Automated Action Planning

Relaxation and Relaxation Heuristics

Carmel Domshlak

Automated Action Planning

— Relaxation and Relaxation Heuristics

Heuristics for Planning

How do we come up with heuristics for general planning tasks?

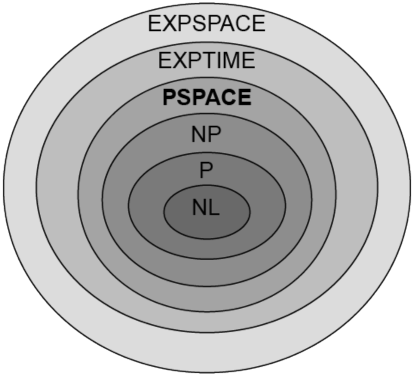
$\rightsquigarrow$ four major approaches in the literature:

- abstraction
- delete relaxation
- critical paths
- landmarks

But before we proceed ...

Computational Tractability (?!)

Planning Tasks and Worst-Case Complexity



deterministic $\rightsquigarrow$ PSPACE-complete

bounded deterministic $\rightsquigarrow$ NP-complete

Looks like the P island is irrelevant.
Still ... what makes planning problems hard?

Why complexity analysis?

- understand the problem
  $\rightsquigarrow$ *what for?*
- know what is not possible
  $\rightsquigarrow$ *well, that shouldn't be hard.*
- find interesting subproblems
  $\rightsquigarrow$ *and do with them what?*
- distinguish essential features from syntactic sugar
  $\rightsquigarrow$ *and take this understanding where?*

What Do We Mean by "Computational Tractability"?

Given a problem Π, ability to solve in polynomial time something useful for solving Π.

1. Ability to solve something in polynomial time.
2. Given a problem Π, ability to solve in polynomial time something useful for solving Π.
3. For a formalism F (model + language), find tractable fragments of F

$\rightsquigarrow$ *Useful?*

Why Computational Tractability?

Bylander, 1994

If the relationship between intelligence and computation is taken seriously, then intelligence cannot be explained by intractable theories because no intelligent creature has the time to perform intractable computations. Nor can intractable theories provide any guarantees about the performance of engineering systems.

- Point 1 is logical but vague (and thus misleading?)
  - What is the definition of "intractable theory"?
  - "Every science has a big lie. The big lie of complexity is worst case analysis." [C. Papadimitriou]
  - Still, worst case intractability severely limits us algorithmically
- Point 2 is a serious concern.

Some conclusions on Why Computational Tractability?

Concrete applications

- building systems with worst-case guarantees
- building new search guidance mechanisms
- combining a set of search guidance mechanisms
- checking whether new developments any needed (*)

Planning as State-Space Heuristic Search

Heuristic functions

What? Something that can be solved in polynomial time to assist us in solving our planning task

How? Solutions to simplifications of the planning task

Window of opportunity for computational tractability!

Finite Domain Representation (FDR) Language

Definition (FDR planning tasks)

An FDR planning task is a tuple $\langle V, A, I, G \rangle$

- V is a finite set of state variables with finite domains $dom(v_i)$
- initial state I is a complete assignment to V
- goal G is a partial assignment to V
- A is a finite set of actions a specified via pre(a) and eff(a), both being partial assignments to V

Definition (BDR planning tasks)

BDR planning tasks are FDR planning tasks with only boolean state variables.

Syntactic fragments

What are syntactic restrictions?

Fragment of tasks $\overset{\text{def}}{\longleftarrow}$ restrictions on action description (preconditions and effects)

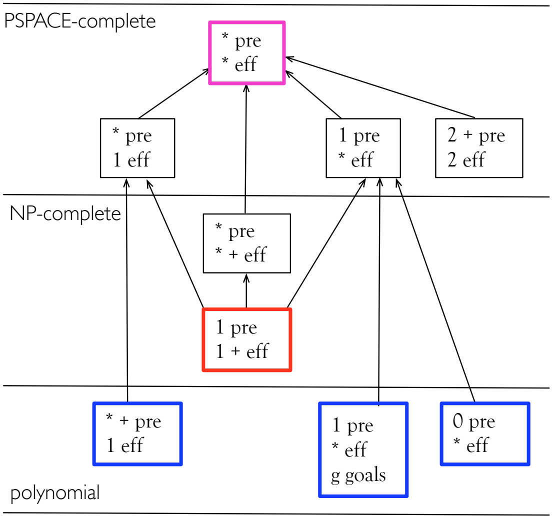
1. Restrictions on individual actions
2. Restrictions on action set as a whole

Note:

- Membership can be verified offline
- Membership can be verified in polynomial time (?)

Bylander's Map of BDR

PlanExt



NP-completeness of BDR^1_{1+}

Membership in NP by monotonicity of state updates.
Hardness by reduction from 3SAT. Let F be a 3CNF formula with n clauses over variables $U = \{u_1, \ldots, u_m\}$. An equivalent BDR^1_{1+} task can be constructed as follows.

