**The Australian National University**
2600 ACT | Canberra | Australia


Australian National University

**School of Computing**

College of Engineering and Computer Science

# Transforming Partially Ordered Planning Problems into Totally Ordered Problems

— 24 pt Honours project (S1/S2 2022)

A thesis submitted for the degree
*Bachelor of Advanced Computing*

**By**:
Ying Xian Wu

**Supervisors:**
Dr. Pascal Bercher
Mr. Songtuan Lin

May 2022

## Declaration of Authorship:

Except where otherwise indicated, this report is my own original work.

Sunday 15$^{\text{th}}$ May, 2022,

(Ying Xian Wu)

# Acknowledgements

This work would not have been possible without Dr Gregor Behnke, as his work forms the basis of this thesis. I am also grateful to Dr Pascal Bercher and Songtuan Lin for their supervision throughout this project.

# Abstract

The refinement step of partially ordered HTN planning has in the worst case k! possible solutions to search through when k plan steps should be ordered into a executable refinement. A large search space can be an advantage in case not enough is known about the domain to sensibly ensure a smaller search space will hold solutions, but also a disadvantage in the case that the search space is too large to efficiently search through. We attempt to linearize partially ordered HTN planning problems in order to reduce this disadvantage, while attempting to preserve at least one solution in the reduced search space. We test our techniques on the partially-ordered track of the bench-mark set of the IPC 2020. We use the Panda progression search planner on both linearised and partially-ordered problem. We find that in the majority of domains, the linearized problem can be solved faster. We also find that for the majority of problems, the linearized problem remains solvable.

# Table of Contents

Chapter 1

# Introduction

## 1.1 Introduction to HTN Planning

I have no idea what to write here.

## 1.2 Motivation

### 1.2.1 Potential advantages and disadvantages of total vs. partial order

- pros of partial order:
  - plan recognition: independent goals can be described in parallel (Daniel should be able to write something about that)
  - partial order is more expressive (both in terms of plan existence and in terms of computational complexity) meaning that more problems can be expressed
  - Domain model might be more intuitive: if a task is independent of some others it might be counter-intuitive to demand a certain position of it (if artificially made totally ordered)
- pros of total order:
  - computational complexity is lower (fewer worst-case solving time)
  - we can exploit specialised algorithms as well as heuristics. Note that heuristic design is comparably easy for total-order problems due to the missing interaction between tasks.

### 1.2.2 Further advantages from having a compilation of PO into TO plans

- We get another class of decidable partially ordered problems that's orthogonal to tail-recursive ones! (Because if the criterion 'matches', we know that the PO domain is equivalent to the resulting TO domain; the latter is decidable whereas the former is not.
- We can use more efficient algorithms and heuristics.

## 1.3 Contributions

Though the algorithm was implemented in the old PANDA-3 planner, and was used to produce many of the totally ordered domains in IPC benchmark, there does not exist empirical analysis of it's performance, or specification of it's formal properties. This paper provides both. It also investigates possible improvements to the original algorithm.

Chapter 2

# Background

## 2.1 Classical planning

A classical planning problem is defined as **Problem** $= (D, S_I, S_G)$
D is the domain of the problem, $S_I \in 2^F$ is the initial state and $S_G \in F$ is the goal. Every fact $\in F$ included in $S_G$ is true, and all other facts are false. This is the closed world assumption.

The domain $\mathbf{D} = (F, A)$. F is a finite set of facts, or propositional state variables. A is a finite set of actions. Every $a \in A$ is of type $2^F \times 2^F \times 2^F$, and of the format $< pre, add, del >$. The preconditions, add, and delete effects, of an action a are referred to as $pre(a)$, $add(a)$, and $del(a)$ respectively. An action a is executable in a state s if its precondition *pre* holds in s, i.e. pre $\subseteq$ s. If executable in s, its result is the successor state s' = (s $\setminus$ del ) $\cup$ add , i.e., variables in del(a) get removed and variables in ass(a) get added.

**Solution**: Solutions to a problem are action sequences executable in the initial state $S_I$ that lead to a state s" that satisfies all goals, i.e., s" $\supseteq$ g. Any such state s" is called a goal state.

## 2.2 Partially Ordered Hierarchical Planning

Also known as Partially Ordered Hierarchical Task Network Planning or PO HTN planning for short.

**Problem** $= (D, S_I, T_I)$ is over some domain D, has an initial state $S_I$, which is a total assignment to F, and has a initial compound task $T_I$. Every fact $\in F$ included in $S_I$ is true, and all other facts are false. This is the closed world assumption.

The domain $\mathbf{D} = (F, T_P, T_C, \delta, M)$.

1. F is the finite set of state variables, $T_P$ is the finite set of primitive task names, $T_C$ is the finite set of compound task names, and
2. $\delta$ is a mapping from primitive task name to action.
3. M is the finite set of decomposition methods. Each one maps a compound task name to a task network.
4. $T_P$ is the set of all possible primitive task names
5. $T_C$ is the set of all possible compound task names
6. $\delta$ maps actions to it's (preconditions, add, deletes) $\in 2^F x \times 2^F \times 2^F$. This can alternatively be referred to as $prec(t), add(t), del(t)$ for $a \in F, t \in T_P$
7. M is a set of methods. If $m \in M$, $m$ maps a compound task to a Task Network.

The **Task Network** $= (T, \prec, \alpha)$. T is a finite set of task identifiers (ids), $\prec$ is a partial order over T, and $\alpha$ maps task ids $\in$ T to task names in $T_C$ and $T_P$.

**Task Decomposition** A method m = (c, $tn_m$) decomposes a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ into a new task network $tn_2$ by replacing $t$ (i.e. $tn_1 \rightarrow_{t,m} tn_2$), if and only if $t \in T_1$, $\alpha_1(t) = c$, and there exists a task network $tn' = (T', \prec' \alpha')$ with $tn' \cong tn_m$ and $T' \cap T = \emptyset$ and

$$tn_2 := ((T_1 \setminus \{t\}) \cup T', \prec_1 \cup \prec' \prec_X, \alpha_1 \cup \alpha') \tag{2.1}$$

$$\prec_X := \{(t_1, t_2) \in T_1 \times T' | (t_1, t) \in \prec_1\} \cup \tag{2.2}$$

$$\{(t_1, t_2) \in T' \times T_1 | (t, t_2) \in \prec_1\} \tag{2.3}$$

$$\tag{2.4}$$

**HTN Solution** A solution to a problem is a task network $tn = (T, \prec, \alpha)$ if and only if

1. tn can be reached via decomposing $tn_I$
2. all tasks are primitive, $(\forall t \in T : \alpha(t) \in T_P)$
3. there exists a sequence $\langle t_1, t_2 ... t_n \rangle$ of the task ids in T that agrees with $\prec$ such that the application of that sequence $\langle \alpha(t_1), \alpha(t_2) ... \alpha(t_n) \rangle$ in $S_I$ is executable.

