

An Introduction to Hierarchical Task Network (HTN) Planning: Theoretical Foundations & Problem Solving

Pascal Bercher

School of Computing
College of Engineering, Computing and Cybernetics
The Australian National University

October 23, 2024



Introduction

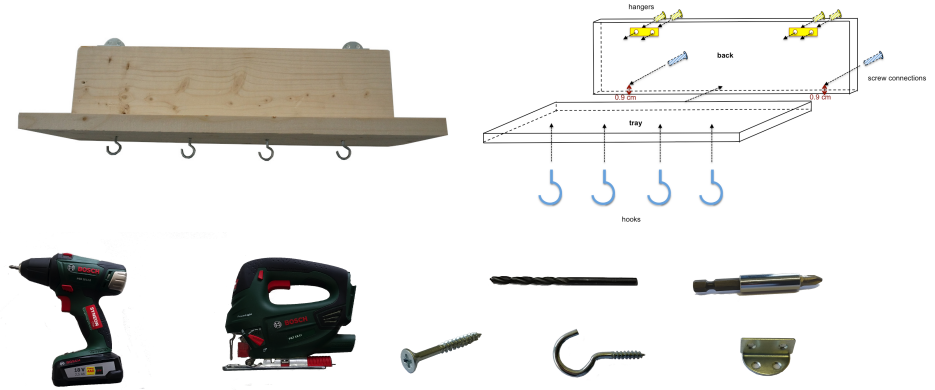


HTN Planning-based Do-It-Yourself (DIY) Assistant

HTN Planning-based Do-It-Yourself (DIY) Assistant



HTN Planning-based Do-It-Yourself (DIY) Assistant: Overview

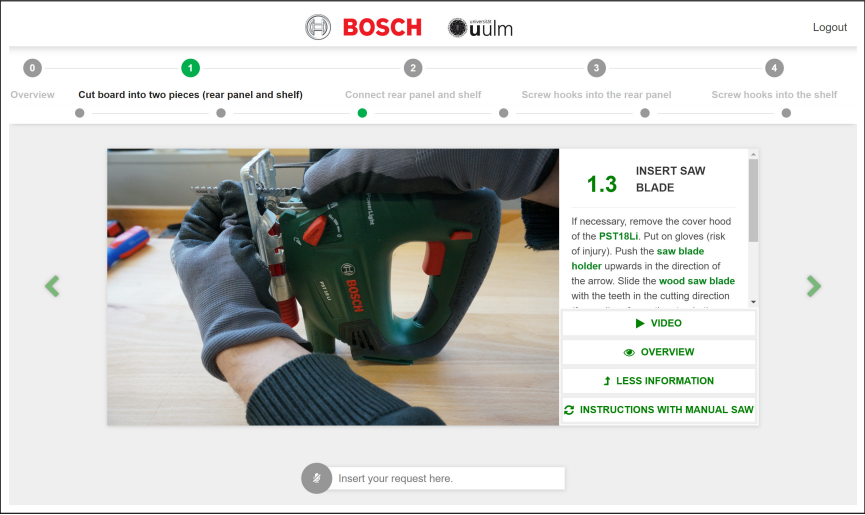


The material:

- Boards (need to be cut first)
- Electrical devices like drills and saws
- Attachments like drill bits and materials like nails



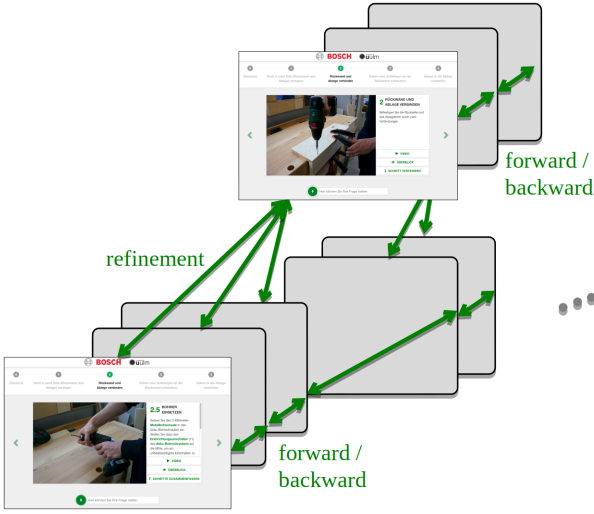
HTN Planning-based Do-It-Yourself (DIY) Assistant: User Interface



HTN Planning-based Do-It-Yourself (DIY) Assistant: Task Hierarchy

Abstract Level

Detailed Level



Talk Outline

Talk Outline

Talk Outline

What is this “talk”, and what not?