- State variables $V = \{c_1, \ldots, c_n, \;\; t_1, \ldots, t_m, \;\; f_1, \ldots, f_m\}$.
- Initial state $I = \emptyset$ (all vars set to *false*).
- Goal $G = \bigwedge_{i=1}^{n} c_i$.
- Actions
  1. For each u_i, two actions: $\neg f_i \Rightarrow t_i$ and $\neg t_i \Rightarrow f_i$
  2. For $1 \leq j \leq n$,
     - if j-th clause contains u_i, then action $t_i \Rightarrow c_j$
     - if j-th clause contains $\overline{u_i}$, then action $f_i \Rightarrow c_j$

⊛ Suggests why HSP for STRIPS planning was stuck

Islands of Tractability

BDR_1^+

- How? Dedicated algorithm, forward + backward search.
  Search for an intermediate state that can be reached with only positive-effect actions, and from which the goal can be reach with only negative-effect actions.
- Example: Blocksworld. $\circledast$ General practice?

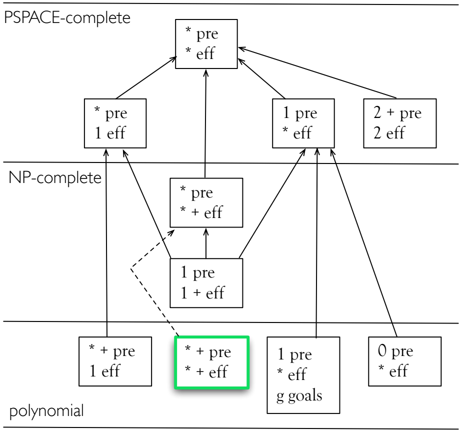
BDR^0

- How? Simple means-end analysis.
- $\circledast$ An advanced variant of "STRIPS heuristic" (missing goals counting).

BDR^1 limited to $g = O(1)$ goals

- How? Exhaustive search through a "small" search space.
  A single goal cannot expand into multiple sub goals

BDR_{+}^{+} is in P



⊛ Footnote 4: *"The following are other results that were left out of the figure because they were judged to be less interesting, but are listed here for completeness. ..."*

Back to heuristics!

Heuristics for Planning

How do we come up with heuristics for general planning tasks?

$\rightsquigarrow$ four major approaches in the literature:

- abstraction
- delete relaxation
- critical paths
- landmarks

What is relaxation?

Relaxations for planning

- Relaxation is a general technique for heuristic design:
  - Straight-line heuristic (route planning): Ignore the fact that one must stay on roads.
  - Manhattan heuristic (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore bad side effects of applying actions.

Example (8-puzzle)

If we move a tile from x to y, then the good effect is
(in particular) that x is now free.
The bad effect is that y is not free anymore, preventing us for moving tiles through it.

Relaxed planning tasks: idea

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good (add effects).
- Effects that make atoms false are bad (delete effects).

Idea for the heuristic: Ignore all delete effects.

Example: FreeCell

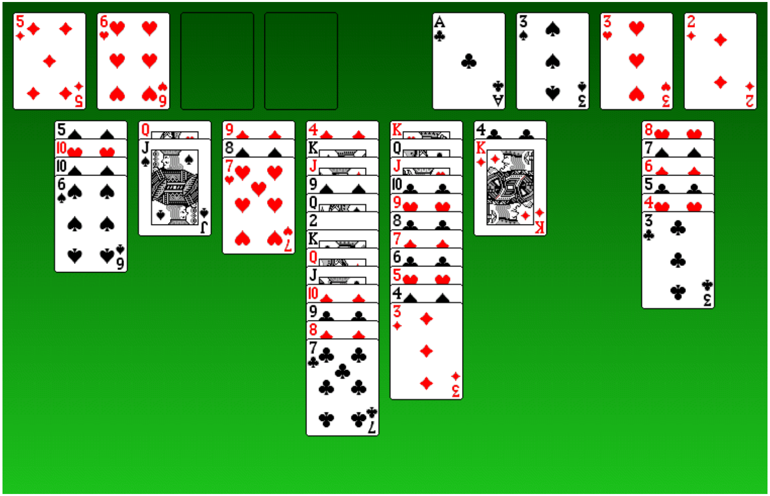


image credits: GNOME Project (GNU General Public License)

Planning Heuristics: Delete Relaxation

Four classes of heuristics:

2. Delete Relaxation

Estimate cost to goal by considering simpler planning task
without negative side effects of actions.

Example: Delete Relaxation in FreeCell

Problem constraints dropped by the delete relaxation in FreeCell:

- free cells and free tableau positions remain available after moving cards into them
- cards remain movable and remain valid targets for other cards after moving cards on top of them

Relaxed planning tasks

Definition (relaxation of actions)

The relaxation a^+ of a STRIPS action $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$ is the action $a^+ = \langle \mathrm{pre}(a), \mathrm{add}(a), \emptyset \rangle$.

Definition (relaxation of planning tasks)

The relaxation Π^+ of a STRIPS planning task $\Pi = \langle P, A, I, G \rangle$ is the planning task $\Pi^+ := \langle P, \{a^+ \mid a \in A\}, I, G \rangle$.

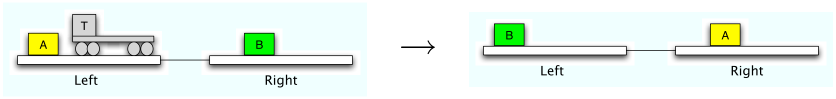
Definition (relaxation of action sequences)

The relaxation of an action sequence $\pi = a_1 \dots a_n$ is the action sequence $\pi^+ := {a_1}^+ \dots {a_n}^+$.

