Heuristics For Planning

Emil Keyder & Blai Bonet

Universitat Pompeu Fabra & Universidad Simón Bolívar

ICAPS 2009

Planning as heuristic search?

Successful and robust

- Top four planners in the “satisficing” track of IPC6
- Several top planners in the “optimal” track
- Many well-performing planners from previous competitions

Standardized framework

- Mix and match heuristics and search tecniques
- Take advantage of results in other fields

Search Problems

Given a graph $G = \langle V, E \rangle$, where

- V is a **finite** set of vertices
- E is a set of (directed) edges

a search problem P is defined by:

- An **initial vertex** $v_0 \in V$
- **goal verteces** $V_G \subseteq V$
- A function $c : E \rightarrow \mathbb{R}$, giving the cost of each edge

Search Problems

A **solution** is a sequence of edges $P = \langle e_0, \ldots, e_n \rangle$, mapping v_0 into $v_{n+1} \in G$

An **optimal** solution is a path with **minimum** total cost, where the cost of a path is given by the sum of the costs of the edges it contains:

$$c(P) = \sum_{e \in P} c(e)$$

Solving Search Problems

Brute–force approach: Systematically explore full graph

Uniform–cost search, Dijkstra

- Starting from v_0, explore reachable verteces until $v_g \in G$ is found

Complexity of search proportional to $|V|$

Heuristics help by *delaying* or *ruling out* the exploration of unpromising regions of the graph

Heuristics: What are they?

In the graph setting, heuristics are methods for estimating the distance from a node to some goal node

Definition

$h^*(s)$ is the cost of the lowest-cost path from s to a goal node

$h^*(s) \longrightarrow$ optimal solution in linear time

Heuristics: What are they?

In the graph setting, heuristics are methods for estimating the distance from a node to some goal node

Definition

$h^*(s)$ is the cost of the lowest-cost path from s to a goal node

$h^*(s) \rightarrow$ optimal solution in linear time

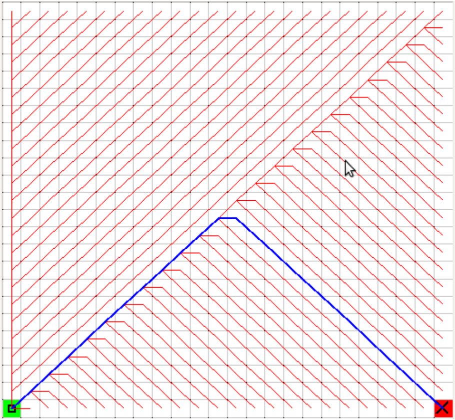
Objective when designing a heuristic is to get as close as possible to h^*

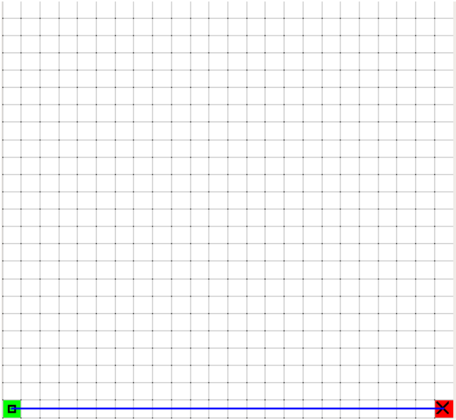
The power of heuristics

Consider the two following heuristics for grid navigation problems:

- $h(s) = 0, \forall s$, equivalent to blind search
- $h(s) = |x_G - x_s| + |y_G - y_s|$, the Manhattan Distance heuristic

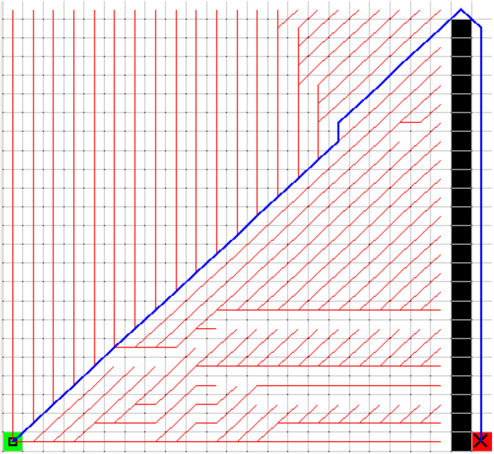
Blind vs. Informed Search





Manhattan Distance heuristic $= h^*$

Blind vs. Informed Search



When obstacles are present, h_{MD} is uninformed and explores large part of state space

