (IPC6)	301	261	0.0	0.1018
Rovers (IPC3)	187	187	0.2003	0.3541
Scanalyzer (IPC6)	343	202	0.1201	0.2418
Sokoban (IPC6)	205	174	0.0	0.0144
Storage (IPC5)	185	117	0.0423	0.1119
Transport (IPC6)	243	156	0.2071	0.2340
Woodworking (IPC6)	277	5	0.4551	0.4551

Table 1. Comparison of step and block deordering. For each domain, the first column shows the number of plans analysed, and the second the number of plans in which block deordering increased the flex value above that achieved by step deordering. The average flex values after step and block deordering are over all plans in each domain.

ordering.) Yet, even in these domains, it is often possible to block deorder plans. As an example, Figure 2 visualises the execution of part of a plan from the Sokoban domain (the reference plan for problem #11 from the IPC6 satisficing track). Sokoban is a puzzle game, involving a man who must push boxes around on a grid, one at a time, to reach a goal configuration. The domain is sequential because actions in the planning encoding of the game move the man from one square to another; thus, every step has a causal link from the step immediately before, and no deordering of individual steps is possible. Block deordering, however, is: In the example plan, the blocks consisting of steps 3–4 and steps 5–28 are independent and can be unordered. As can be seen clearly from the visualisation, moving steps 3–4, as a block, to after step 28 does not invalidate the plan. There are also two blocks, consisting of steps 10–21 and 22–23, within the larger block 5–28, that can be deordered. Our algorithm found all these possibilities.

5 Conclusions

Deordering makes the structure of a plan explicit, showing us which parts are necessarily sequential (because of dependency or interference) and which are in-

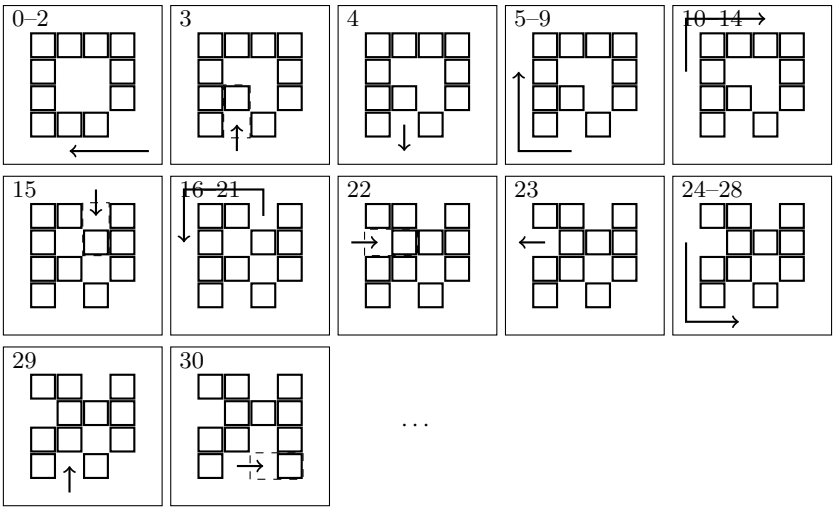

Fig. 2. Visualisation of the execution of (part of) an example plan in the Sokoban domain. Arrows show the movements of the man; dashed outlines show the movement of boxes that he pushes. The blocks consisting of steps 3–4 and 5–28 can be safely unordered, as can the blocks 10–21 and 22–23.

dependent and non-interfering. Block deordering improves on this by creating an on-the-fly hierarchical decomposition of the plan, encapsulating some dependencies and interferences within each block. Considering blocks, instead of primitive actions, as the units of partial ordering thus enables deordering plans more, including in cases where no deordering is possibly using the standard, step-wise, partial order plan notion. We showed that using a simple greedy algorithm to find block decompositions we could substantially increase the flex of deordered plans across many planning domains.

Maximising flex is not an end in itself; we use it only as a way to measure the "amount of deordering" done. The ultimate significance of block deordering will be determined by how much we can exploit the additional structural information it provides to improve on various post-plan generation tasks, such as explaining the plan to a user, or minimising execution coordination. We are particularly motivated by the use of plan structure information to improve the quality of plans by identifying subplans that can be locally improved [4]. As a simple example, in Figure 2 it can be observed that if the block 3–4 is relocated to after the block 5–28, steps 4 and 29 become redundant and can be removed. We are currently developing plan optimisation methods based on block deordering.

Acknowledgements This work was supported by the Australian Research Council discovery project DP0985532 "Exploiting Structure in AI Planning". NICTA is funded by the Australian Government as represented by the Depart-

ment of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Bäckström, C.: Computational aspects of reordering plans. Journal of AI Research **9** (1998) 99–137
2. Policella, N., Smith, S., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: Proc. 14th International Conference on Automated Planning & Scheduling (ICAPS'04). (2004) 209–218
3. Cox, J., Durfee, E., Bartold, T.: A distributed framework for solving the multiagent plan coordination problem. In: Proc. 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05). (2005) 821–827
4. Chrpa, L., McCluskey, T., Osborne, H.: Optimizing plans through analysis of action dependencies and independencies. In: Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12). (2012)
5. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Publishers (2004) ISBN: 1-55860-856-7.
6. Chapman, D.: Planning for conjunctive goals. Artificial Intelligence **32** (1987) 333–377
7. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proc. 9th National Conference on Artificial Intelligence. (1991)
8. Pednault, E.: Formulating multiagent, dynamic-world problems in the classical planning framework. In: Reasoning about Actions and Plans. (1986)
9. Regnier, P., Fade, B.: Complete determination of parallel actions and temporal optimization in linear plans of action. In: Proc. European Workshop on Planning. Volume 522 of Lecture Notes in AI., Springer (1991) 100–111
10. Veloso, M.M., Pérez, M.A., Carbonell, J.G.: Nonlinear planning with parallel resource allocation. In: Workshop on Innovative Approaches to Planning, Scheduling and Control, Morgan Kaufmann (1990) 207–212
11. Kambhampati, S., Kedar, S.: A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. Artificial Intelligence **67**(1) (1994) 29–70
12. Winner, E., Veloso, M.: Analyzing plans with conditional effects. In: Proc. 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02). (2002) 23–33
13. Muise, C., McIlratih, S., Beck, J.: Optimally relaxing partial-order plans with MaxSAT. In: Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12). (2012)
14. Chrpa, L., Barták, R.: Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In: Proc. 21st International Florida AI Research Society Conference (FLAIRS'08), AAAI Press (2008) 531–536 ISBN 978-1-57735-365-2.
15. Haslum, P., Jonsson, P.: Planning with reduced operator sets. In: Proc. 5:th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00). (2000) 150–158
16. Nguyen, X., Kambhampati, S.: Reviving partial order planning. In: Proc. 17th International Conference on Artificial Intelligence (IJCAI'01). (2001)