Relaxed planning tasks: terminology

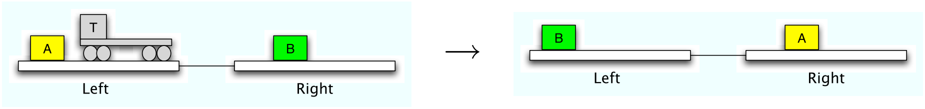
- STRIPS planning tasks without delete effects are called relaxed planning tasks.
- Plans for relaxed planning tasks are called relaxed plans.
- If Π is a STRIPS planning task and π^+ is a plan for Π^+, then π^+ is called a relaxed plan for Π.

Example: Logistics



- Initial state I: $\{at(A, Left), at(T, Left), at(B, Right)\}$
- $f\,(I, Drive(Left, Right)) = \{at(A, Left), at(T, Right), at(B, Right)\}$
- $f\,\left(I, Drive(Left, Right)^{+}\right) = \{at(A, Left), at(T, Left), at(T, Right), at(B, Right)\}$
- $f\,(I, \langle Drive(Left, Right), Load(A, Left)\rangle)$ is undefined
- $f\,\left(I, \langle Drive(Left, Right)^{+}, Load(A, Left)^{+}\rangle\right) =$
  $\{at(A, Left), at(T, Left), at(T, Right), at(B, Right), in(A, T)\}$

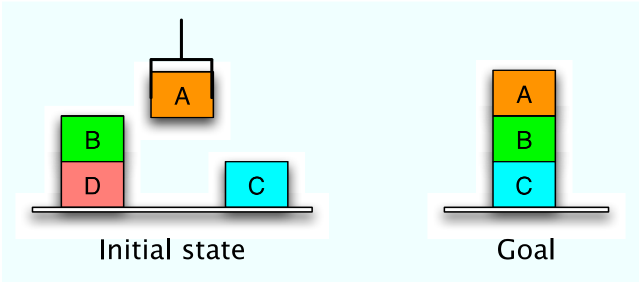
Example: Logistics



- Optimal plan:
  1. *load*$(A, T, Left)$,
  2. *drive*$(Left, Right)$,
  3. *unload*$(A, T, Right)$,
  4. *load*$(B, T, Right)$,
  5. *drive*$(Right, Left)$,
  6. *unload*$(B, T, Left)\}$
- Optimal relaxed plan: ??? (subsequence of the optimal plan)
- $h^*(I) = 6$, $h^+(I) = ???$

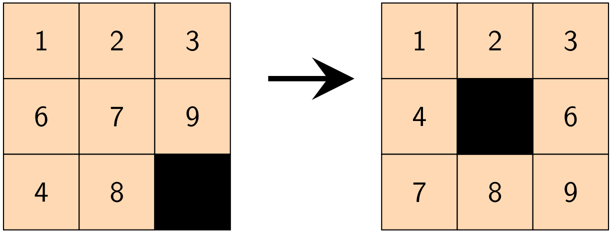
Always subsequence? (Just curious)

An optimal relaxed plan can *not* always be obtained by skipping actions from the (real) optimal plan.



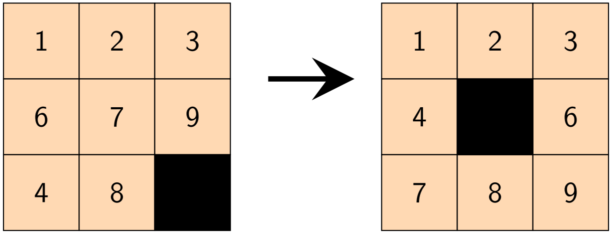
- Optimal plan:
  $\langle putdown(A), unstack(B, D), stack(B, C), pickup(A), stack(A, B)\rangle$
- Optimal relaxed subsequence: ???
- Optimal relaxed plan: ???

Example: 8-Puzzle



- Real problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank
- Monotonically relaxed problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank (!!!)
  - In effect ...

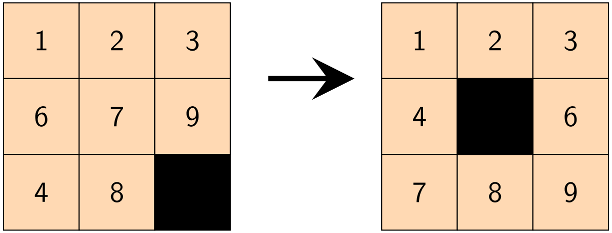
Example: 8-Puzzle



- A tile can move from square A to square B if A is adjacent to B and B is blank - solution distance h^*
- A tile can move from square A to square B if A is adjacent to B - manhattan distance heuristic h^{MD}
- A tile can move from square A to square B if A is adjacent to B and B is blank; in effect, the tile is at both A and B, and both A and B are blank - h^+

Here: $h^*(s_0) = 8$, $h^{MD}(s_0) = 6$, $h^+(s_0) =???$

Example: 8-Puzzle



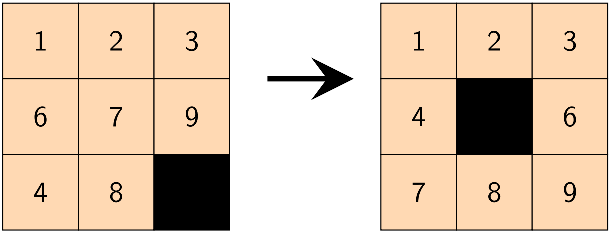
Optimal MD plan:

1. $move(t_9, p_6, p_9)$
2. $move(t_7, p_5, p_8)$
3. $move(t_6, p_4, p_5)$
4. $move(t_6, p_5, p_6)$
5. $move(t_4, p_7, p_4)$
6. $move(t_7, p_8, p_7)$

Optimal relaxed plan:

1. $move(t_9, p_6, p_9)$
2. $move(t_8, p_8, p_9)$
3. $move(t_7, p_5, p_8)$
4. $move(t_6, p_4, p_5)$
5. $move(t_6, p_5, p_6)$
6. $move(t_4, p_7, p_4)$
7. $move(t_7, p_8, p_7)$

Example: 8-Puzzle



Optimal MD plan:

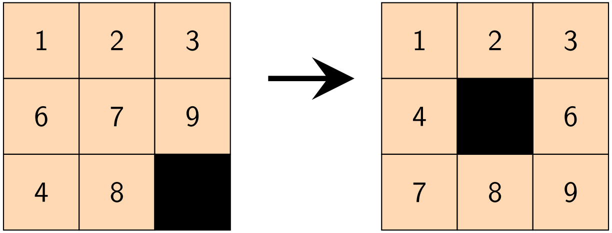
1. $move(t_9, p_6, p_9)$
2. $move(t_7, p_5, p_8)$
3. $move(t_6, p_4, p_5)$
4. $move(t_6, p_5, p_6)$
5. $move(t_4, p_7, p_4)$
6. $move(t_7, p_8, p_7)$

Optimal relaxed plan:

1. $move(t_9, p_6, p_9)$
2. $move(t_8, p_8, p_9)$
3. $move(t_7, p_5, p_8)$
4. $move(t_6, p_4, p_5)$
5. $move(t_6, p_5, p_6)$
6. $move(t_4, p_7, p_4)$
7. $move(t_7, p_8, p_7)$

So $h^*(s_0) = 8,\ h^{MD}(s_0) = 6,\ h^+(s_0) = 7 (> h^{MD}!)$

8-Puzzle: h^+ vs. h^{MD}



h^+ dominates h^{MD}

- The goal is given as a conjunction of $at(t_i, p_j)$ atoms
- Achieving each single one of them takes at least as many steps as the respective tile's Manhattan distance
- Each action moves a single tile only

And we have just seen that h^+ strictly dominates h^{MD}

Dominating states

The on-set $on(s)$ of a state s is the set of atoms that are true in s. A state s' dominates another state s iff $on(s) \subseteq on(s')$.

Lemma (relaxation)

Let s be a state, let s' be a state that dominates s,
and let π be an action sequence which is applicable in s.
Then π^+ is applicable in s' and $app_{\pi^+}(s')$ dominates $app_{\pi}(s)$.
Moreover, if π leads to a goal state from s, then π^+ leads to a goal state from s'.

Proof.

The “moreover” part is immediate from $app_{\pi^+}(s')$ dominating $app_{\pi}(s)$.
Prove the rest by induction over the length of π.

Consequences of the relaxation lemma

Corollary (relaxation leads to dominance and preserves plans)

Let π be an action sequence which is applicable in state s.
Then π^+ is applicable in s and $app_{\pi^+}(s)$ dominates $app_\pi(s)$.
If π is a plan for Π, then π^+ is a plan for Π^+.

Proof.

Apply relaxation lemma with $s' = s$. $\square$

- $\leadsto$ Relaxations of plans are relaxed plans.
- $\leadsto$ Relaxations are no harder to solve than the original task.
- $\leadsto$ Optimal relaxed plans are never longer than optimal plans for original tasks.

Consequences of the relaxation lemma (ctd.)

Corollary (relaxation preserves dominance)

Let s be a state, let s' be a state that dominates s,
and let π^+ be a relaxed action sequence applicable in s.
Then π^+ is applicable in s' and $app_{\pi^+}(s')$ dominates $app_{\pi^+}(s)$.

Proof.

Apply relaxation lemma with π^+ for π, noting that $(\pi^+)^+ = \pi^+$. $\square$

- $\rightsquigarrow$ If there is a relaxed plan starting from state s, the same plan can be used starting from a dominating state s'.
- $\rightsquigarrow$ Making a transition to a dominating state never hurts in relaxed planning tasks.

Monotonicity of relaxed planning tasks

We need one final property before we can provide an algorithm for solving relaxed planning tasks.

Lemma (monotonicity)

Let $a^+ = \langle \mathrm{pre}(a), \mathrm{add}(a), \emptyset \rangle$ *be a relaxed action and let s be a state in which* a^+ *is applicable.*
Then $app_{a^+}(s)$ *dominates* s*.*

Proof.

Since relaxed actions only have positive effects, we have
$on(s) \subseteq on(s) \cup \mathrm{add}(a) = on(app_{o^+}(s))$. □

⤳ Together with our previous results, this means that making a transition in a relaxed planning task never hurts.