Heuristics: Some properties

Definition

A heuristic h is *admissible* if for all $s \in S$: $h(s) \leq h^*(s)$

Heuristics: Some properties

Definition

A heuristic h is *admissible* if for all $s \in S$: $h(s) \leq h^*(s)$

Definition

A heuristic h is *consistent* if for all $s \in S$ and edge (s, s'):

$$h(s) \leq c(s, s') + h(s')$$

Heuristics: Some properties

Definition

A heuristic h is *admissible* if for all $s \in S$: $h(s) \leq h^*(s)$

Definition

A heuristic h is *consistent* if for all $s \in S$ and edge (s, s'):

$$h(s) \leq c(s, s') + h(s')$$

- Consistency implies admissibility
- Admissible heuristics used to compute optimal solutions
- Consistent heuristics guarantee optimal behaviour
- $h(s) = 0$ is a consistent but **non-informative** heuristic

Classical planning with costs as a search problem

- Classical planning problems are search problems in the state-space graph where
  - nodes are planning states
  - edges correspond to operators
- Paths from s_0 to a goal state are **valid plans**
- An **optimal plan** is a plan of minimum cost

Classical planning as a search problem

- Set of states S
- Initial state $s_0 \in S$
- A function $G(s)$ that tells us whether a state is a goal
- Planning operators O
- Applicable operators $A(s) \subseteq O$ in state s
- Transition function $f(s, o)$ for $s \in S$ and $o \in A(s)$
- Non-negative operator costs, $c(o) \in \mathbb{R}_0^+$

Planning as heuristic search

Idea: Search the state space using a heuristic

The “better” the heuristic:

- the fewer the generated (stored) states
- the faster a solution is found

For optimal planning, use admissible heuristics

Problem: How to compute good heuristics?

Planning as heuristic search

Idea: Search the state space using a heuristic

The “better” the heuristic:

- the fewer the generated (stored) states
- the faster a solution is found

For optimal planning, use admissible heuristics

Problem: How to compute good heuristics?

This is the topic of this tutorial

Outline for the rest of the tutorial

- Planning Problems
- Part I: Delete-Relaxation Heuristics
- Part II: Exact Computation of h^+
- Part III: The h^m Heuristics
- Part IV: The Context-Enhanced Additive Heuristic

Problem representation

Factored representations of states:

- Set V of variables
- Domain D_X of values for each variable X

Most common: STRIPS (boolean variables)

- $D_{\textit{truck-at-Thessaloniki}} = \{\textit{true}, \textit{false}\}$

Another option: SAS$^+$ (multi-valued variables)

- $D_{\textit{loc-truck}} = \{\textit{Thessaloniki}, \textit{Athens}, \textit{Istanbul}\}$

STRIPS

- Problems $P = \langle F, I, G, O, c \rangle$:
  - fluents (boolean variables) F
  - initial state $I \subseteq F$
  - goal description $G \subseteq F$
  - operators $O = \langle Pre(o), Add(o), Del(o) \rangle$, each $\subseteq F$
  - positive costs $c(o)$
- States are subsets of fluents that have value *true*
- State space is exponential $O(2^{|F|})$

SAS$^+$

- Problems $P = \langle V, I, G, O, c \rangle$:
  - set of variables V, each with associated domain D_V
  - initial state I, a *full* variable assignment
  - goal G, a *partial* variable assignment
  - operators O, consisting of two *partial* variable assignments $\langle Pre(o), Eff(o) \rangle$
  - positive costs $c(o)$
- A *full* variable assignment assigns to each $v_i \in V$ a value $d_i \in D_{v_i}$ and represents a state
- A *partial* variable assignment assigns values to a subset $C \subseteq V$
- State space: $\prod_{v \in V} |D_v|$

Translation from STRIPS to SAS$^+$

Basic Idea: Make a graph with node set F, and edges between any two fluents that cannot occur in the same state

- Example:
  $\langle$*truck-at-Thessaloniki*, *truck-at-Athens*, *truck-at-Istanbul*$\rangle$