- Is it a scientific talk? → Absolutely not!
- Is it a tutorial? → Yeah, on some slides...
- Is it a survey / overview? → Mostly!

So what do I want to achieve? That you ... get hooked! :)

- understand the core differences to non-hierarchical planning,
 - Is it another algorithm? Or another problem class?
 - Are task hierarchies advice or constraints?
- know what makes it harder/easier,
- have a basic understanding of progression search,
- obtain a high-level overview of:
 - different solving techniques
 - heuristics (for search)
 - other stuff that’s going on in HTN planning

Introduction
○○○○○○○

HTN Problem Definition
●○○○○○○○○○○○○


Expressivity
○○○○○○○

Complexities
○○○○○○○○○

Solving Problems
○○○○○○○○○

Rounding it Up
○○○

HTN Problem Definition



Australian National University

Pascal Bercher

8.46

Introduction
○○○○○○○

HTN Problem Definition
●●○○○○○○○○○○○

Expressivity
○○○○○○○

Complexities
○○○○○○○○○

Solving Problems
○○○○○○○○○

Rounding it Up
○○○

Formalism

Formalism



Australian National University

Pascal Bercher

9.46

Introduction
○○○○○○○

HTN Problem Definition
●●○○○○○○○○○○○

Expressivity
○○○○○○○


Complexities
○○○○○○○○○

Solving Problems
○○○○○○○○○


Rounding it Up
○○○

Formalism: Introduction to HTN Planning

primitive tasks




compound tasks



$\mathcal{P} = (F, N_P, \delta, N_C, M, s_I, c_I, g)$

- F a set of facts
- N_P a set of primitive task names
- $\delta : N_P \rightarrow (2^F)^3$ the task name mapping
- N_C a set of compound task names



Australian National University

Pascal Bercher

10.46

Introduction
○○○○○○○

HTN Problem Definition
●●○○○○○○○○○○○

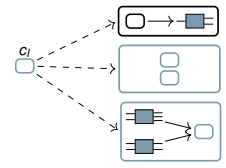
Expressivity
○○○○○○○

Complexities
○○○○○○○○○

Solving Problems
○○○○○○○○○

Rounding it Up
○○○

Formalism: Introduction to HTN Planning




$\mathcal{P} = (F, N_P, \delta, N_C, M, s_I, c_I, g)$

- F a set of facts
- N_P a set of primitive task names
- $\delta : N_P \rightarrow (2^F)^3$ the task name mapping
- N_C a set of compound task names
- $c_I \in N_C$ the initial task
- $M \subseteq N_C \times 2^{TN}$ the methods

We must find a task network tn , such that:

- it is a refinement of c_I ,
- only contains primitive tasks, and



Australian National University

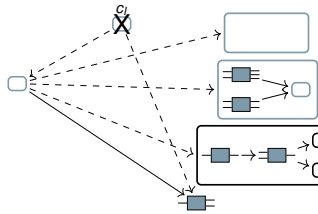
Pascal Bercher

10.46

Formalism: Introduction to HTN Planning

$$\mathcal{P} = (F, N_P, \delta, N_C, M, s_I, c_I, g)$$

- F a set of facts
- N_P a set of primitive task names
- $\delta : N_P \rightarrow (2^F)^3$ the task name mapping
- N_C a set of compound task names
- $c_I \in N_C$ the initial task
- $M \subseteq N_C \times 2^{TN}$ the methods