Greedy algorithm for relaxed planning tasks

The relaxation and monotonicity lemmas suggest the following algorithm for solving relaxed planning tasks:

Greedy planning algorithm for $\langle P, A^+, I, G \rangle$

```
$s := I$
$\pi^+ := \epsilon$
**forever**:
    **if** $G \subseteq s$:
        **return** $\pi^+$
    **else if** there is an action $a^+ \in A^+$ applicable in $s$
            with $app_{a^+}(s) \neq s$:
        Append such an action $a^+$ to $\pi^+$.
        $s := app_{a^+}(s)$
    **else**:
        **return** unsolvable
```

Correctness of the greedy algorithm

The algorithm is sound:

- If it returns a plan, this is indeed a correct solution.
- If it returns "unsolvable", the task is indeed unsolvable
  - Upon termination, there clearly is no relaxed plan from s.
  - By iterated application of the monotonicity lemma, s dominates I.
  - By the relaxation lemma, there is no solution from I.

What about completeness (termination) and runtime?

- Each iteration of the loop adds at least one atom to $on(s)$.
- This guarantees termination after at most $|P|$ iterations.
- Thus, the algorithm can clearly be implemented to run in polynomial time.
  - A good implementation runs in $O(\|\Pi\|)$.

Using the greedy algorithm as a heuristic

We can apply the greedy algorithm within heuristic search:

- In a search node σ, solve the relaxation of the planning task with *state*(σ) as the initial state.
- Set $h(\sigma)$ to the length of the generated relaxed plan.

Is this an admissible heuristic?

- Yes, IF the relaxed plans are optimal (due to the plan preservation corollary).
- However, usually they are not, because our greedy planning algorithm is very poor.

⊛ What about safety? Goal-awareness? Consistency?

The set cover problem

To obtain an admissible heuristic, we need to generate optimal relaxed plans. Can we do this efficiently?

This question is related to the following problem:

Problem (set cover)

Given: *a finite set U, a collection of subsets $C = \{C_1, \ldots, C_n\}$ with $C_i \subseteq U$ for all $i \in \{1, \ldots, n\}$, and a natural number K.*

Question: *Does there exist a set cover of size at most K, i. e., a subcollection $S = \{S_1, \ldots, S_m\} \subseteq C$ with $S_1 \cup \cdots \cup S_m = U$ and $m \leq K$?*

The following is a classical result from complexity theory:

Theorem

The set cover problem is NP-complete.

Hardness of optimal relaxed planning

Theorem (optimal relaxed planning is hard)

The problem of deciding whether a given relaxed planning task has a plan of length at most K is NP-complete.

Proof.

For membership in NP, guess a plan and verify. It is sufficient to check plans of length at most $|P|$, so this can be done in nondeterministic polynomial time.

For hardness, we reduce from the set cover problem.

Hardness of optimal relaxed planning (ctd.)

Proof (ctd.)

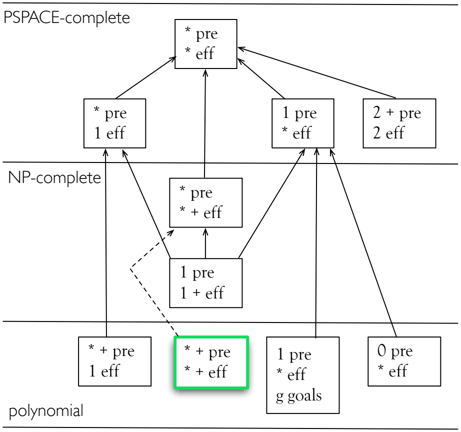
Given a set cover instance $\langle U, C, K \rangle$, we generate the following relaxed planning task $\Pi^+ = \langle P, I, A^+, G \rangle$:

- $P = U$
- $I = \emptyset$ $\equiv I = \{p = 0 \mid p \in P\}$
- $A^+ = \{\langle \emptyset, \bigcup_{p \in C_i} \{p\}, \emptyset \rangle \mid C_i \in C\}$
- $G = U$

If S is a set cover, the corresponding actions form a plan. Conversely, each plan induces a set cover by taking the subsets corresponding to the actions. Clearly, there exists a plan of length at most K iff there exists a set cover of size K.

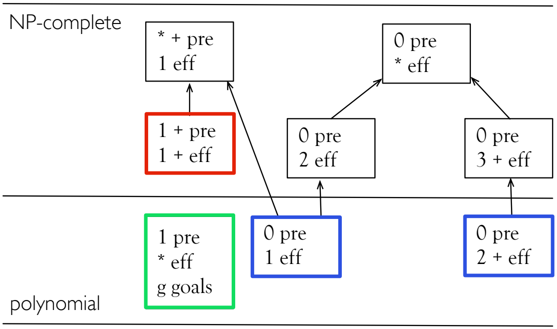
Moreover, Π^+ can be generated from the set cover instance in polynomial time, so this is a polynomial reduction. □

Bylander's Map of BDR: *PlanExt*



Bylander's Map of BDR: *PlanMin*

PlanMin



⊛ The islands are getting smaller and rarer …

Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an optimal planner for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.
  $\leadsto$ h^+ heuristic *(not that realistic. why?)*
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.
  $\leadsto$ h_{max} heuristic, h_{add} heuristic
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but "reasonable".
  $\leadsto$ h_{FF} heuristic

Reminder: Greedy algorithm for relaxed planning tasks

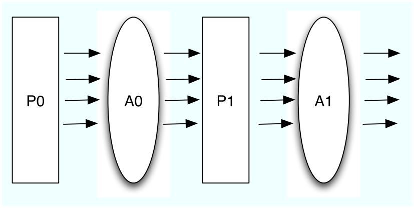
Greedy planning algorithm for $\langle P, A^+, I, G\rangle$

```
$s := I$
$\pi^+ := \epsilon$
**forever**:
    **if** $G \subseteq s$:
        **return** $\pi^+$
    **else if** there is an action $a^+ \in A^+$ applicable in $s$
            with $app_{a^+}(s) \neq s$:
        Append such an action $a^+$ to $\pi^+$.
        $s := app_{a^+}(s)$
    **else**:
        **return** unsolvable
```