Translation from STRIPS to SAS$^+$

Basic Idea: Make a graph with node set F, and edges between any two fluents that cannot occur in the same state

- Example:
  ⟨*truck-at-Thessaloniki*, *truck-at-Athens*, *truck-at-Istanbul*⟩

Cliques in graph represent multi–valued variables

Replace n boolean variables with single n–valued variable

- More efficient representation - 2^n vs. n configurations

Part I

Delete-Relaxation Heuristics

Relaxations

Relaxations are assumptions about a problem that make it easier to solve

- Manhattan Distance assumption: no barriers

Idea: Solve *relaxed* problem and use cost of this solution as h

The Delete Relaxation P^+

Assumption: Once a variable has a certain value, that value can always be used to satisfy preconditions/goals

- When something is achieved, it stays achieved

$\Rightarrow$
move-right

$h^+ = h^*(P^+)$ is **admissible**

- Any solution to P is a solution to P^+ as well
- In P, a fluent may have to be made true > 1 times

The Delete Relaxation P^+ – Formalization

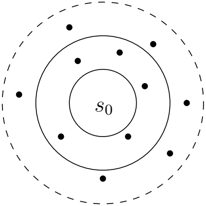
In STRIPS, remove delete lists:

Definition

Given a STRIPS problem $P = \langle F, I, G, O, c\rangle$, its delete relaxation is given by $P^+ = \langle F, I, G, O', c\rangle$, where

$$O' = \{\langle Pre(o), Add(o), \emptyset\rangle \mid o \in O\}$$

- $|s|$ increases with each action
- Stratification of fluents by first *level* at which they can be achieved



The Delete Relaxation P^+

Plan existence for P^+ is easy

- Use all applicable operators until goal achieved or no further fluents can be added

Optimal solution to P^+ is NP-hard

- Why?

Simplifying P^+

Ideally, $|s_0| = |Pre(o)| = |Add(o)| = |G| = 1$

- Search problem on graph $G = \langle V, E \rangle$, $V = F$, $E = O$
- $|F|$ is small, so easily solvable

Question: Does there exist a $P^{+\prime}$ equivalent to P^+ with this property?

Simplifying P^+

Ideally, $|s_0| = |Pre(o)| = |Add(o)| = |G| = 1$

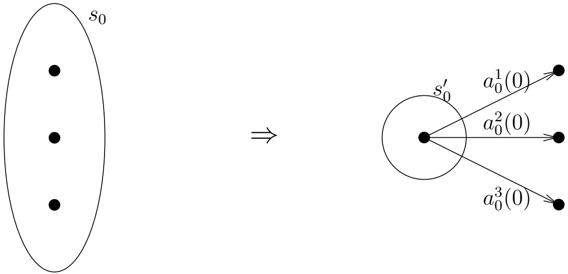
- Search problem on graph $G = \langle V, E \rangle$, $V = F$, $E = O$
- $|F|$ is small, so easily solvable

Question: Does there exist a $P^{+'}$ equivalent to P^+ with this property?

Theorem

For any P^+, there exists an equivalent problem $P^{+'}$ such that $|s'_0| = 1$, $|G'| = 1$, and $|Add(o)| = 1$ for all $o \in O'$

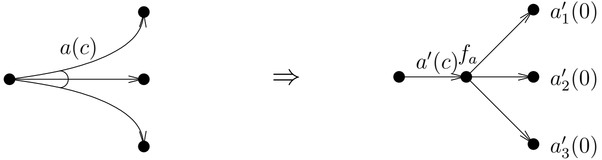
Simplifying P^+



A start state with size $|s_0| = n$ can be replaced with:

- A new start state containing a single newly-introduced fluent: $s'_0 = \{f\}$
- A set of n zero-cost actions: $A_0 = \{\langle\{f\}, \{s_i\}, \emptyset\rangle \mid s_i \in s_0\}$

