Flexible FOND HTN Planning: A Complexity Analysis

Dillon Z. Chen, Pascal Bercher

The Australian National University

{dillon.chen, pascal.bercher} @anu.edu.au

FOND^{MP} HTN problem

A task network is a **directed acyclic graph** of tasks.

A FOND^{MP} HTN problem has the form $P = \langle F, N_p, N_c, \delta, M, s_I, \mathsf{tn}_I \rangle$

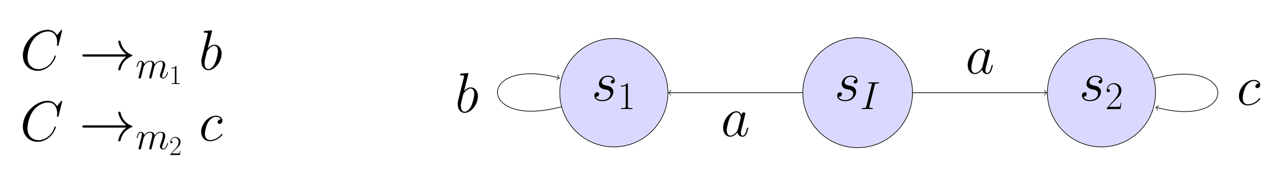
- F is a set of facts, of which a subset is a state
- N_p is a set of primitive task names
- N_c is a set of compound task names
- δ maps primitive task names to **nondeterministic actions**
- M maps compound task names to task networks
- $s_I \subseteq F$ is an initial state
- tn_I is an initial task network

FOND^{MP} HTN solution

A FOND^{MP} HTN solution consists of a **policy** of task selection on task network-state tuples $\sigma_\alpha = (\mathsf{tn}_\alpha, s_\alpha)$ which either

1. executes a first primitive task $t \in \mathsf{tn}_\alpha$ applicable to s_α, *or*
2. decomposes a first compound task $t \in \mathsf{tn}_\alpha$

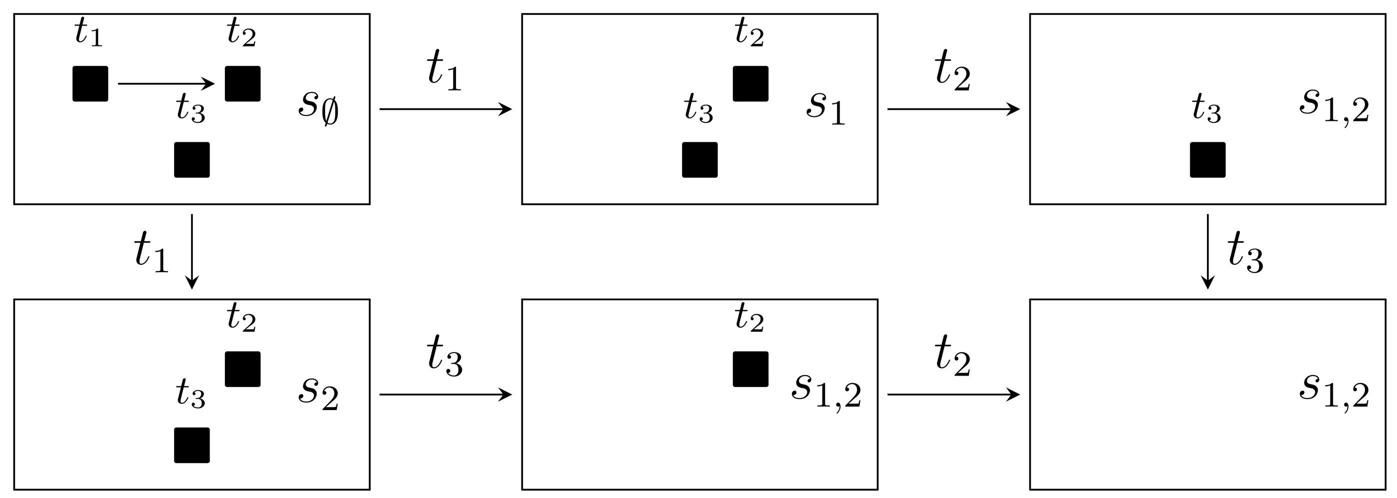
Note: for well definedness, input task networks for a policy are quotiented out by their **(task network) isomorphism class**



A FOND^{MP} HTN problem example: $\mathsf{tn}_I = a \to C$ where a is a primitive nondeterministic task applicable at s_I with successors s_1 and s_2, and C is a compound task and may decompose into either b or c. The tasks b and c can only be applied at s_1 and s_2 respectively.

Isn't task network/graph isomorphism hard?

- *Maybe* in the general case. <u>*No in the practical case*</u>.
- TN isomorphism solvers exist [Höller and Behnke 2021]
  - idea: hashing on layers of tasks in a task network
- `nauty`: fast graph isomorphism solver [McKay and Piperno 2014]
  - idea: individualisation and (colour) refinement



Visualisation of a FOND^{MP} HTN problem compiled into a nondet. search problem.

