A Survey on Plan Optimization

Pascal Bercher[1] and Patrik Haslum[1] and Christian Muise[2]

[1]The Australian National University
[2]Queen's University

August 9, 2024

Introduction

Motivation

Plan optimization in a nutshell:

- Input: A plan and the underlying model, but no search space
- Output: “A better version” of said plan (details: see later)

Motivation

Plan optimization in a nutshell:

- Input: A plan and the underlying model, but no search space
- Output: “A better version” of said plan (details: see later)

Why doing plan optimization?

- Optimality or at least non-redundancy is often important, e.g.,
  - clearly, we want to save costs
  - reduce execution time (exploiting parallelism)
  - be more flexible during execution
  - in plan explanation, what if somebody asks why action X is required, but it’s redundant in the plan?

Motivation

Plan optimization in a nutshell:

- Input: A plan and the underlying model, but no search space
- Output: “A better version” of said plan (details: see later)

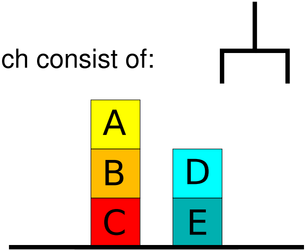
Why doing plan optimization?

- Optimality or at least non-redundancy is often important, e.g.,
  - clearly, we want to save costs
  - reduce execution time (exploiting parallelism)
  - be more flexible during execution
  - in plan explanation, what if somebody asks why action X is required, but it’s redundant in the plan?
- But finding an optimal plan is *much* harder than finding *any*
- We might also not be in control of the plans we are given

Considered Types of Planning Problems

Considered Types of Planning Problems

Considered Types of Planning Problems: Classical (=non-hierarchical) Planning

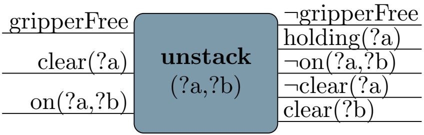


We consider *classical planning problems*, which consist of:

- All existing "facts" F.
- An initial state $s_I \in 2^F$.
- A set of available actions A.
- A goal description $g \subseteq F$.

→ Find an action sequence (i.e., a *plan*) that transforms s_I into g.

For example, one of the available actions is:

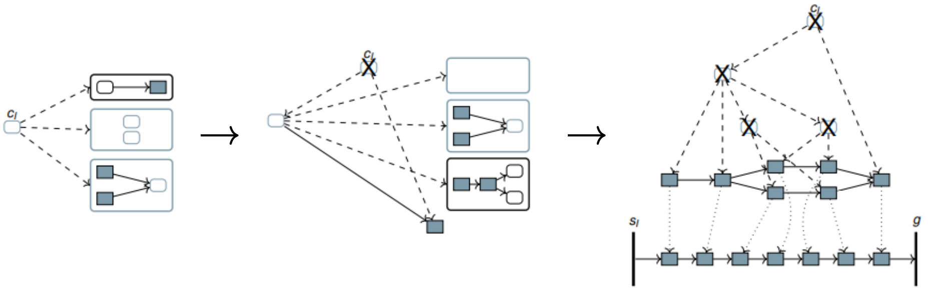


- For an action to be executable, all preconditions must hold.
- Actions change states by adding or deleting their effects.

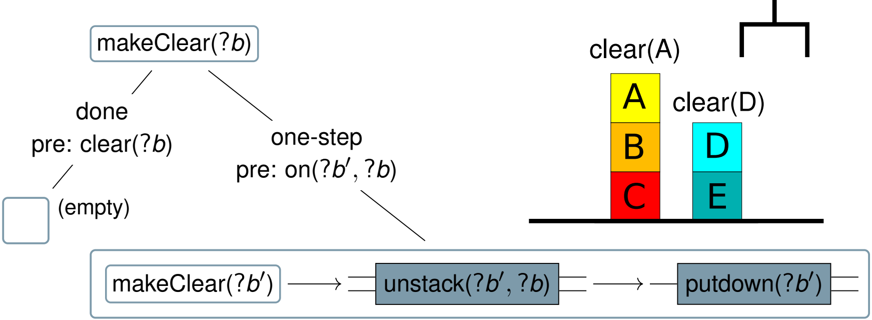
Considered Types of Planning Problems: Hierarchical Task Network (HTN) Planning

In HTN Planning,

- we do not plan for state-based facts, instead,
- we have initial *compound* tasks that need to be refined
  for which the model contains “methods”, the refinement rules.
- The solution is an executable, primitive task network (refinement).