## 2.3 Totally Ordered Hierarchical Planning

Totally ordered hierarchical planning is the same as partially ordered planning in all respects except the task network. Both define a **Task Network** = $(T, \prec, \alpha)$. The difference is that $\prec$ now specifies the order between task ids such that for every a, b, in T, there exists an ordering in $\prec$ that $(a, b)$ or $(b, a)$. In other words, the tasks are totally ordered.

**Undecidable** A decision problem is equivalent to a function that accepts (potentially infinite) inputs and returns a yes or no. The decision problem is undecidable if it can be proved that there exists no algorithm for the problem that always leads to a correct yes-or-no answer.

Partially ordered HTN planning is undecidable, while totally ordered HTN planning is decidable.

## 2.4 Further Definitions (may be required)

**Task Decomposition Tree** Given a planning problem P, then a decomposition tree g = (T, E, $\prec$, $\alpha$, $\beta$) is a 5-tuple with the following properties. (T,E) is a tree with nodes T and directed edges E pointing towards the leafs. There is a strict partial order defined over the nodes, given by $\prec$. The nodes are labeled with task names by $\alpha$ : T $\rightarrow$ C $\cup$ O. Additionally, $\beta$ : T $\rightarrow$ M labels inner inner nodes with methods. $T(g)$ refers to the tasks of $g$, and $ch(g, t)$ refers to the direct children of $t \in T(g)$ in $g$.

**Lifted** Each object that exists in the world has a type from a pre-determined set of types defined by the domain. Methods apply to the set of object(s) that belong to a certain type.

**Grounded** A transition is ground if the parameters list only involves specific objects. Problems are grounded when all methods are grounded.

So to ground a problem: Let $\omega$ be a set of typed objects. The groundings of a transition schema a over $\omega$ is denoted by $\sigma$(a, $\omega$ ) and corresponds to the set of all ground transitions obtained by substituting $\sigma$ with a list of compatible objects taken from $\omega$ , and then substituting each occurrence of the variables which were in $\sigma$ with the newly introduced objects.

**Partial Order Causal Link Planning** *POCL criterion.* A partial POCL plan P = (PS, $\prec$, CL) is called POCL plan (also POCL solution) to a planning problem if and only if every precondition is supported by a causal link and there are no causal threats.

*PO criterion.* We refer to a partial POCL plan P = (PS , $\prec$, CL) without causal links, i.e., CL = $\emptyset$, as a partial partially ordered (PO) plan. Due to the absence of causal links, the solution criteria are here defined directly by the desired property of every linearization being executable.

i.e. A partial PO plan P = (PS , $\prec$) is called PO plan (also PO solution) to a planning problem if and only if every linearization is executable in the initial state and results into a goal state. i.e. it has every necessary ordering.

Therefore a linearization exists if and only if a POCL solution exists.

**Chapter 3**

# Related Work

On the Efficient Inference of Preconditions and Effects of Compound Tasks in Partially Ordered HTN Planning Domains, Conny Olz (2022)

Chapter 4

# Algorithm, Formalisation, Benchmark

## 4.1 Algorithm 1

Domain $(F, T_P, T_C, \delta, M)$ and Problem = $(T_I, S_0, TN_G)$

**Algorithm 1:** Calculate all possible preconditions and effects for compound tasks

**Function** *GetPreEff(*$F, T_P, T_C, \delta, M, visited$*)*
  function **for** $c \in T_C$ **do**
    **if** $c \notin$ *visited* **then**
      **for** $t \in \{t | m \in M \wedge m(c) = tn\}$ **do**
        **for** $a \in F$ **do**
          **if** $t \in T_P$ **then**
            $pre^*[v] = pre^*[v] \wedge Pre(t)$ $add^*[v] = add^*[v] \wedge Add(t)$
            $del^*[v] = del^*[v] \wedge Del(t)$
          **else**
            visited.add(c) ;
            GetPreEff(F, $T_P$, $T_C$, $\delta$, M, visited);
          **end**
        **end**
      **end**
    **end**
  **end**
  **return** $pre^*, add^*, del^*$;
**end**

**Algorithm 2:** Calculation of linearized methods

**Data:** $(F, T_P, T_C, \delta, M)$, $pre^*$, $add^*$, $del^*$
**Result:** $(F, T_P, T_C, \delta, M)$
**for** $m \in M$ **do**
  $NS_m \leftarrow \emptyset$;
  **for** $a \in F$ **do**
    $NS_m = \{(t, t', 1) | (a \in add^*(t) \wedge a \in pre^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in add^*(t) \wedge a \in del^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in del^*(t) \wedge a \in pre^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t, t', 1) | (a \in del^*(t) \wedge a \in add^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
  **end**
  directed graph G =(tasks(m),E);
  E $\leftarrow \emptyset$ ;
  **for** *ordering* $\in \prec$ **do**
    E = E $\cup$ (ordering, 0)
  **end**
  **for** *(t, t', w)* $\in NS_m$ **do**
    E = E $\cup$ ((t,t'), w)
  **end**
  BackEdges $\leftarrow$ DFS(G) ;
  **for** $e \in$ *BackEdges* **do**
    **if** *weight(e) = 1* **then**
      E = E $\setminus$ e
    **end**
    **if** *weight(e) = 0* **then**
      A, B = e ;
      Path = Dijkstra(B, A) ;
      randEdge = RANDOM( {e | weight(e) = 0 $\wedge$ e $\in$ *Path* } ) ;
      E = E $\setminus$ randEdge
    **end**
  **end**
  $\prec' =$ TopologicalSort(G) ;
  $m' = (tasks(m), \prec', \alpha(t))$ ;
  $M' = m' \cup M'$ ;
**end**
Return new domain = $(F, T_P, T_C, \delta, M')$

1. GetPreEff (time)= NumMethods * log(TDG height) * NumStateBits
2. Linearize (time) = order based on precondtions + (create graph + DFS + Dijkstra)*(as needed)
3. Let the method size be V, and the number of edges be E
4. Linearize = NumMethods * (NumStateBits + Method Size + Method Size +

(orderings) + (Method Size)log(method Size))

Therefore this algorithm is O(?) time?

### 4.1.1 Solution preserving properties of Algorithm 1

**Proposition 0.** *Removes linearizations*
*Proof.* This algorithm linearises all the methods to be totally ordered. Since sub-tasks inherit the orderings of their parents, it's impossible to preserve a solution that requires the interleaving of sub-tasks if their respective parents that are already ordered with respect to each other. This proves that the algorithm will always remove some linearizations, assuming the original domain was not already totally ordered.