Graphical "interpretation": Relaxed planning graphs

- Build a layered reachability graph $P_0, A_0, P_1, A_1, \ldots$

$$\begin{aligned} P_0 &= \{p \in I\} \\ A_i &= \{a \in A \mid \mathrm{pre}(a) \subseteq P_i\} \\ P_{i+1} &= P_i \cup \{p \in \mathrm{add}(a) \mid a \in A_i\} \end{aligned}$$

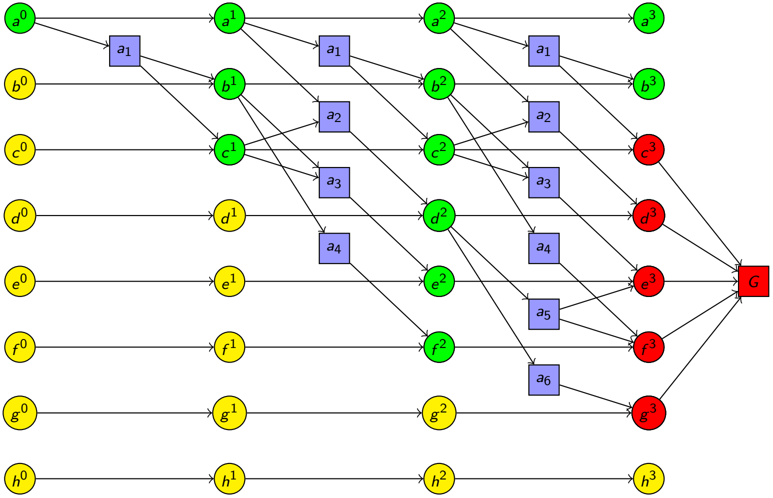


- Terminate when $G \subseteq P_i$

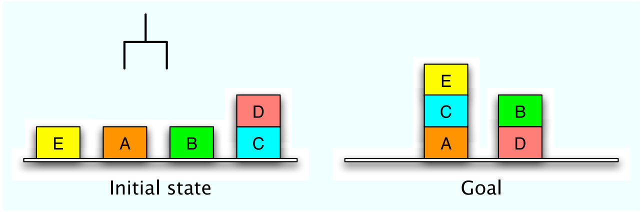
Running example

$$\begin{aligned}
I &= \{a = 1, b = 0, c = 0, d = 0, e = 0, f = 0, g = 0, h = 0\} \\
a_1 &= \langle \{a\}, \{b, c\}, \emptyset \rangle \\
a_2 &= \langle \{a, c\}, \{d\}, \emptyset \rangle \\
a_3 &= \langle \{b, c\}, \{e\}, \emptyset \rangle \\
a_4 &= \langle \{b\}, \{f\}, \emptyset \rangle \\
a_5 &= \langle \{d\}, \{e, f\}, \emptyset \rangle \\
a_6 &= \langle \{d\}, \{g\}, \emptyset \rangle
\end{aligned}$$

Running example: Relaxed planning graph


a^0 b^0 c^0 d^0 e^0 f^0 g^0 h^0
a_1
a^1 b^1 c^1 d^1 e^1 f^1 g^1 h^1
a_1 a_2 a_3 a_4
a^2 b^2 c^2 d^2 e^2 f^2 g^2 h^2
a_1 a_2 a_3 a_4 a_5 a_6
a^3 b^3 c^3 d^3 e^3 f^3 g^3 h^3
G

Example: Blocksworld



1. $\{on(E, Table), clear(E), on(A, Table), clear(A), on(B, Table), clear(B), on(C, Table), on(D, C), clear(D), holding(NIL)\}$
2. $\{\ldots, holding(E), holding(A), holding(B), holding(D), clear(C)\}$
3. $\{\ldots, holding(C), on(E, A), on(A, E), \ldots\}$
4. $\{\ldots, on(C, A), \ldots\}$

Home: Relaxed planning graph for this example

Generic relaxed planning graph heuristics

Computing heuristics from relaxed planning graphs

```
def generic-rpg-heuristic($\langle P, I, O, G\rangle, s$):
    $\Pi^+ := \langle P, s, O^+, G\rangle$
    for $k \in \{0, 1, 2, \ldots\}$:
        $rpg := RPG_k(\Pi^+)$
        if $G \subseteq P_k$:
            Annotate nodes of rpg.
            if termination criterion is true:
                return heuristic value from annotations
        else if $k = |P|$:
            return $\infty$
```

$\rightsquigarrow$ generic template for heuristic functions

$\rightsquigarrow$ to get concrete heuristic: fill in highlighted parts

Concrete examples for the generic heuristic

Many planning heuristics fit the generic template:

- max heuristic h_{max}
- additive heuristic h_{add}
- FF heuristic h_{FF}
- ...

Remarks:

- For all these heuristics, equivalent definitions that don't refer to relaxed planning graphs are possible.
- For some of these heuristics, the most efficient implementations do not use relaxed planning graphs explicitly.