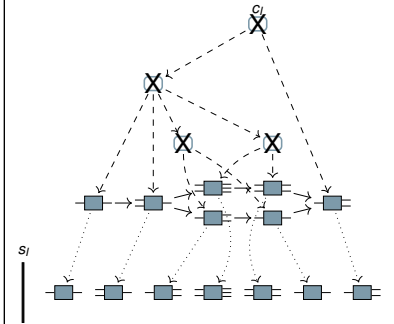
We must find a task network tn , such that:

- it is a refinement of c_I ,
- only contains primitive tasks, and

Formalism: Introduction to HTN Planning

$$\mathcal{P} = (F, N_P, \delta, N_C, M, s_I, c_I, g)$$

- F a set of facts
- N_P a set of primitive task names
- $\delta : N_P \rightarrow (2^F)^3$ the task name mapping
- N_C a set of compound task names
- $c_I \in N_C$ the initial task
- $M \subseteq N_C \times 2^{TN}$ the methods
- $s_I \in 2^F$ the initial state
- $g \subseteq F$ the (optional) goal description



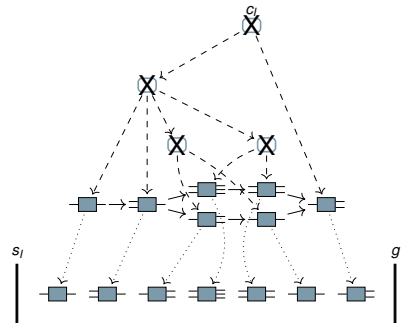
We must find a task network tn , such that:

- it is a refinement of c_I ,
- only contains primitive tasks, and
- has an executable linearization that makes the goals in g true.

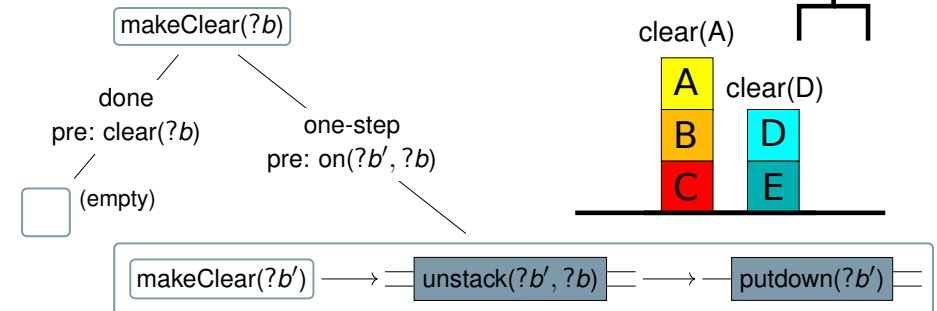
Formalism: HTN Planning: Solution Criteria in more Detail

An action sequence $\bar{p} \in N_P^*$ is a solution if and only if:

- There is a sequence of decomposition methods \bar{m} that transforms c_I into some tn ,
- tn contains only primitive tasks (those in \bar{p}), and
- tn (still partially ordered) admits \bar{p} as linearization, is executable, leads to a goal state $s \supseteq g$.