Simplifying P^+

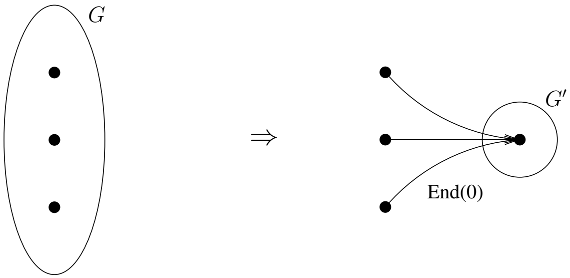


An action a with $|Add(a)| = n$ can be replaced with:

- A new fluent f_a, representing that the action has been executed
- An action $a' = \langle Pre(a), f_a, \emptyset \rangle$ with cost $c(a') = c(a)$
- A set of n zero-cost actions: $A' = \{\langle \{f_a\}, \{a_i\}, \emptyset \rangle \mid a_i \in Add(a)\}$

Simplifying P^+

And $|G|$?



A goal with size $|G| = n$ can be replaced with:

- A new goal with a single newly-introduced fluent: $G' = \{g\}$
- A single action $End = \langle G, g, \emptyset \rangle$

P^+ as a graph problem

Special cases of P^+ are well-known graph problems in which nodes $\longrightarrow$ fluents, edges $\longrightarrow$ actions

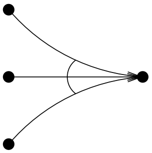
- $|G| = |Pre(o)| = 1 \longrightarrow$ Shortest Path — tractable
- $|G| > 1$, $|Pre(o)| = 1 \longrightarrow$ Directed Steiner Tree — NP-hard
- $|Pre(o)| > 1 \longrightarrow$ Optimal Directed Hyperpath — NP-hard

Hypergraphs vs. AND/OR graphs?

Equivalent representations:

Hypergraphs

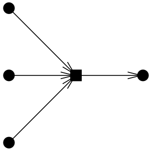
- Multi-source edges



- Weights on edges

AND/OR graphs

- AND nodes



- Weights on AND nodes

The Delete Relaxation P^+

Difficulty of P^+: estimating cost of *sets*

To be optimal, must take into account interactions between plans for fluents in set

Interactions between plans for individual fluents are always *positive*:

Theorem

For any set of fluents $G \subseteq F$,

$$h^+(G) \leq \sum_{g \in G} h^+(g)$$

The Delete Relaxation P^+

Idea: Solve P^+ suboptimally, use cost of this solution as h

- Resulting heuristics no longer admissible
- But useful for finding **suboptimal** solutions fast
- **Fundamental tradeoff:** Computation time vs. solution quality

The (Numeric) Additive Heuristic h_{add}

First practical domain-independent planning heuristic (used in HSP)

Relies on the independence assumption *for costs*:

$$h_{add}(G) \stackrel{\text{def}}{=} \sum_{g \in G} h_{add}(g)$$

- **Intuition:** Estimate the cost of a set as the sum of the costs of the individual fluents
- Assume **no** positive interactions

The (Numeric) Additive Heuristic h_{add}

$$h_{add}(s) = h_{add}(G; s)$$

$$h_{add}(P; s) \stackrel{\text{def}}{=} \sum_{p \in P} h_{add}(p; s)$$

where

$$h_{add}(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ h_{add}(a_p(s)) & \text{otherwise} \end{cases}$$

$$a_p(s) \stackrel{\text{def}}{=} \text{argmin}_{a \in O(p)} \; h_{add}(a; s)$$

$$h_{add}(a; s) \stackrel{\text{def}}{=} cost(a) + h_{add}(Pre(a); s)$$

Computation

Equations give properties of estimates, not how to calculate them

Basic idea: value iteration

- Start with rough estimates (e.g. $0, \infty$), do updates

Methods differ principally in choice of update ordering:

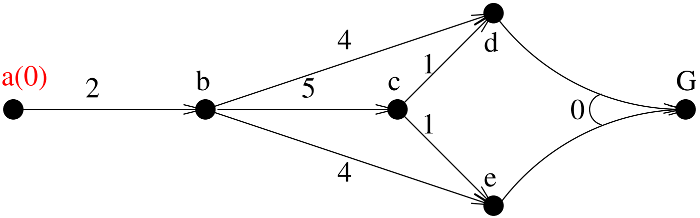
1. Label-correcting methods
   - Arbitrary action choice
   - Multiple updates per fluent
2. Dijkstra method
   - Updates placed in priority queue
   - Single update per fluent