Consider the simple problem:

F = {a, b, c, g}
$N_p = \{T_A, T_B, T_C, \text{G}\}$
$N_c = \{\text{AB }\}$
$\delta = \{(T_A, A), (T_B, B), (T_C, C)\}$
M = $(AB, \{\{4, 5\}, \{\}, \{(4, A), (5, B)\}\})$
$TN_Init = \{\{0, 1, 2\}, \{(0, 2), (1, 2)\}, \{(0, AB), (1, C), (2, G)\}\}$

$S_I = < a >$ A = <pre a, del a, add c>
C = <pre c, del c, add b>
B = <pre b, del b, add g>
G = <pre g, ,>

The initial task network enforces that G is the last action. To make G executable it needs the variable g, which only B can add. To make B executable it needs the variable b, which only C can add. To make C executable it needs the variable c, which only A can add. A is executable in the initial state. Therefore, the only solution is A C B G. This is impossible to achieve by linearizing methods, since either AB before C or C before AB, both of which exclude the solution.

This proves that the algorithm can remove all solutions, so Algorithm 1 is not complete.

**Proposition 1.** *Soundness*
*Proof.* We do not modify the sub-tasks a method produces, just the ordering between them, so the set of plans from the totally ordered method is just a subset of the plans possible from the partially ordered one. Any solution to the linearized problem is then obviously a solution to the original problem.

**Proposition 2.** *If Algorithm 1 didn't have to cycle-break, at least one solution is preserved*
*Proof.* Assume that there exists a solution in the PO domain. Using the same decomposition in the linearised domain, we can produce a linearization $(a_0, a_1, ..., a_n)$ of those

actions in the PO solution. We then prove by induction over the sequence $(a_0, ..a_n)$ that it is executable.

If $(a_0, ...a_n)$ is not executable, that means there exists some action $a_k, 0 < k < n$ that is not executable in the corresponding state. However $a_k$ must be executable in some linearization for $\{a_0, ..., a_n\}$, as we assumed it was a PO solution. So there must exist an action $a_i$, $0 < i < n$, that will adds A. Actions $a_0$ and $a_k$ must have a shared parent p in TDG. Then p has subtasks $t_0$ and $t_k$ that are parents of $a_0$ and $a_k$ respectively.

The linearization of this method would have drawn an ordering $(t_i, t_k)$ due to the way the algorithm defines $prec^*, add^*$ etc. We are assuming that all methods linearized without conflict, so $(t_i, t_k)$ should not be required. This safely enforces $(a_k, a_0)$ ordering in the final TO plan, meaning $a_0$ is not the first action in the resulting total order imposed by the algorithm. Therefore if $a_k$'s precondition could be met by any action $a_i$, $a_i$ would be ordered in front of it.

If $a_i$ does not exist then $a_k$ can never be executed for any linearization of $\{a_0, ...a_n\}$, contradicting the assumption that this was a PO solution. Since each action in the solution is executable, the entire sequence is executable linearization of actions produced by decomposition of initial task, i.e. the solution.

**2.1** *Even if Algorithm 1 didn't have to cycle break,* $\exists t.add^*(t)! = add(t) \vee del^*(t)! = del(t)$
*Proof.* Suppose there exists an compound task t whose method 1 decomposes to an action a, with $add(a) = \{A\}$. Assume there is another method which decomposes to action b, with $add(b) = \{B\}$ Therefore $prec^*(t) = \{A, B\}$ but both A and B, will not be applied in every execution of t.

**2.2** *If Algorithm 1 didn't have to cycle break, there may be t such that* $prec^*(t)$
*Proof.* Assume some compound task t' decomposes into at least two sub-tasks, $\{t_1, t_2\}$. Suppose that $a \in F, a \in add(t_1)$ and $a \in F, a \in prec(t_2)$. By the algorithm rules

- Add $\{(t, t') | (a \in add^*(t) \wedge a \in prec^*(t') \wedge t, t' \in tasks(m))\}$ to $NS_m$
- Add $\{(t', t) | (a \in add^*(t) \wedge a \in del^*(t') \wedge t, t' \in tasks(m))\}$ to $NS_m$
- Add $\{(t', t) | (a \in del^*(t) \wedge a \in prec^*(t') \wedge t, t' \in tasks(m))\}$ to $NS_m$
- Add $\{(t, t') | (a \in del^*(t) \wedge a \in add^*(t') \wedge t, t' \in tasks(m))\}$ to $NS_m$

We know that $t_1$ is ordered before $t_2$. And by the previous proposition 2.1, we know that there does not exist a task $t_3$ such that $a \in del(t_3)$ Therefore, $t_2$ will always be executable regardless of what other tasks might do. However, $a \in prec^*(t')$ because $a \in prec^*(t_2)$. Therefore it's possible for preconditions to be erroneous, even if Algorithm 1 did not need to cycle break.

**Proposition 3.** *When it does have to cycle break, it may preserve solutions*
*Proof.* As per proposition 2.1 and 2.2, not all of the preconditions and/or effects are always needed. Suppose the initial task network consisted of 2 sub-tasks, $t_1, t_2$. If $t_2$ can decompose into an action $a_1$ such that $add(a_1) = A$, and an action $a_2$ such that

$del(a_2) = A$, and $t_1$ can decompose into an action $a_3$ with $prec(a_3) = A$ $t_2 = addA, delA$ and $t_1 = precA$. The rules $\{(addA, precA), (precA, delA))\}$ of Algorthm 1 require orderings $\{(t_2, t_2), (t_2, t_1), (t_1, t_2)\}$, i.e. Despite a cycle break being needed, this is obviously a solvable problem. Order $t_2$ before $t_1$ for this method. Then when solving decompose $t_2$ to $a_1$ and $t_1$ to $a_3$ - this creates a solution.

**Proposition 4.** *For some tasks it doesn't matter where they are executed*
*Proof.* Assume that some pair of tasks $(t_1), (t_2)$ has no ordering between them. Assume also that $t_1t_2$ is ok but $(t_2)(t_1)$ is not, i.e. it matters where they are executed.

The required execution sequence implies that $t_2$ deletes some variable A $t_1$ relies on. But from the algorithm, that would mean that an ordering $(t_1, t_2)$ was created, which contradicts the assumption that there is no ordering between them. Therefore $(t_2)(t_1)$ must also be ok.

Thus if some task t has no ordering from the algorithm, it can never matter where it is in relation to any other task. The algorithm will not order it specifically when $(\forall t' \in m, \forall a \in prec(t).a \notin add(t') \wedge a \notin del(t')) \wedge (\forall t' \in m, \forall a \in add(t).a \notin prec(t') \wedge a \notin del(t')) \wedge (\forall t' \in m, \forall a \in del(t).a \notin add(t') \wedge a \notin prec(t'))$ In other words, the variables that affect t, do not affect other tasks in that method.