Example: Example Domain Excerpt: Blockworld and HDDL



```
(: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)

Introduction ○○○○○○○	HTN Problem Definition ○○○○●○○○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--------------------------------------	-----------------------	---------------------------	-----------------------------	-----------------------

PO vs. TO

PO vs. TO

Australian National University

Pascal Bercher

13.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○●○○○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--------------------------------------	-----------------------	---------------------------	-----------------------------	-----------------------

PO vs. TO: On the (Non?)equivalence of PO and TO HTN models

Each partially ordered task network is just a compact representation of its linearizations, right?

$A \rightarrow B \rightarrow \begin{cases} C \\ D \end{cases}$

=

$A \rightarrow B \rightarrow C \rightarrow D$

+

$A \rightarrow B \rightarrow D \rightarrow C$

?

Let:

$C \dashrightarrow C1$

$D \dashrightarrow D1 \rightarrow D2$

Can we create the following task network?

$A \rightarrow B \rightarrow D1 \rightarrow C1 \rightarrow D2$

No! Not anymore...

Australian National University

Pascal Bercher

14.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○●○○○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--------------------------------------	-----------------------	---------------------------	-----------------------------	-----------------------

PO vs. TO: Conclusion, Trivia, and more

Some facts on PO vs. TO:

- We just saw: PO is *not* the same as all linearizations!
 - Thus, a TO HTN model isn't just a "larger" PO model.
 - Thus, PO models can't be compiled into TO models (in general) (Though the IPC-23 PO-track winner does exactly that!)
- This is confirmed by complexity theory, because:
 - TO-HTN problems are known to be in EXPTIME (proved by Erol et al. '94, and proved complete by Alford et al. '15)
 - whereas PO-HTN problems are known to be undecidable (proved by Erol et al. '94), so we can't compile!
- This is also confirmed by expressivity studies, because:
 - TO-HTN models can express *exactly* context-free (CF) languages (proved by Höller et al. '14)
 - PO-HTN models can express *strictly more* than CF languages, but strictly less than context-sensitive ones (proved by Höller et al. '14, '16)

Australian National University

Pascal Bercher

15.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○●○○○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--------------------------------------	-----------------------	---------------------------	-----------------------------	-----------------------

PO vs. TO: Conclusion, Trivia, and more (cont'd)

Some facts on PO vs. TO: // (cont'd)

- Most existing domains seem to be TO:
 - 9 PO domains in the HTN track of IPC 2020.
 - 24 TO domains (again IPC 2020)
 - In the HTN 2023 IPC 2 more domains were added to each track.
- There are many specialized TO HTN planners! (And techniques)
 - IPC 2020:
 - ▶ 3 PO planners
 - ▶ 3 (additional) TO planners
 - IPC 2023:
 - ▶ 4 PO planners (compared to 2020: two new ones; one dropped)
 - ▶ 2 (additional) TO planners (but one is a PO planner with TO pruning)
 - ▶ 1 technique that compiles PO into TO if possible
 - There are further PO and TO planners that did not participate.

Australian National University


Pascal Bercher

16.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○○○○○●○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--	-----------------------	---------------------------	-----------------------------	-----------------------