Computation

Algorithm 1 Heuristic calculation

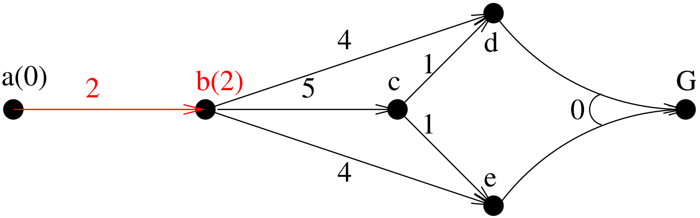
$s \leftarrow$ current state
for $p \in F$ **do**
  if $p \in s$ **then** $h(p) = 0$ **else** $h(p) = \infty$ // Initialization
end for
repeat
  a $=$ chooseAction()
  for $q \in Add(a)$ **do**
    if $h(a; s) < h(q; s)$ **then**
      $h(q; s) = h(a; s)$
    end if
  end for
until fixpoint

Example - Label correcting method



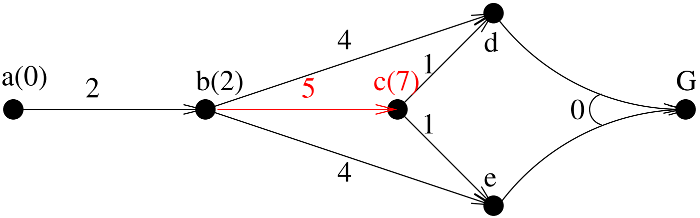
a	b	c	d	e	G
0	∞	∞	∞	∞	∞

Example - Label correcting method



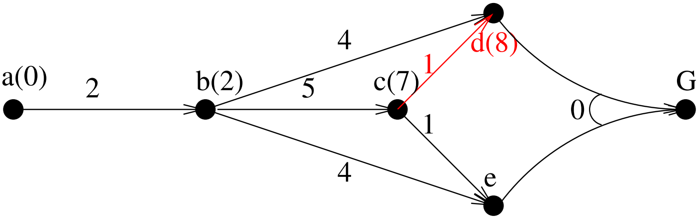
a	b	c	d	e	G
0	2	∞	∞	∞	∞

Example - Label correcting method



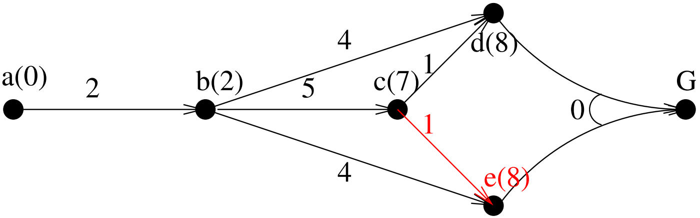
a	b	c	d	e	G
0	2	7	∞	∞	∞

Example - Label correcting method



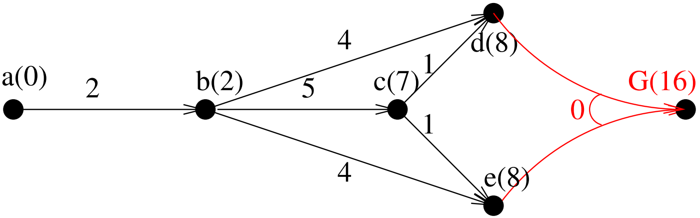
a	b	c	d	e	G
0	2	7	8	∞	∞

Example - Label correcting method



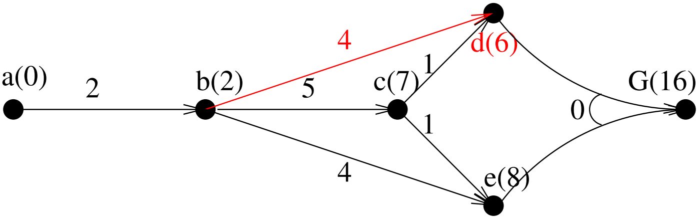
a	b	c	d	e	G
0	2	7	8	8	∞