## 4.2 Algorithm 2

**Algorithm 3:** Calculation of linearized methods

**Data:** $(F, T_P, T_C, \delta, M)$, $pre^*$, $add^*$, $del^*$
**Result:** $(F, T_P, T_C, \delta, M)$
**for** $m \in M$ **do**
  $NS_m \leftarrow \emptyset$;
  **for** $a \in F$ **do**
    $NS_m = \{(t, t', 1) | (a \in add^*(t) \wedge a \in pre^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in add^*(t) \wedge a \in del^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in del^*(t) \wedge a \in pre^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
    $NS_m = \{(t, t', 1) | (a \in del^*(t) \wedge a \in add^*(t') \wedge t, t' \in tasks(m))\} \cup NS_m$ ;
  **end**
  directed graph G =(tasks(m),E);
  E $\leftarrow \emptyset$ ;
  **for** *ordering* $\in \prec$ **do**
    E = E $\cup$ (ordering, 0)
  **end**
  **for** *(t, t', w)* $\in NS_m$ **do**
    E = E $\cup$ ((t,t'), w)
  **end**
  BackEdges $\leftarrow$ DFS(G) ;
  **if** $|BackEdges| > 0$ **then**
    $\prec' =$ TopologicalSort(G) ;
    $m' = (tasks(m), \prec', \alpha(t))$ ;
    $M' = m' \cup M'$ ;
  **else**
    $M' = m \cup M'$
  **end**
**end**
Return new domain = $(F, T_P, T_C, \delta, M')$

A possible option is to only linearize the methods which do not need cycle breaking.

### 4.2.1 Solution preserving properties of Algorithm 2

**Proposition 5.** *Algorithm 2 is sound*
*Proof.* Same as Proposition 1

**Proposition 6.** *Does Algorithm 2 preserve at least one solution? When?*
*Proof.* ?

## 4.3 Algorithm 3

The definitions in Conny's paper assumes that there are no negative preconditions. Strict State-independent positive and negative effects:

$$\mathit{eff}_*^+ := (\bigcap_{s\in E(c)} \bigcap_{s'\in R_s(c)} s') \backslash (\bigcap_{s\in E(c)} s)$$

$$\mathit{eff}_*^- := \bigcap_{s\in E(c)} (F \backslash \bigcup_{s'\in R_s(c)} s')$$

if $E(c) \neq \emptyset$ otherwise $eff_*^{+/-}(c) := undef$

Strict Possible state-independent effects:

$$poss - \mathit{eff}_*^+ := \bigcup_{s\in E(c)} (\bigcup_{s'\in R_s(c)} s' \backslash s)$$

$$poss - \mathit{eff}_*^- := \bigcup_{s\in E(c)} ((\bigcup_{s'\in R_s(c)} (F \backslash s')) \cap s)$$

Relaxation: Define a new domain such that A' = $\{\emptyset, add, del\}|\{\emptyset, add, del\} \in A$. Using this new domain, define the relaxed guaranteed and relaxed possible effects $\mathit{eff}_*^{\emptyset+}$,

$\mathit{eff}_*^{\emptyset-}$,

$poss - \mathit{eff}_*^{\emptyset+}$,

$poss - \mathit{eff}_*^{\emptyset-}$ as before.

These effects will *definitely* happen, even if some methods cannot be linearised, Unlike Algorithm 1, this set of preconditions and effects have no false candidates in them. Unlike Algorithm 1, this set of preconditions and effects may be missing some State-independent

positive and negative effects.

**Algorithm 4:** Calculation of linearized methods

**Data:** $(F, T_P, T_C, \delta, M)$,
, $\mathit{eff}_*^{\emptyset+}, \mathit{eff}_*^{\emptyset-}, \mathit{poss} - \mathit{eff}_*^{\emptyset+}, \mathit{poss} - \mathit{eff}_*^{\emptyset-}$
**Result:** $(F, T_P, T_C, \delta, M)$
**for** $m \in M$ **do**
  $NS_m \leftarrow \emptyset$;
  **for** $a \in F$ **do**
    $NS_m = \{(t, t', 1) | (a \in \mathit{add}^*(t) \wedge a \in \mathit{pre}^*(t') \wedge t, t' \in \mathit{tasks}(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in \mathit{add}^*(t) \wedge a \in \mathit{del}^*(t') \wedge t, t' \in \mathit{tasks}(m))\} \cup NS_m$ ;
    $NS_m = \{(t', t, 1) | (a \in \mathit{del}^*(t) \wedge a \in \mathit{pre}^*(t') \wedge t, t' \in \mathit{tasks}(m))\} \cup NS_m$ ;
    $NS_m = \{(t, t', 1) | (a \in \mathit{del}^*(t) \wedge a \in \mathit{add}^*(t') \wedge t, t' \in \mathit{tasks}(m))\} \cup NS_m$ ;
    Add edges between definite adds and definite deletes, weight =1 ;
    Add edges between definite effects and possible preconditions, weight=2 ;
    Add edges between possible effects and possible preconditions, with weight=3 ;
  **end**
  directed graph G =(tasks(m),E);
  E $\leftarrow \emptyset$ ;
  **for** *ordering* $\in \prec$ **do**
    E = E $\cup$ (ordering, 0)
  **end**
  **for** *(t, t', w)* $\in NS_m$ **do**
    E = E $\cup$ ((t,t'), w)
  **end**
  BackEdges $\leftarrow$ DFS(G) ;
  **for** $e \in \mathit{BackEdges}$ **do**
    **if** *weight(e) = 4* **then**
      E = E $\setminus$ e
    **else**
      weight(e) = 3
    **end**
    A, B = e ;
    Path = Dijkstra(B, A) ;
    randEdge = RANDOM( {e | weight(e) = 0 $\wedge$ e $\in \mathit{Path}$ } ) ;
    E = E $\setminus$ randEdge
  **end**
  $\prec' =$ TopologicalSort(G) ;
  $m' = (\mathit{tasks}(m), \prec', \alpha(t))$ ;
  $M' = m' \cup M'$ ;
**end**
Return new domain = $(F, T_P, T_C, \delta, M')$

### 4.3.1 Solution preserving properties of Algorithm 3

**Proposition 7.** *Algorithm 3 is sound*
*Proof.* Same as Proposition 1

**Proposition 8.** *Does Algorithm 3 preserve at least one solution? When?*
*Proof.* ?

Chapter 5

# Evaluation

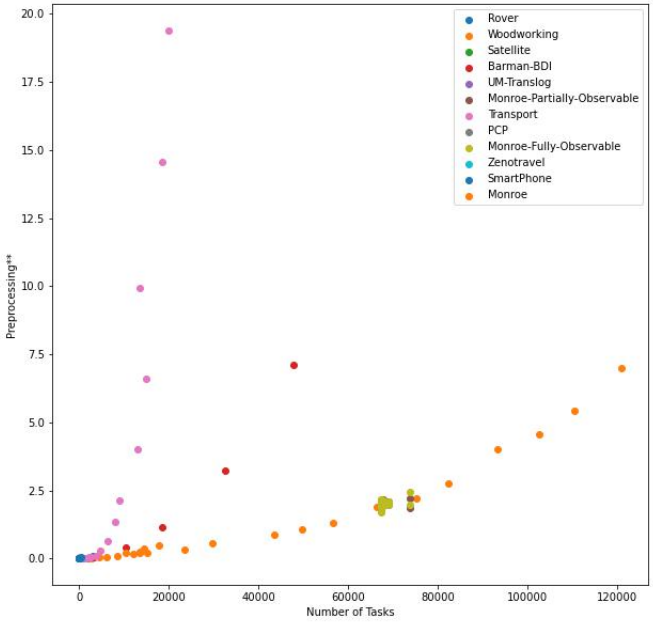
## 5.1 Empirical Evaluation

To prove that our technique is beneficial, we conduct a standard empirical evaluation on PO domains and compare the runtime of PO planners vs. transformation + TO planners

### 5.1.1 Hardware Setup

The following benchmark used ?GB RAM, Nectar cores, threads, processor, Nectar OS, and C++

Full Tables in Appendix. Plots/Tables of relevant properties, e.g.