Considered Types of Planning Problems: HTN Planning by Example




```
(:task makeClear :parameters (?b - block))

(:method one-step
 :parameters (?b1 ?b2 - block)
 :task (makeClear ?b1)
 :precondition (and (on ?b2 ?b1))
 :ordered-tasks (and (makeClear ?b2)
                     (unstack ?b2 ?b1)
                     (putdown ?b2)))
```

makeClear(C):

makeClear(A)
unstack(A,B)
putdown(A)
makeClear(B)
unstack(B,C)
putdown(B)

Problem Statement

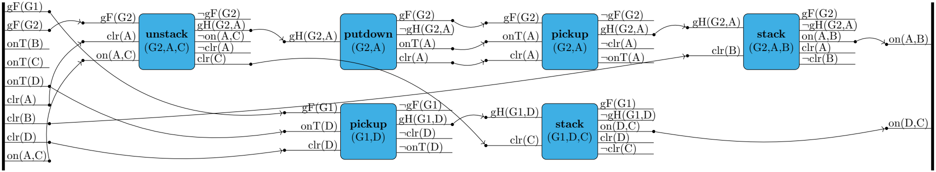
Input/Output Plans

- Input in most cases: action sequences!
  (Simply because that's what most algorithms produce.)
- Output: partially ordered ones, mostly.
  (See next slides.)

Input/Output Plans: Partial Order Causal Link (POCL) Plans

POCL plans are used as basis for several optimization techniques!

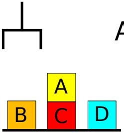
- They contain a set of ordering constraints
- They contain a set of causal links to ensure executability



Important properties:

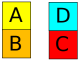
- In POCL plans, every linearization is executable
- There are some PO plans (where every linearization is executable), for which no corresponding POCL plan exists with the same ordering constraints/linearizations (cf. paper).
- They allow for parallelism; the makespan here is 4

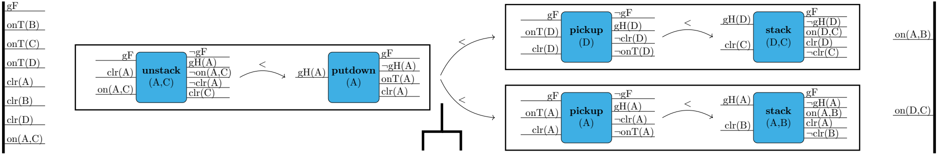
Input/Output Plans: Block-decomposed PO (BDPO) Plans



Another generalization of action sequences are BDPO plans:

- Here, we define ordering constraints between *blocks*
- Every linearization of the blocks is executable





Important properties:

- Blocks can contain blocks, so the definition is recursive
- BDPO plans can express more linearizations than POCL plans:
  - In Blocks World with one gripper, there can't be parallelism
  - Yet, here we have a partial order – but no parallelism
  - (In the last plan, there were two grippers (G1 and G2) available, hence the partial order.)

Problem Statement: Optimization Tasks

One usually optimizes one of two things:

- Minimize number of actions or action costs
  (Different notions of (sub)optimality exist)
- Optimize Orderings:
  - Maximize number of linearizations,
    usually done by minimizing ordering constraints
  - Minimize makespan (also done by removing orderings)

Problem Statement: What changes are allowed/done?

For optimizing ordering constraints, one can:

- just *delete* ordering constraints, called *deordering*, or
- *change* ordering constraints, called *reordering*

→ Sometimes we can only remove orderings after removing actions.

Problem Statement: What changes are allowed/done?