Example - Label correcting method



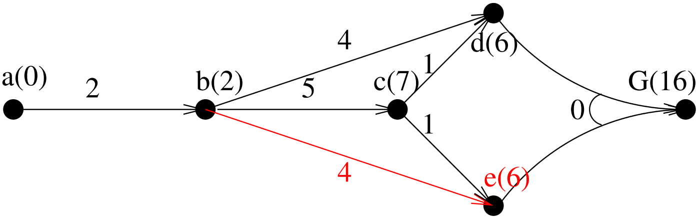
a	b	c	d	e	G
0	2	7	8	8	16

Example - Label correcting method



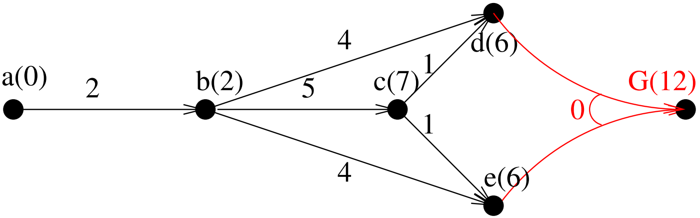
a	b	c	d	e	G
0	2	7	6	8	16

Example - Label correcting method



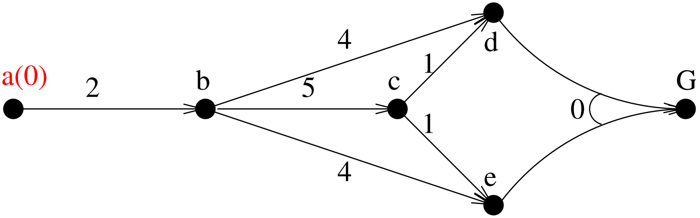
a	b	c	d	e	G
0	2	7	6	6	16

Example - Label correcting method



a	b	c	d	e	G
0	2	7	6	6	12

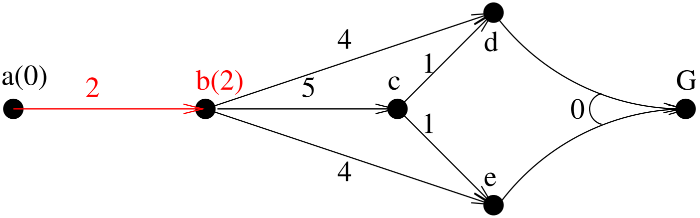
Example - Dijkstra method



a	b	c	d	e	G
0	∞	∞	∞	∞	∞

$\leftarrow$ Priority			
b(2)			

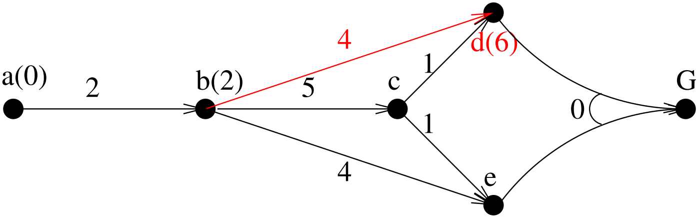
Example - Dijkstra method



a	b	c	d	e	G
0	2	∞	∞	∞	∞

$\leftarrow$ Priority			
d(6)	e(6)	c(7)	

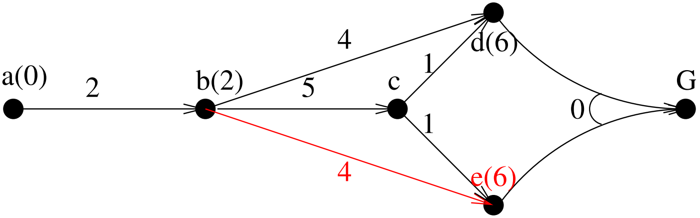
Example - Dijkstra method



a	b	c	d	e	G
0	2	∞	6	∞	∞

← Priority			
e(6)	c(7)	G(∞)	

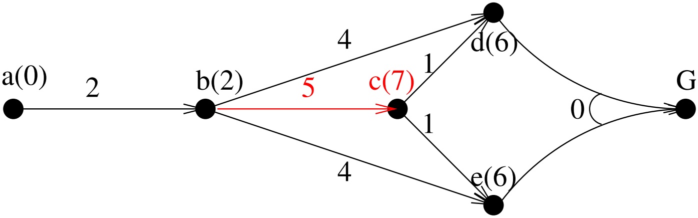
Example - Dijkstra method



a	b	c	d	e	G
0	2	∞	6	6	∞

← Priority			
c(7)	G(12)		