1. None of the instances can be linearized.
2. Despite this, ?percent of linearized problems are solvable.
3. The additional processing time is within the same order of time as the time needed for grounding and a tiny fraction of the overall time $\sim$
4. We compare the ratio of (Grounding + Pre-processing time + TO solve time) to (Grounding + PO solve time), as those are the steps needed to solve the respective problems. The average solve time for the TO problem is ?? percent of the PO problem, for all problems.
5. Difference in reduction of total solving time influenced most by ? (e.g. domain, recursiveness of problem, percentage of linearizable methods, etc)

Chapter 6

# Concluding Remarks

If you wish, you may also name that section *"Conclusion and Future Work"*, though it might not be a perfect choice to have a section named "A & B" if it has subsections "A" and "B". Also note that you don't necessarily have to use these subsections; that also depends on how much content you have in each. (E.g., having a section header might be odd if it contains just three lines.)

## 6.1 Conclusion

This chapter usually summarizes the entire paper including the conclusions drawn, i.e., did the developed techniques work? Maybe add why or why not. Also note that every single scientific paper has such a section, so you can check out many examples, preferably at top-tier venues, e.g., by your supervisor(s).

## 6.2 Future Work

On top of that, you could discuss future work (and make clear why that is future work, i.e., by which observations did they get justified?).

Note that future work in scientific papers is often not mentioned at all or just in a very few sentences within the conclusion. That should not stop you from putting some effort in. This will (also) show the examiner(s)/supervisor(s) how well you understood the topic or how engaged you are.

Appendix A

# Appendix: All Results For Algorithms

## A.1 Algorithm 1

### A.1.1 Algorithm 1 pre-processing performance

* Percentage of number of linearised methods / number of methods with 2 or more subtasks
** All time in seconds
*** Average number of tasks per method, excluding the initial task network

| Rover | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 396 | 2.24823 | 424 | 35.2727 | 0.049 | 0.004175 |
| 2258 | 2.18431 | 2589 | 25.1229 | 0.1288 | 0.037174 |
| 1484 | 2.20161 | 1866 | 39.6364 | 0.1101 | 0.023642 |
| 460 | 2.32727 | 441 | 25.3776 | 0.0283 | 0.006185 |
| 65 | 2.34694 | 50 | 35.1351 | 0.0119 | 0.001148 |
| 59 | 2.35556 | 46 | 40.625 | 0.0112 | 0.001594 |
| 1202 | 2.22602 | 1377 | 33.114 | 0.0767 | 0.029776 |
| 382 | 2.37709 | 359 | 30.303 | 0.0235 | 0.006008 |
| 561 | 2.28114 | 595 | 32.2115 | 0.0363 | 0.009243 |
| 626 | 2.30711 | 662 | 33.6323 | 0.0422 | 0.008775 |
| 89 | 2.47143 | 71 | 44.0 | 0.0132 | 0.001044 |
| 595 | 2.19498 | 678 | 34.4262 | 0.0364 | 0.010964 |
| 494 | 2.30469 | 513 | 32.7684 | 0.0292 | 0.013627 |
| 245 | 2.22321 | 225 | 24.1379 | 0.0177 | 0.02508 |
| 152 | 2.26016 | 124 | 21.5686 | 0.0141 | 0.004343 |
| 414 | 2.3222 | 420 | 33.5593 | 0.0255 | 0.006752 |
| 707 | 2.26154 | 716 | 22.7941 | 0.0367 | 0.010028 |
| 243 | 2.42857 | 218 | 33.5443 | 0.0187 | 0.002684 |
| 81 | 2.46667 | 61 | 40.0 | 0.0124 | 0.00088 |

Recursive: False

| Woodworking | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 1005 | 1.16356 | 1450 | 0.0 | 0.045 | 0.004655 |
| 29804 | 1.02155 | 129218 | 0.0 | 1.4686 | 0.562532 |
| 2748 | 1.11924 | 5620 | 0.0 | 0.0975 | 0.016805 |
| 896 | 1.13949 | 1528 | 0.0 | 0.0444 | 0.00505 |
| 93209 | 1.01596 | 485051 | 0.0 | 5.5487 | 4.02608 |
| 12 | 1.0 | 9 | 0.0 | 0.013 | 5.5e-05 |
| 75238 | 1.01763 | 373234 | 0.0 | 4.0672 | 2.19396 |
| 56590 | 1.02022 | 258945 | 0.0 | 2.8876 | 1.29956 |
| 21 | 1.0 | 15 | 0.0 | 0.0179 | 0.000101 |
| 82285 | 1.01898 | 390298 | 0.0 | 4.5634 | 2.7429 |
| 16 | 1.54545 | 12 | 0.0 | 0.0137 | 8.8e-05 |
| 964 | 1.1419 | 1798 | 0.0 | 0.0547 | 0.006413 |
| 23568 | 1.03312 | 84962 | 0.0 | 1.1152 | 0.331937 |
| 1629 | 1.13057 | 2988 | 0.0 | 0.0644 | 0.00987 |
| 15266 | 1.02848 | 57717 | 0.0 | 0.7028 | 0.210608 |
| 32 | 1.12121 | 34 | 0.0 | 0.0143 | 0.000138 |
| 4386 | 1.11759 | 9594 | 0.0 | 0.152 | 0.035009 |
| 6167 | 1.06607 | 14167 | 0.0 | 0.2109 | 0.04804 |
| 66431 | 1.01987 | 306499 | 0.0 | 3.4698 | 1.87959 |
| 146 | 1.11111 | 199 | 0.0 | 0.019 | 0.000637 |
| 721 | 1.24538 | 1191 | 0.0 | 0.0378 | 0.004715 |
| 102683 | 1.01772 | 512099 | 0.0 | 5.9302 | 4.55552 |
| 36 | 1.10811 | 38 | 0.0 | 0.0149 | 0.000148 |
| 110366 | 1.01926 | 521261 | 0.0 | 7.7864 | 5.41724 |
| 1267 | 1.22291 | 2087 | 0.0 | 0.057 | 0.008655 |
| 120819 | 1.01804 | 592235 | 0.0 | 6.9148 | 6.97687 |
| 43515 | 1.02161 | 196969 | 0.0 | 2.4068 | 0.891157 |
| 254 | 1.11599 | 320 | 0.0 | 0.0233 | 0.001087 |
| 49650 | 1.02036 | 230394 | 0.0 | 2.477 | 1.07154 |