Alternative HTN Problem Definition

Alternative HTN Problem Definition



Australian National University

Pascal Bercher

17.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○○○○○●○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--	-----------------------	---------------------------	-----------------------------	-----------------------

Alternative HTN Problem Definition: Formal Grammar/Language Recap


Recap from Theoretical Computer Science:

A *context-free grammar* G is a tuple $\langle N, \Sigma, S, R \rangle$ where

- N is a finite set of *non-terminal symbols*,
- Σ , disjoint from N , is a finite set of *terminal symbols* (Σ is also called *alphabet*),
- $S \in N$ is the *start symbol*,
- $R \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *production rules*.

Languages:

- A language L is any (possibly infinite) set of words (sequences of symbols). E.g., the sets \emptyset , $\{abc, \dots, xyz\}$, and \mathbb{N} are languages.
- The *language of a grammar*, $L(G) \subseteq \Sigma^*$, is the set of terminal words obtainable by refining S by only using production rules.



Australian National University


Pascal Bercher

18.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○○○○○●○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--	-----------------------	---------------------------	-----------------------------	-----------------------

Alternative HTN Problem Definition: Formal Grammar/Language Recap, Example

- Let $G = \langle \{a, b\}, \{S, A, B\}, S, \{S \rightarrow aB, B \rightarrow Ab, A \rightarrow S, A \rightarrow \epsilon\} \rangle$, so we have:
 - Terminal symbols: $\{a, b\}$
 - Non-terminals: $\{S, A, B\}$
 - Start symbol: S
 - Production rules:
 - $S \rightarrow aB$
 - $B \rightarrow Ab$
 - $A \rightarrow S \mid \epsilon$
- Some example derivations:
 - $S \rightarrow aB \rightarrow aAb \rightarrow ab$
 - $S \rightarrow aB \rightarrow aAb \rightarrow aSb \rightarrow \dots \rightarrow aabb$
- So, the language of G is $L(G) = \{a^n b^n \mid n \geq 1\}$



Australian National University

Pascal Bercher

19.46

Introduction ○○○○○○○	HTN Problem Definition ○○○○○○○○●○○○	Expressivity ○○○○○	Complexities ○○○○○○○○○	Solving Problems ○○○○○○○	Rounding it Up ○○○
-------------------------	--	-----------------------	---------------------------	-----------------------------	-----------------------


Alternative HTN Problem Definition: Based on Grammars/Languages

- Actions were defined by their name: $\delta : N_P \rightarrow 2^F \times 2^F \times 2^F$. Thus, solutions are (the same as) sequences of task names.
- Thus, any solution set $sol(\mathcal{P})$ is a language. Let:
 - $L_H(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}'), \text{ where } \mathcal{P}' \text{ ignores all facts}\}$
 - $L_C(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}'), \text{ where } \mathcal{P}' \text{ is the induced classical problem}\}$
- Now we can decompose the solution criteria:
 - L_H just looks at the word produced by the hierarchy,
 - L_C just looks at the executable words that produce the goal. $\rightarrow sol(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$

This observation gives a new/simplified view on HTN planning:

HTN planning = classical planning + grammar to filter solutions

!! Maybe the most important statement of this presentation !!



Australian National University

Pascal Bercher

20.46

Expressivity

Motivation

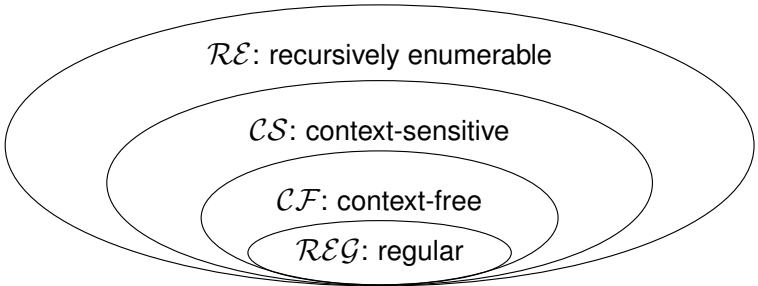