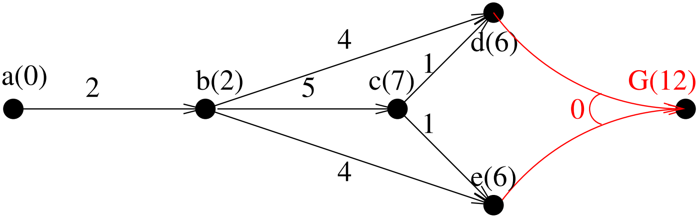
Example - Dijkstra method



a	b	c	d	e	G
0	2	7	6	6	∞

$\leftarrow$ Priority			
G(12)			

Example - Dijkstra method



a	b	c	d	e	G
0	2	7	6	6	12

← Priority			

Comments

Generalized Dijkstra performs single update per fluent

- Cannot overcome overhead from priority queue

In practice, Generalized Bellman-Ford often used

- However, GD + incremental computation shown to give speedup

See *Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning* by *Liu, Koenig and Furcy*

Problems with h_{add}

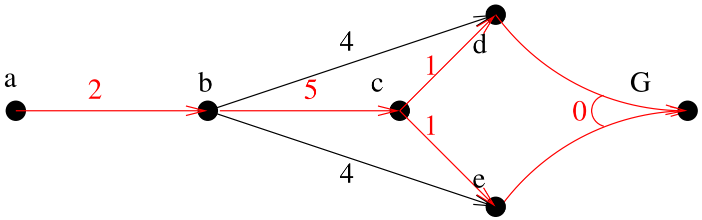


Figure: $h^+(G) = 2 + 5 + 1 + 1 + 0 = 9, h_{add}(G) = 12$

Sources of error in $h_{add}(G)$?

Problems with h_{add}

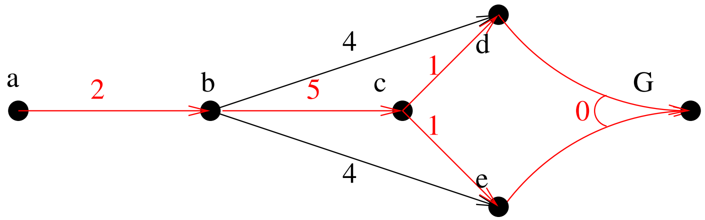


Figure: $h^+(G) = 2 + 5 + 1 + 1 + 0 = 9$, $h_{add}(G) = 12$

Sources of error in $h_{add}(G)$?

1. Overcounting
   - $a \rightarrow b$ counted twice

Problems with h_{add}

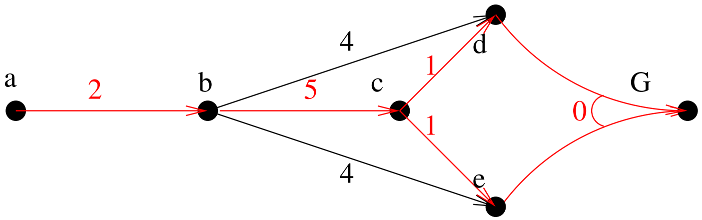


Figure: $h^+(G) = 2 + 5 + 1 + 1 + 0 = 9, h_{add}(G) = 12$

Sources of error in $h_{add}(G)$?

1. Overcounting
   - $a \longrightarrow b$ counted twice
2. Independence assumption
   - $h_{add}(d, e) = h_{add}(d) + h_{add}(e)$ – not optimal

Problems with h_{add}

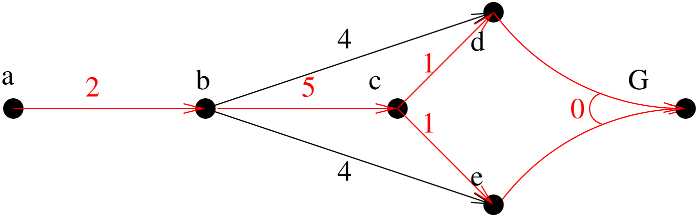


Figure: $h^+(G) = 2 + 5 + 1 + 1 + 0 = 9, h_{add}(G) = 12$

Sources of error in $h_{add}(G)$?