Recursive: False

| Satellite | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 29 | 2.16667 | 37 | 53.125 | 0.01 | 0.000293 |
| 128 | 2.27027 | 260 | 37.5 | 0.0081 | 0.002181 |
| 10 | 2.33333 | 10 | 57.1429 | 0.0045 | 9.8e-05 |
| 17 | 2.55 | 21 | 50.0 | 0.0044 | 0.000223 |
| 32 | 2.18919 | 38 | 56.25 | 0.0053 | 0.000301 |
| 88 | 2.22388 | 135 | 46.9565 | 0.0068 | 0.001072 |
| 22 | 2.60714 | 29 | 48.0 | 0.0048 | 0.000327 |
| 58 | 2.1875 | 81 | 54.9296 | 0.0061 | 0.000598 |
| 55 | 2.24176 | 92 | 41.5584 | 0.0056 | 0.000872 |
| 66 | 2.64286 | 99 | 50.5747 | 0.0064 | 0.001052 |
| 76 | 2.7395 | 120 | 49.5413 | 0.0068 | 0.001529 |
| 55 | 2.17808 | 74 | 54.6875 | 0.006 | 0.000559 |
| 139 | 2.39801 | 202 | 53.2967 | 0.0091 | 0.001877 |
| 98 | 2.27273 | 155 | 42.4 | 0.0087 | 0.001523 |
| 40 | 2.55172 | 59 | 50.0 | 0.0052 | 0.000595 |
| 44 | 2.63492 | 64 | 50.0 | 0.0058 | 0.000685 |
| 24 | 2.14286 | 29 | 56.0 | 0.0047 | 0.000207 |
| 17 | 2.55 | 21 | 50.0 | 0.0048 | 0.000222 |
| 36 | 2.0 | 45 | 50.0 | 0.005 | 0.000288 |
| 63 | 2.19512 | 83 | 54.9296 | 0.0064 | 0.000621 |
| 66 | 2.20225 | 90 | 55.1282 | 0.0066 | 0.000683 |
| 34 | 2.2 | 46 | 52.5 | 0.0051 | 0.000346 |
| 131 | 2.75943 | 213 | 50.5102 | 0.0094 | 0.002685 |
| 31 | 2.55 | 41 | 51.4286 | 0.0059 | 0.000431 |

Recursive: False

| Barman-BDI | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 1419 | 2.46201 | 1146 | 27.7451 | 0.0561 | 0.019318 |
| 32612 | 2.52212 | 26972 | 22.9102 | 1.2605 | 3.24646 |
| 18408 | 2.51788 | 15184 | 23.4252 | 0.6573 | 1.16557 |
| 1125 | 2.45215 | 910 | 27.599 | 0.0446 | 0.017246 |
| 165 | 2.32283 | 128 | 36.6337 | 0.0182 | 0.001367 |
| 300 | 2.38723 | 236 | 34.3434 | 0.021 | 0.002747 |
| 10434 | 2.51241 | 8586 | 23.929 | 0.3766 | 0.422801 |
| 649 | 2.42747 | 518 | 30.4444 | 0.0346 | 0.006654 |
| 2597 | 2.4759 | 2117 | 25.7727 | 0.0814 | 0.042462 |
| 3060 | 2.49415 | 2480 | 26.899 | 0.0924 | 0.057508 |
| 393 | 2.40909 | 309 | 33.9695 | 0.0239 | 0.003794 |
| 3250 | 2.49112 | 2648 | 25.8966 | 0.0994 | 0.059478 |
| 1484 | 2.475 | 1201 | 27.5349 | 0.0505 | 0.019722 |
| 580 | 2.43573 | 460 | 31.9095 | 0.0298 | 0.006146 |
| 559 | 2.42568 | 445 | 31.1688 | 0.0287 | 0.005748 |
| 782 | 2.4464 | 626 | 29.8725 | 0.0364 | 0.008496 |
| 1871 | 2.48379 | 1512 | 27.6549 | 0.0641 | 0.027674 |
| 628 | 2.41833 | 503 | 29.7483 | 0.0333 | 0.006294 |
| 444 | 2.39601 | 352 | 32.0 | 0.0256 | 0.004284 |

Recursive: False

| UM-Translog | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 8 | 0.0 | 1 | 0.0 | 0.0188 | 0.000111 |
| 12 | 0.0 | 1 | 0.0 | 0.016 | 0.000217 |
| 11 | 7.5 | 3 | 0.0 | 0.016 | 0.000185 |
| 12 | 0.0 | 1 | 0.0 | 0.0161 | 0.00024 |
| 14 | 0.0 | 1 | 0.0 | 0.0162 | 0.000583 |
| 12 | 3.5 | 3 | 0.0 | 0.0221 | 0.00026 |
| 20 | 14.5 | 3 | 0.0 | 0.0221 | 0.001484 |
| 10 | 7.5 | 3 | 0.0 | 0.0161 | 0.000186 |
| 8 | 0.0 | 1 | 0.0 | 0.0165 | 0.000198 |
| 22 | 0.0 | 1 | 0.0 | 0.0183 | 0.001156 |
| 8 | 0.0 | 1 | 0.0 | 0.0174 | 0.000162 |
| 8 | 0.0 | 1 | 0.0 | 0.017 | 0.000167 |
| 14 | 0.0 | 1 | 0.0 | 0.0167 | 0.000537 |
| 10 | 0.0 | 1 | 0.0 | 0.0168 | 0.000198 |
| 14 | 7.5 | 3 | 0.0 | 0.0168 | 0.000551 |
| 24 | 17.5 | 3 | 0.0 | 0.0191 | 0.001745 |
| 10 | 0.0 | 1 | 0.0 | 0.0212 | 0.00023 |
| 10 | 0.0 | 1 | 0.0 | 0.0173 | 0.000195 |
| 8 | 0.0 | 1 | 0.0 | 0.0168 | 0.00017 |
| 21 | 0.0 | 1 | 0.0 | 0.0174 | 0.000967 |
| 8 | 0.0 | 1 | 0.0 | 0.0171 | 0.0002 |