Connection to nondeterministic planning

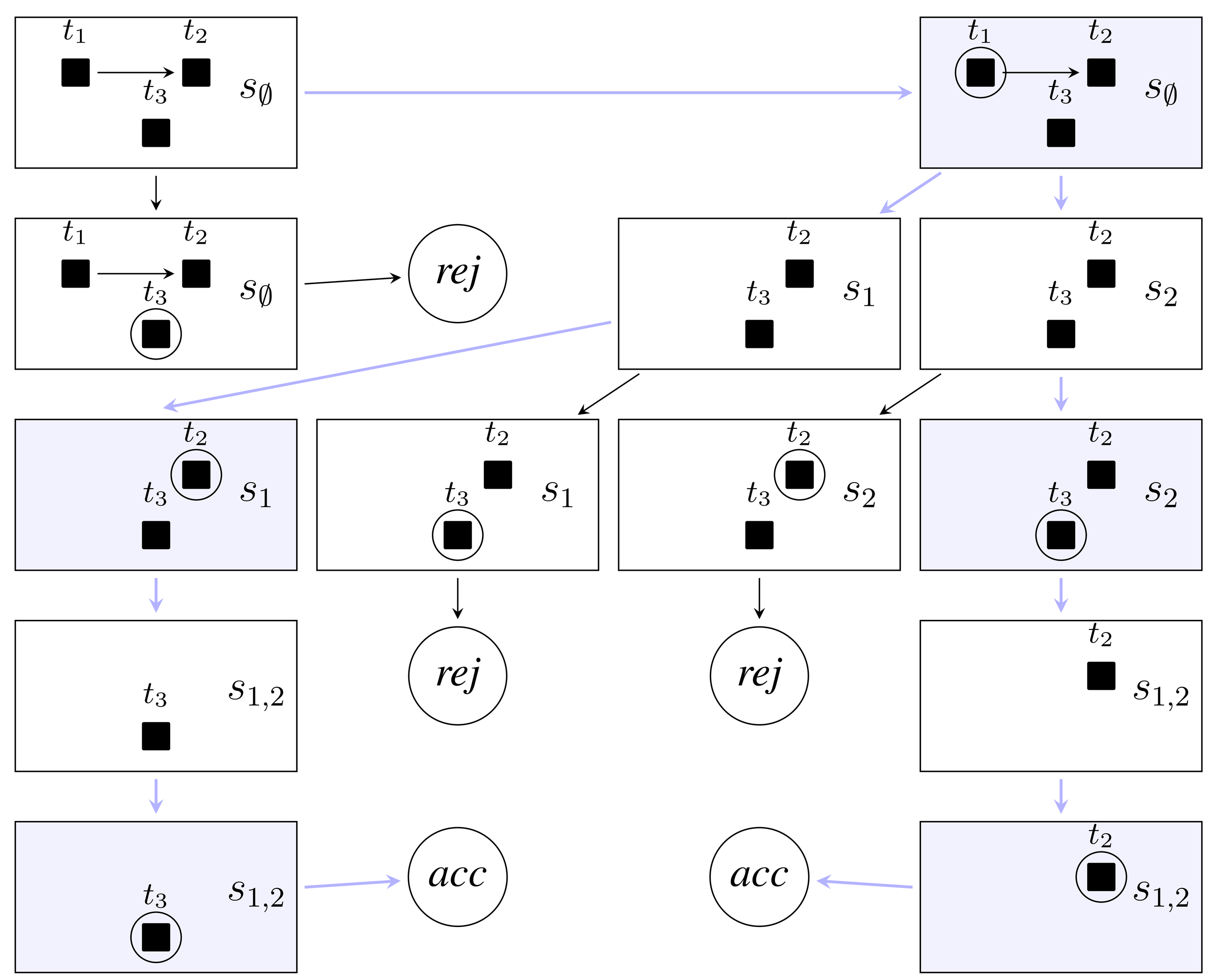
We can compile into a nondeterministic search problem as follows.

- a search node is a task network-state tuple $\sigma_\alpha = (\mathsf{tn}_\alpha, s_\alpha)$
- a search node can be viewed as a FOND^{MP} HTN **subproblem**
- transitions between search nodes correspond to choice of decomposition or primitive task transitions:
  - if a first task t in tn is primitive, define a nondet. transition

$$a = (\sigma_\alpha, \{\sigma_i = (\mathsf{tn}_\alpha \setminus \{t\}, s_i) \mid s_i \in \tau(t, s_\alpha)\})$$

  - else for each method applicable to t, define a nondet. transition

$$a = (\sigma_\alpha, \{\sigma_\beta = (\mathsf{tn}_\beta, s_\alpha)\}), \quad \text{s.t.} \quad \mathsf{tn}_\alpha \to^{t}_{m} \mathsf{tn}_\beta$$



Visualisation of a FOND^{MP} HTN problem being solved with AND-OR search.

Simple algorithms

- Compile a FOND^{MP} HTN problem into a nondet. search problem.
- Solve the nondeterministic search problem with any nondet. search problem solver.
- We consider the following two algorithms.
  1. AND-OR search
  2. backwards search [Cimatti et al. 2003]
- We use these algorithms for complexity membership proofs.

HTN subclasses

- FOND^{MP} HTN at least semidecidable
- divide HTN planning problems into subclasses based on
  1. order of task networks: total or partial
  2. hierarchy classes of task networks:
     - primitive: no compound tasks
     - acyclic: no compound task can reach itself with decomposition
     - regular: at most one compound task in each network and is the last task
     - tail-recursive: ~ acyclic + regular

Complexity results for FOND^{MP} HTN planning. The first column lists known special cases by restricting the hierarchy, where the general case is undecidable. Classes marked $*$ indicate membership only.

Hierarchy	Order	Weak	Strong	Strong cyclic
primitive	total	NP	P*	P*
	partial	NP	PSPACE	PSPACE
acyclic	total	PSPACE	EXPTIME	EXPTIME
	partial	NEXPTIME	EXPSPACE	EXPSPACE
regular	total	PSPACE	EXPTIME	EXPTIME
	partial	PSPACE	EXPTIME	EXPTIME
tail-recursive	total	PSPACE	EXPTIME	EXPTIME
	partial	EXPSPACE	2-EXPTIME	2-EXPTIME*

Complexity results

- *membership proofs idea*: use aforementioned simple algorithms
- *hardness proofs idea*: reductions from alternating Turing machines
- almost all problems made one class harder