Forward cost heuristics

- The simplest relaxed planning graph heuristics are forward cost heuristics.
- Examples: h_{max}, h_{add}
- Here, node annotations are cost values (natural numbers).
- The cost of a node estimates how expensive (in terms of required operators) it is to make this node true.

Forward cost heuristics: fitting the template

Forward cost heuristics

Computing annotations:

- Propagate cost values bottom-up using a combination rule for action nodes and a combination rule for proposition nodes.
- At action nodes, add 1 after applying combination rule.

Termination criterion:

- stability: terminate if $P_k = P_{k-1}$ and cost for each proposition node $p^k \in P_k$ equals cost for $p^{k-1} \in P_{k-1}$

Heuristic value:

- The heuristic value is the cost of the auxiliary goal node.

- Different forward cost heuristics only differ in their choice of combination rules.

The max heuristic $h_{\max}$ (again)

Forward cost heuristics: max heuristic $h_{\max}$

Combination rule for action nodes:

- $cost(u) = \max(\{cost(v_1), \ldots, cost(v_k)\})$
  (with $\max(\emptyset) := 0$)

Combination rule for proposition nodes:

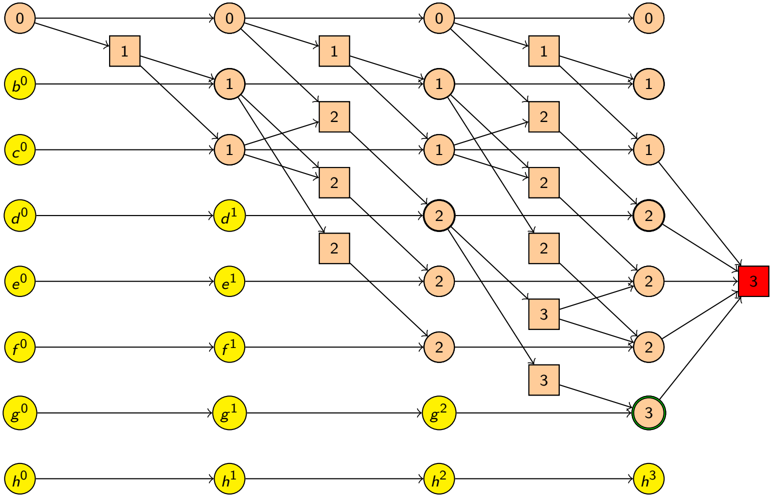
- $cost(u) = \min(\{cost(v_1), \ldots, cost(v_k)\})$

In both cases, $\{v_1, \ldots, v_k\}$ is the set of immediate predecessors of u.

Intuition:

- Action rule: If we have to achieve several preconditions, estimate this by the most expensive cost.
- Proposition rule: If we have a choice how to achieve a proposition, pick the cheapest possibility.

Running example: $h_{\max}$



The additive heuristic

Forward cost heuristics: additive heuristic h_{add}

Combination rule for action nodes:

- $cost(u) = cost(v_1) + \ldots + cost(v_k)$
  (with $\sum(\emptyset) := 0$)

Combination rule for proposition nodes:

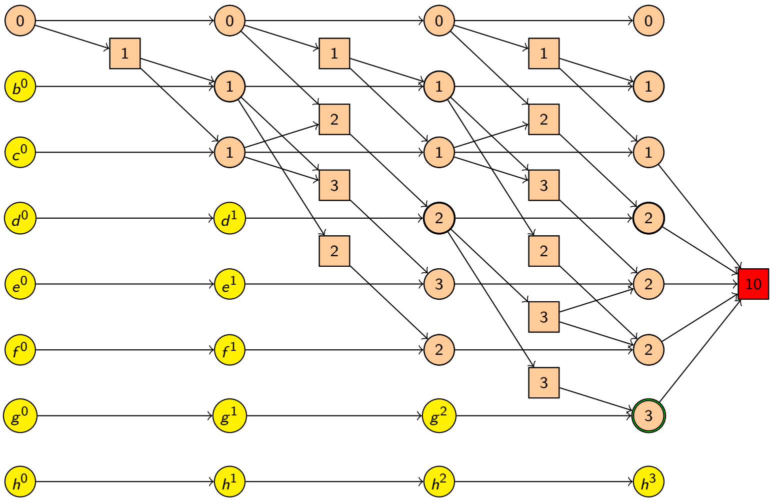
- $cost(u) = \min(\{cost(v_1), \ldots, cost(v_k)\})$

In both cases, $\{v_1, \ldots, v_k\}$ is the set of immediate predecessors of u.

Intuition:

- Action rule: If we have to achieve several preconditions, estimate this by the cost of achieving each in isolation.
- Proposition rule: If we have a choice how to achieve a proposition, pick the cheapest possibility.

Running example: h_{add}



Remarks on h_{add}

- h_{add} is safe and goal-aware.
- Unlike $h_{\max}$, h_{add} is a very informative heuristic in many planning domains.
  Q: Intuitively, when it will be informative?
- The price for this is that it is not admissible (and hence also not consistent), so not suitable for optimal planning.
- In fact, it almost always overestimates the h^+ value because it does not take positive interactions into account.

FF heuristic: fitting the template

The FF heuristic h_{FF}

Computing annotations:

- Annotations are Boolean values, computed top-down.

  A node is marked when its annotation is set to 1 and unmarked if it is set to 0. Initially, the goal node is marked, and all other nodes are unmarked.