Recursive: False

| Monroe-Partially-Observable | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 67418 | 2.21429 | 55016 | 68.1127 | 2.0101 | 1.9434 |
| 68890 | 2.21431 | 56051 | 68.0574 | 2.225 | 2.07487 |
| 67422 | 2.2142 | 55020 | 68.1127 | 2.039 | 2.01091 |
| 67433 | 2.2142 | 55029 | 67.8145 | 2.1308 | 2.00445 |
| 73827 | 2.19955 | 60547 | 68.0642 | 2.1147 | 1.8394 |
| 67430 | 2.21402 | 55028 | 68.1127 | 2.1293 | 2.03887 |
| 67429 | 2.21408 | 55027 | 68.1134 | 2.7032 | 2.05474 |
| 67812 | 2.21714 | 55315 | 67.682 | 2.1563 | 2.18095 |
| 67811 | 2.21707 | 55314 | 67.9555 | 2.107 | 2.13088 |
| 67428 | 2.21407 | 55026 | 68.1127 | 2.0435 | 1.89057 |
| 68884 | 2.21444 | 56045 | 68.0574 | 2.1099 | 1.95732 |
| 67425 | 2.21417 | 55023 | 68.1134 | 2.0865 | 1.90319 |
| 67418 | 2.21429 | 55016 | 68.1127 | 2.0806 | 1.98215 |
| 67801 | 2.21729 | 55304 | 67.9758 | 2.0694 | 2.11927 |
| 73821 | 2.19967 | 60541 | 68.0642 | 2.2321 | 2.20587 |
| 67425 | 2.21438 | 55021 | 68.1086 | 2.0549 | 1.84788 |
| 67428 | 2.21407 | 55026 | 68.1127 | 2.0925 | 1.85908 |
| 67422 | 2.2142 | 55020 | 68.1127 | 2.016 | 1.90154 |
| 67433 | 2.21407 | 55031 | 67.8186 | 2.0786 | 1.87328 |
| 67425 | 2.21438 | 55021 | 68.1086 | 2.0699 | 1.97273 |
| 67801 | 2.21729 | 55304 | 67.9758 | 2.1719 | 2.01427 |
| 67429 | 2.21415 | 55027 | 68.1127 | 2.1738 | 2.08523 |
| 67443 | 2.21377 | 55041 | 68.1134 | 2.3989 | 1.95702 |

Recursive: False

| Transport | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 8062 | 1.38812 | 9073 | 0.0 | 0.2104 | 1.34733 |
| 6386 | 1.37436 | 6826 | 0.0 | 0.1462 | 0.637496 |
| 3673 | 1.38752 | 4135 | 0.0 | 0.0918 | 0.100144 |
| 12969 | 1.40114 | 15377 | 0.0 | 0.3408 | 3.99726 |
| 13421 | 1.36337 | 13761 | 0.0 | 0.3683 | 9.91777 |
| 511 | 1.43529 | 511 | 0.0 | 0.016 | 0.004958 |
| 129 | 1.43077 | 131 | 0.0 | 0.0077 | 0.000732 |
| 402 | 1.36 | 401 | 0.0 | 0.0142 | 0.003512 |
| 381 | 1.375 | 401 | 0.0 | 0.0143 | 0.003101 |
| 116 | 1.20792 | 102 | 0.0 | 0.0093 | 0.032754 |
| 19 | 1.53333 | 16 | 0.0 | 0.0055 | 0.000315 |
| 45 | 1.3 | 41 | 0.0 | 0.0058 | 0.000963 |
| 4763 | 1.38814 | 5365 | 0.0 | 0.1162 | 0.285897 |
| 340 | 1.37288 | 355 | 0.0 | 0.0125 | 0.002433 |
| 14939 | 1.40585 | 18057 | 0.0 | 0.4223 | 6.58352 |
| 104 | 1.2551 | 99 | 0.0 | 0.0072 | 0.010046 |
| 2286 | 1.36952 | 2396 | 0.0 | 0.052 | 0.050062 |
| 175 | 1.42857 | 169 | 0.0 | 0.0084 | 0.001073 |
| 18581 | 1.36651 | 19291 | 0.0 | 0.5784 | 14.5498 |
| 397 | 1.43704 | 406 | 0.0 | 0.0128 | 0.00289 |
| 235 | 1.42982 | 229 | 0.0 | 0.0115 | 0.001958 |
| 1473 | 1.37047 | 1545 | 0.0 | 0.0348 | 0.023913 |
| 44 | 1.31707 | 42 | 0.0 | 0.0052 | 0.000912 |
| 1416 | 1.4306 | 1406 | 0.0 | 0.0323 | 0.027404 |
| 605 | 1.36967 | 634 | 0.0 | 0.0176 | 0.004384 |
| 1161 | 1.43952 | 1241 | 0.0 | 0.0273 | 0.011946 |
| 2649 | 1.3913 | 3037 | 0.0 | 0.0639 | 0.037584 |
| 204 | 1.42857 | 197 | 0.0 | 0.0085 | 0.001312 |
| 20001 | 1.36381 | 20561 | 0.0 | 0.5632 | 19.3826 |
| 363 | 1.38824 | 341 | 0.0 | 0.0132 | 0.008704 |
| 9038 | 1.38179 | 9920 | 0.0 | 0.2186 | 2.12052 |
| 115 | 1.42593 | 109 | 0.0 | 0.0068 | 0.000621 |
| 71 | 1.30769 | 66 | 0.0 | 0.0055 | 0.005033 |
| 81 | 1.23288 | 74 | 0.0 | 0.0067 | 0.005131 |
| 103 | 1.21111 | 91 | 0.0 | 0.0074 | 0.018213 |
| 144 | 1.42647 | 137 | 0.0 | 0.0074 | 0.000798 |
| 79 | 1.28986 | 70 | 0.0 | 0.0066 | 0.010016 |
| 68 | 1.24194 | 63 | 0.0 | 0.0068 | 0.002338 |
| 309 | 1.39205 | 353 | 0.0 | 0.0118 | 0.001922 |

Recursive: False

| PCP | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 13 | 3.66667 | 13 | 0.0 | 0.0051 | 0.000401 |
| 15 | 3.875 | 17 | 0.0 | 0.0051 | 0.00056 |
| 13 | 3.66667 | 13 | 0.0 | 0.0049 | 0.000446 |
| 15 | 3.16667 | 13 | 0.0 | 0.0052 | 0.00028 |
| 15 | 4.5 | 17 | 0.0 | 0.0057 | 0.000935 |
| 13 | 3.83333 | 13 | 0.0 | 0.0048 | 0.000576 |
| 13 | 3.83333 | 13 | 0.0 | 0.005 | 0.000466 |
| 13 | 3.66667 | 13 | 0.0 | 0.0049 | 0.000404 |
| 17 | 3.125 | 17 | 0.0 | 0.0052 | 0.000377 |
| 11 | 3.0 | 9 | 0.0 | 0.0044 | 0.000177 |
| 11 | 3.5 | 9 | 0.0 | 0.0044 | 0.000277 |
| 11 | 4.0 | 9 | 0.0 | 0.0046 | 0.000415 |
| 11 | 4.5 | 9 | 0.0 | 0.0101 | 0.000928 |
| 11 | 5.0 | 9 | 0.0 | 0.0045 | 0.00096 |
| 11 | 3.5 | 9 | 0.0 | 0.0045 | 0.000274 |
| 13 | 3.16667 | 13 | 0.0 | 0.0052 | 0.000282 |