For optimizing ordering constraints, one can:

- just *delete* ordering constraints, called *deordering*, or
- *change* ordering constraints, called *reordering*

→ Sometimes we can only remove orderings after removing actions.

For the minimization of plans (actions),

- we can just *remove* actions, or
- we can *replace* actions/subplans.

→ Sometimes, we can only remove actions after reordering!

Main Content

Main Content

- Related topics, e.g.,
  - is branch and bound a solution to our problem?
    (we could take the length of the input plan as first bound!)
  - plan repair often does (is!) almost the same!

Main Content

- Related topics, e.g.,
  - is branch and bound a solution to our problem?
    (we could take the length of the input plan as first bound!)
  - plan repair often does (is!) almost the same!
- Complexity results for all these questions, e.g.,
  - is there a subplan that works?
  - is there a de-/reordering with k or less ordering constraints?
  - is there a de-/reordering with makespan k?

  $\rightarrow$ “perfect justification” is NP-complete (and many more)

Main Content

- Related topics, e.g.,
  - is branch and bound a solution to our problem?
    (we could take the length of the input plan as first bound!)
  - plan repair often does (is!) almost the same!
- Complexity results for all these questions, e.g.,
  - is there a subplan that works?
  - is there a de-/reordering with k or less ordering constraints?
  - is there a de-/reordering with makespan k?
  - → "perfect justification" is NP-complete (and many more)
- Optimization techniques for all these questions, i.e.,
  - for optimizing plans (actions) and
  - for linearizations/ordering constraints.

Main Content

- Related topics, e.g.,
  - is branch and bound a solution to our problem? (we could take the length of the input plan as first bound!)
  - plan repair often does (is!) almost the same!
- Complexity results for all these questions, e.g.,
  - is there a subplan that works?
  - is there a de-/reordering with k or less ordering constraints?
  - is there a de-/reordering with makespan k?
  - → “perfect justification” is NP-complete (and many more)
- Optimization techniques for all these questions, i.e.,
  - for optimizing plans (actions) and
  - for linearizations/ordering constraints.

Reminder: all this both for classical and hierarchical planning

Techniques for Removing/Replacing Actions

An incomplete list sneak-peek:

- Fink and Yang [1992], authors of one of the landmark papers in plan optimization, propose various degrees of redundancy (and algorithms), some in P.

Techniques for Removing/Replacing Actions

An incomplete list sneak-peek:

- Fink and Yang [1992], authors of one of the landmark papers in plan optimization, propose various degrees of redundancy (and algorithms), some in P.
- Removing redundant actions:
  - Encodings exist via MaxSAT, weighted MaxSAT, and planning. Especially the former build on POCL plans.
  - Also "algorithms" exist (by several groups).
  - For HTN planning, one approach bases on grammar parsing.
- Replacing subplans:
  - Some approaches again base on SAT and planning; the latter uses BDPO plans.

Techniques for Removing/Replacing Actions

An incomplete list sneak-peek:

- Fink and Yang [1992], authors of one of the landmark papers in plan optimization, propose various degrees of redundancy (and algorithms), some in P.
- Removing redundant actions:
  - Encodings exist via MaxSAT, weighted MaxSAT, and planning. Especially the former build on POCL plans.
  - Also ”algorithms” exist (by several groups).
  - For HTN planning, one approach bases on grammar parsing.
- Replacing subplans:
  - Some approaches again base on SAT and planning; the latter uses BDPO plans.

Again, there are *plenty* more!

Techniques for Improving Orderings/Linearizations

Some key messages:

- Again, encodings exist into MaxSAT, MIP, and CSPs.
- Some solve:
  - the NP-complete perfect justification,
  - the weaker polytime justifications, and
  - some use P-approximations to the NP-complete problem.
  - *The P-approximation was extremely strong in the tested benchmark domains, finding optimal results in most cases.*

Conclusion

High-level summary:

- We looked into complexity results and practical techniques
- Both for optimizing actions and orderings
- Both for classical and hierarchical planning (but mostly classical)

Look into the paper! :)

And see me at the poster.

$\rightsquigarrow$ Thank you! :)