Again, we have seen that $sol(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

Why is that useful?

- Consider $L_H = (a(b|c)d)^*$ with a, b, c, d actions. We now know that even if aa or ad would be executable, no plan can ever contain those!
- This might be important in safety-critical scenarios, where full control over plans is essential. Consider the DIY scenario from earlier! We can control better when or where saw blades are switched and instructions are shown etc.
- So HTN planning is becomes very useful (or important) when the execution of actions carries more meaning than the world properties these actions establish!

Chomsky Hierarchy

The Chomsky Hierarchy defines a hierarchy of expressiveness.



For example, we know that:

- The context-free languages are exactly those for which there is a context-free grammar. \rightarrow Thus $\{a^n b^n \mid n \geq 1\}$ is context-free.
- Regular languages are exactly those for which there exists a finite automaton. \rightarrow Thus $\{a^n b^n \mid n \geq 1\}$ is *not* regular.

Classes of Planning Problems

We can define the following Language classes:

- Let $HTN = \{sol(\mathcal{P}) \mid \mathcal{P} \text{ is an HTN planning problem.}\}$
- Let $CLASSIC = \{sol(\mathcal{P}) \mid \mathcal{P} \text{ is a classical planning problem.}\}$
- We can do the same for any restriction on planning problems:
 - $TOHTN = \{sol(\mathcal{P}) \mid \mathcal{P} \text{ is a TO HTN planning problem.}\}$
 - and for any other restriction!

What's the idea/purpose?

- Now we can also classify HTN , $TOHTN$, etc. within the Chomsky Hierarchy.
- E.g., can $CLASSIC$ express solutions of the form $\langle a^n, b, b, c^n \rangle$?
Answer: No! We need HTN planning! :)

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ (by Höller et al. '14)

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathcal{REG}$
 - Notice that each planning problem encodes an (exponentially larger) DFA: Each node is a state, each edge is an action.
 - We know that each DFA is regular, thus showing the claim.
- We now show $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$.
 - We prove that $\{aa\} \in \mathcal{REG}$ is not the language of any classical problem \mathcal{P} , $sol(\mathcal{P}) \neq \{aa\}$ for all \mathcal{P} .
 - Assume $aa \in sol(\mathcal{P})$ for some classical problem \mathcal{P} . We know that a leads to a state in which a is executable (since aa is executable). Hence, aaa must be executable as well! But then $aaa \in sol(\mathcal{P})$, so $\{aa, aaa\} \subseteq sol(\mathcal{P})$, and hence $sol(\mathcal{P}) \neq \{aa\}$.

Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathcal{CF}$ (by Höller et al. '14)

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathcal{CF}$.
 - Any context-free grammar *is* an HTN problem! (With $F = \emptyset$)
- Now we show $\mathcal{TOHTN} \subseteq \mathcal{CF}$.
 - We know that $sol(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$ for all HTN problems \mathcal{P} .
 - We know that $L_H(\mathcal{P})$ is context-free and that $L_C(\mathcal{P})$ is regular.
 - It is known that the intersection of a context-free and regular language is context-free.

Complexities

General Case

General Case

General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. (by Erol et al., '94)

We reduce from the (undecidable) grammar intersection problem. Given context-free grammars G and G' , construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$. using the following decision procedure:

- Construct an HTN planning problem \mathcal{P} that has a solution if and only if the correct answer is yes.
- Translate the production rules to decomposition methods in a way that only words in both $L(G)$ and $L(G')$ can be produced.
- Our desired primitive task network tn contains only one executable linearization $\omega = \omega_1, \omega_2, \dots, \omega_{2n-1}, \omega_{2n}$:
 - $\omega^1 = \omega_1, \omega_3, \dots, \omega_{2n-1}$, $|\omega^1| = n$, and $\omega^1 \in L(G)$
 - $\omega^2 = \omega_2, \omega_4, \dots, \omega_{2n}$, $|\omega^2| = |\omega^1|$, and $\omega^2 \in L(G')$

We show the encoding using an example.

General Case: Example Reduction

Let $G = (\overset{\text{non-terminals}}{N = \{H, Q\}} , \overset{\text{terminals}}{\Sigma = \{a, b\}} , \overset{\text{rules}}{R} , \overset{\text{start symbol}}{H})$
and $G' = (N' = \{D, F\} , \Sigma' = \{a, b\} , R' , D)$.

Production rules R : $H \mapsto aQb$ $Q \mapsto aQ \mid bQ \mid a \mid b$
Production rules R' : $D \mapsto aFD \mid ab$ $F \mapsto a \mid b$

$\mathcal{P} = (\{v_G, v_{G'}, v_a, v_b\}, \overset{N_G}{\{H, Q, D, F\}}, \overset{N_P}{\{a, b, a', b'\}}, \delta, M, \overset{\text{initial state}}{\{v_G\}}, tn_I, \overset{\text{goal description}}{\{v_G\}})$
 $\delta = \{ a \mapsto (\{v_G\}, \{v_{G'}, v_a\}, \{v_G\}),$
 $\phantom{\delta = \{ } b \mapsto (\{v_G\}, \{v_{G'}, v_b\}, \{v_G\}),$
 $\phantom{\delta = \{ } a' \mapsto (\{v_{G'}, v_a\}, \{v_G\}, \{v_{G'}, v_a\}),$
 $\phantom{\delta = \{ } b' \mapsto (\{v_{G'}, v_b\}, \{v_G\}, \{v_{G'}, v_b\}) \}$
 $M = M(G) \cup M(G')$ (translated production rules of G and G')
 $tn_I = (\underbrace{\{t, t'\}}_T, \underbrace{\emptyset}_{\prec}, \underbrace{\{t \mapsto H, t' \mapsto D\}}_{\alpha})$

Complexities of Special Cases

Complexities of Special Cases

Complexities of Special Cases: More known Complexity results

Hierarchy	Order	Class
general	PO	undecidable (Erol et al., '94)
	TO	EXPTIME (Erol et al., '94, Alford et al., '15)
acyclic	PO	NEXPTIME (Alford et al., '15)
	TO	PSPACE (Alford et al., '15)
tail-rec	PO	EXPSpace (Alford et al., '15)
	TO	PSPACE (Alford et al., '15)
regular	*	PSPACE (Erol et al., '94)

Additional relaxations:

What is relaxed away?	Order	Class
delete-effects	*	NP (Alford et al., '14)
... and preconditions	TO	NP & FPT (Olz & Bercher, '23)

... More results exist!
(e.g., landmarks, inferred precs/effs, lifted formalism)

TIHTN Planning

- Let $\mathcal{P} = (F, N_P, \delta, N_C, M, s_I, c_I, g)$ be an HTN planning and $sol_{HTN}(\mathcal{P})$ its set of HTN solutions.
- HTN planning with task insertion (TIHTN Planning) allows to insert tasks anywhere! (Geier & Bercher, '11)
 - Pros: Less effort designing the task hierarchy
 - Cons: Less control over generated plans (less expressivity)
- A TIHTN Problem \mathcal{P} is an HTN problem, such that $sol_{TIHTN}(\mathcal{P}) = \{\bar{p} \mid \bar{p} \text{ is executable and leads to a } s \supseteq g \text{ and there exists some } \bar{p}' \subseteq \bar{p}, \text{ with } \bar{p}' \in L_H(\mathcal{P})\}$, i.e., $sol_{HTN}(\mathcal{P}) \subseteq sol_{TIHTN}(\mathcal{P}) = L_C(\mathcal{P}) \cap \{\bar{p} \mid \exists \bar{p}' \in L_H(\mathcal{P}), \bar{p}' \subseteq \bar{p}\}$
- Thus, TIHTN planning still requires decomposition, but we can insert actions wherever possible.

Main observation: In TIHTN planning, cycles are pointless! (Geier & Bercher, '11)

Additional relaxations:

35.46

Solving Problems

Overview

There exist many algorithms to solve HTN problems:

- Heuristic search via:
 - progression (similar to state-based planning) for PO (too many to list!)