Recursive: False

| Monroe-Fully-Observable | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 67418 | 2.21429 | 55016 | 68.1127 | 1.9754 | 1.88009 |
| 68890 | 2.21431 | 56051 | 68.0574 | 2.1289 | 1.98078 |
| 67422 | 2.2142 | 55020 | 68.1127 | 2.0137 | 2.16721 |
| 67435 | 2.21416 | 55031 | 67.8145 | 2.144 | 2.076 |
| 73827 | 2.19955 | 60547 | 68.0642 | 2.1169 | 1.96254 |
| 67436 | 2.21389 | 55034 | 67.8186 | 2.0457 | 1.70454 |
| 67433 | 2.21399 | 55031 | 68.1134 | 2.145 | 1.75014 |
| 67818 | 2.21701 | 55321 | 67.682 | 2.3581 | 2.08889 |
| 67811 | 2.21707 | 55314 | 67.9555 | 2.143 | 2.06021 |
| 67428 | 2.21407 | 55026 | 68.1127 | 2.1133 | 2.05435 |
| 68884 | 2.21444 | 56045 | 68.0574 | 2.1945 | 2.07282 |
| 67425 | 2.21417 | 55023 | 68.1134 | 2.1373 | 1.98499 |
| 67418 | 2.21429 | 55016 | 68.1127 | 2.5008 | 2.08575 |
| 67801 | 2.21729 | 55304 | 67.9758 | 2.4429 | 2.13274 |
| 73821 | 2.19967 | 60541 | 68.0642 | 3.5701 | 2.43004 |
| 67425 | 2.21438 | 55021 | 68.1086 | 2.252 | 2.13456 |
| 67430 | 2.21402 | 55028 | 67.8186 | 2.1134 | 1.9436 |
| 67422 | 2.2142 | 55020 | 68.1127 | 2.0775 | 1.89634 |
| 67439 | 2.21393 | 55037 | 67.8186 | 2.1591 | 2.0417 |
| 67425 | 2.21438 | 55021 | 68.1086 | 2.1023 | 1.99735 |
| 67801 | 2.21729 | 55304 | 67.9758 | 2.0668 | 1.9264 |
| 67429 | 2.21415 | 55027 | 68.1127 | 2.1173 | 1.91552 |
| 67451 | 2.2136 | 55049 | 68.1134 | 2.297 | 2.05399 |
| 67427 | 2.2142 | 55025 | 68.1127 | 2.2347 | 2.17898 |

Recursive: False

| Zenotravel | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 485 | 2.01786 | 449 | 48.8998 | 0.0262 | 0.002971 |
| 123 | 2.02913 | 104 | 44.2105 | 0.011 | 0.000698 |
| 97 | 2.05263 | 77 | 46.3768 | 0.0138 | 0.0006 |
| 281 | 2.09244 | 239 | 46.9484 | 0.017 | 0.00185 |

Recursive: False

| SmartPhone | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 10 | 1.5 | 3 | 0.0 | 0.0279 | 0.000249 |
| 74 | 1.18421 | 77 | 40.0 | 0.0279 | 0.003691 |
| 26 | 2.27273 | 12 | 0.0 | 0.0247 | 0.000326 |
| 281 | 1.05831 | 344 | 62.5 | 0.034 | 0.03811 |
| 22 | 1.125 | 25 | 50.0 | 0.0243 | 0.000192 |
| 23 | 1.375 | 9 | 0.0 | 0.025 | 0.001222 |

Recursive: False

| Monroe | | | | | |
|---|---|---|---|---|---|
| Tasks | avg*** | Methods | Linearizable Methods % * | Grounding** | Preprocessing** |
| 10303 | 1.80762 | 13973 | 75.0408 | 0.9821 | 0.197555 |
| 13609 | 1.87757 | 14132 | 65.5856 | 0.8972 | 0.218261 |
| 13609 | 1.87757 | 14132 | 65.5856 | 0.9332 | 0.26452 |
| 17739 | 2.08939 | 27207 | 68.4257 | 1.2638 | 0.488081 |
| 14373 | 1.97155 | 23727 | 78.7107 | 1.0089 | 0.357765 |
| 12092 | 1.84136 | 12684 | 68.3509 | 0.9201 | 0.184313 |

Recursive: False

### A.1.2 Solving Performance

We use the partially ordered track in the IPC 2020 benchmark.

| Rover | | |
|---|---|---|
| Tasks | time to solve PO | time to solve TO |
| 396 | 203.617 | 0.168 |
| 460 | 187.619 | 4.077 |
| 65 | 0.236 | 0.057 |
| 59 | 0.111 | 0.055 |
| 1202 | 380.662 | 5.17 |
| 382 | 155.551 | 0.6 |
| 561 | 148.941 | 0.191 |
| 626 | 204.075 | 4.275 |
| 89 | 0.134 | 0.058 |
| 595 | 202.559 | 0.403 |
| 494 | 179.874 | 24.747 |
| 245 | 125.468 | 0.257 |
| 152 | 134.515 | 0.152 |
| 414 | 201.005 | 0.204 |
| 707 | 261.94 | 47.815 |

## A.2 Algorithm 2

## A.3 Algorithm 3

Appendix B

# Appendix: Explanation on Page Borders

What you find here is an explanation of why the border width keeps flipping from left to right – which you might have spotted and wondered why that's the case.

Firstly, that is *intended* and thus correct, so there is no reason to worry about this. The reason is that this document is configured as a two-sided book, which means:

- We assume the document will be printed out,
- that this will be done in a two-sided mode (i.e., the document will be printed on both sides of each page), and
- that the bookbinding will be in the middle, just like in every book.

When you open the book, there are three borders of equal size $n$. This however requires that even pages have a border of $n$ on their left and $\frac{n}{2}$ on their right, and odd pages have a border of $\frac{n}{2}$ on their left and $n$ on their right. This is illustrated in Figure B.1.

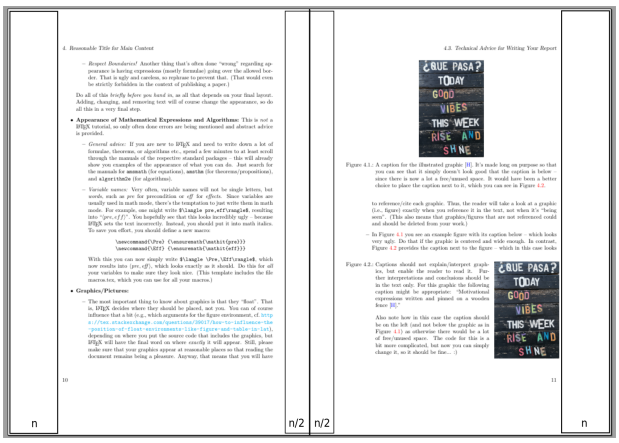



Figure B.1: Illustration showing why page borders flip.

# Bibliography