  We say that an action node is justified if all its true immediate predecessors are marked, and that a proposition node is justified if at least one of its immediate predecessors is marked.

  . . .

FF heuristic: fitting the template (ctd.)

The FF heuristic h_{FF} (ctd.)

Computing annotations:

- . . .

  Apply these rules until all marked nodes are justified:

  1. Mark all immediate predecessors of a marked unjustified ACTION node.
  2. Mark the immediate predecessor of a marked unjustified PROP node with only one immediate predecessor.
  3. Mark an immediate predecessor of a marked unjustified PROP node connected via an idle arc.
  4. Mark any immediate predecessor of a marked unjustified PROP node.

  The rules are given in priority order: earlier rules are preferred if applicable.

FF heuristic: fitting the template (ctd.)

The FF heuristic h_{FF} (ctd.)

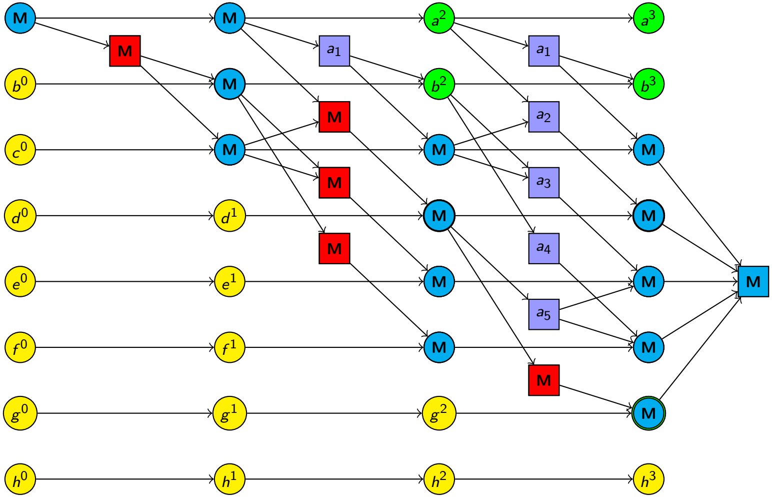
Termination criterion:

- Always terminate at first layer where goal node is true.

Heuristic value:

- The heuristic value is the number of marked action nodes.

Running example: h_{FF}



Remarks on h_{FF}

- Like h_{add}, h_{FF} is safe and goal-aware, but neither admissible nor consistent.
- Always more accurate than h_{add} with respect to h^+.
  - Marked actions define a relaxed plan.
- h_{FF} can be computed in linear time.
  - The h_{FF} value depends on tie-breaking when the marking rules allow several possible choices, so h_{FF} is not well-defined without specifying the tie-breaking rule.
  - The best implementations of FF use additional rules of thumb to try to reduce the size of the generated relaxed plan.

Comparison of relaxation heuristics

Relationship between relaxation heuristics

Let s be a state of planning task $\langle P, I, O, G \rangle$. Then:

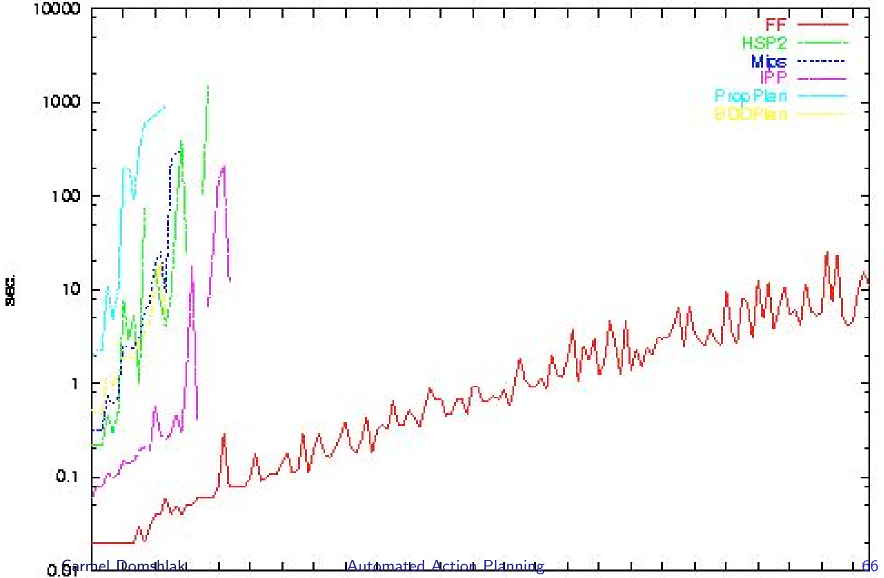
- $h_{\mathrm{max}}(s) \leq h^{+}(s) \leq h^{*}(s)$
- $h_{\mathrm{max}}(s) \leq h^{+}(s) \leq h_{\mathrm{FF}}(s) \leq h_{\mathrm{add}}(s)$
- h^{*} and h_{FF} are pairwise incomparable
- h^{*} and h_{add} are incomparable

Moreover, h^{+}, h_{max}, h_{add}, and h_{FF} assign ∞ to the same set of states.

Note: For inadmissible heuristics, dominance is in general neither desirable nor undesirable. For relaxation heuristics, the objective is usually to get as close to h^{+} as possible.

Does the heuristic really matter?

Example: The 2nd Planning Competition; Schedule domain


sec.
FF
HSP2
Mips
IPP
PropPlan
BDDPlan