 - plan space search (similar to POCL planning) for PO (too many to list!)
- Monte Carlo Tree Search for PO (Wichlacz et al., '20)
- Compilation into:
 - Logic for PO/TO (Behnke et al., '18++ and Schreiber et al., '19/'21)
 - Answer Set Programming for TO (Dix et al., '03)
 - Binary Decision Diagrams for TO (Behnke & Speck, '21)
 - Classical Planning for PO (Alford et al., '09++, Behnke et al., '22)
 - Classical Planning for TO (Höller, '21)

In this talk:

- Progression search (because it's simple)
- (Very) High-level overview of heuristics

Progression Search

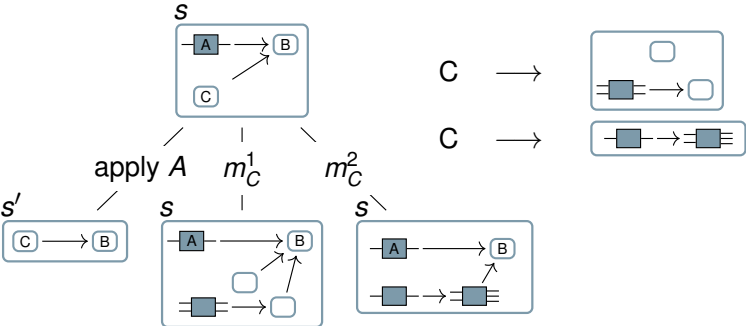
Progression Search

Progression Search: Overview

Trivia/History:

- Progression search is deployed by *several* HTN systems (see, e.g., the IPCs 2020 and 2023)
- Simple Hierarchical Ordered Planner (SHOP) is a famous family of planners:
 - First version is for TO models (Nau et al., '99)
 - SHOP2 (and successors) can deal with PO (Nau et al., '03)
 - SHOP3 (Goldman & Kuter, '19)
 - They all support method preconditions
- Later it was shown that SHOP search is unsystematic and how that can be prevented (Höller et al., '18/'20)

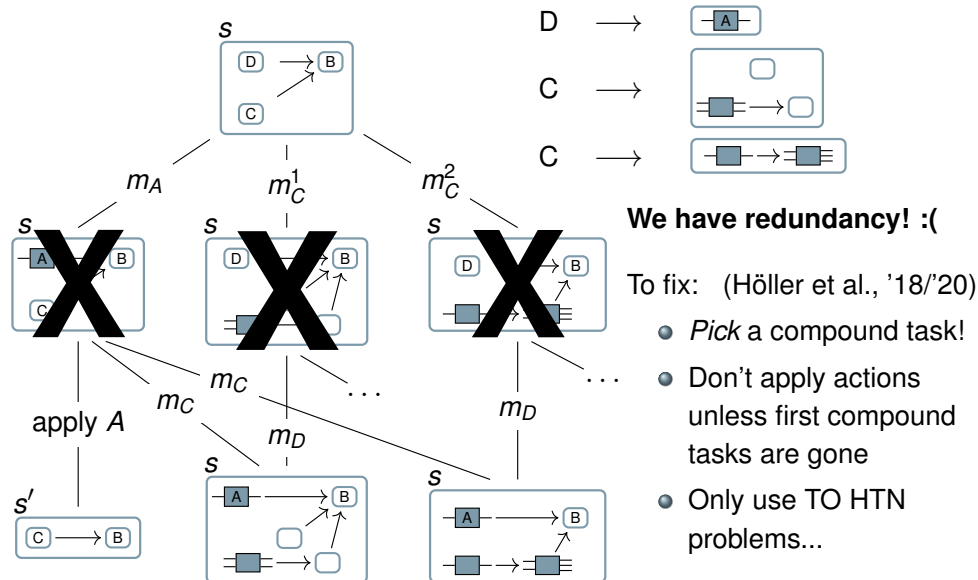
Progression Search: By Example



Textbook progression: *(deployed in many planners and papers)*

- *node selection*: standard A*, BFS, or whatever!
- *node expansion*:
 - Identify all “first tasks” (those without predecessor)
 - branch over *all* their successors
- *solution*: Sequence of progressions to empty task network!

Progression Search: We can do better! (Make search systematic)



Heuristics

Heuristics

Heuristics: Overview

- TDG heuristic (Bercher et al., '17)
 - in P: task insertion, delete relaxation, ordering relaxation.
 - But doesn't count those inserted actions – **not** well informed.
- HTN node → classical problem → heuristic (Höller et al., '18/'20)
 - in P: task insertion, delete relaxation, ordering relaxation.
 - More general and informed than the previous. Best-performing one.
- Landmark counting heuristic (Höller & Bercher, '21)
 - in P: delete relaxation, ordering relaxation (and computes only a subset)
 - Note very informed in its original version.
- Delete- and Ordering-Relaxation as ILP (Höller et al., '20)
 - NPC: delete relaxation, ordering relaxation.
 - More informed, but very expensive (does not pay out).

Take-aways:

- No heuristic exploits any special case, such as TO, acyclic, etc.
- All heuristics but one rely on task insertion, none uses orderings

→ **lots of potential for novel heuristics !!**

Rounding it Up

What was *not* mentioned yet?

- There exist *many* HTN formalisms!
 - Lifted formalisms, method preconditions, time, uncertainly, state constraints, hybrid (HTN+POCL), TIHTN planning, ...
 - Hierarchical Goal Network (HGN) planning, where goals (facts) are decomposed (Shivashankar et al., '12++)
- Research directions within HTN planning: pretty much exactly the same as in classical planning! E.g.,
 - plan verification and optimization
 - plan explanation
 - plan repair, replanning
 - model learning and modeling support
 - applications
 - many more...
- Some resources can be found online:
<https://hierarchical-task.net> (apologies if incomplete)

Disclaimer and thank you!

Disclaimer:

- I tried to provide width and be comprehensive, but not complete
- I apologize if I missed some work (especially if it's "yours")

I hope I convinced some to:

- play around with HTN systems, there are so many!
- maybe do some research on it :)

THANK YOU FOR YOUR ATTENTION !!