1. Overcounting – Easier to address
   - $a \longrightarrow b$ counted twice
2. Independence assumption
   - $h_{add}(d, e) = h_{add}(d) + h_{add}(e)$ – not optimal

Relaxed Plan Heuristics

Idea: Find explicit plan π^+ for P^+, $h = cost(\pi^+)$

- Incrementally construct plan, make sure no duplicates occur
- No overcounting

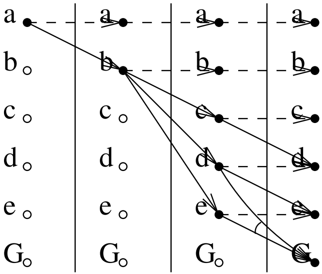
Relies on the idea of *supporters*

- The *supporter* of a fluent p is the designated action used to make p true

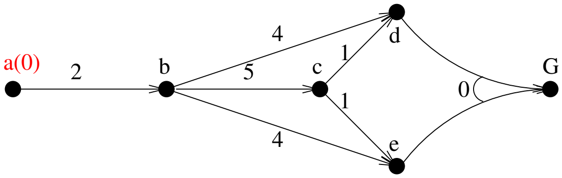
First used in FF with *relaxed planning graphs*

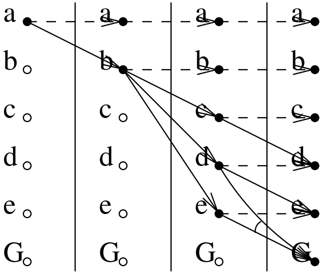
Relaxed Planning Graphs

- Tool to graphically represent heuristic computation
- Layer *i* contains facts achievable with *i* layer parallel plan



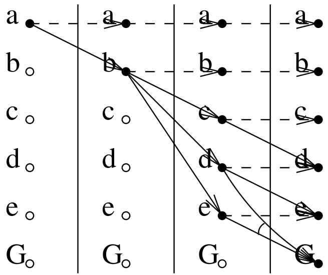
Relaxed Planning Graphs





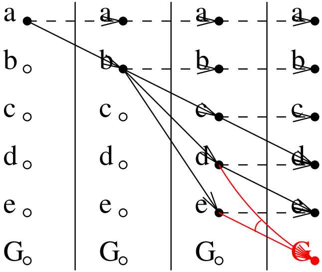
Relaxed Plan Heuristics

Construct relaxed plan by starting from goal, choose supporter for each fluent *at first layer in which it appears*



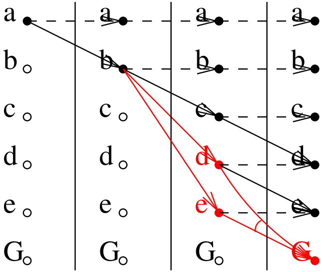
Relaxed Plan Heuristics

Construct relaxed plan by starting from goal, choose supporter for each fluent *at first layer in which it appears*



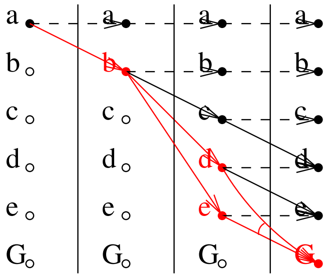
Relaxed Plan Heuristics

Construct relaxed plan by starting from goal, choose supporter for each fluent *at first layer in which it appears*



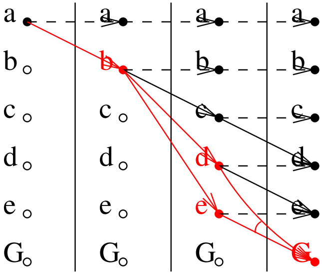
Relaxed Plan Heuristics

Construct relaxed plan by starting from goal, choose supporter for each fluent *at first layer in which it appears*



Relaxed Plan Heuristics

Construct relaxed plan by starting from goal, choose supporter for each fluent *at first layer in which it appears*



$$h_{\mathsf{FF}}(s) = c(\pi) = \{\langle a \rightarrow b\rangle, \langle b \rightarrow d\rangle, \langle b \rightarrow e\rangle, \langle d, e \rightarrow G\rangle\} = 10$$

h_{FF} Relaxed Plan

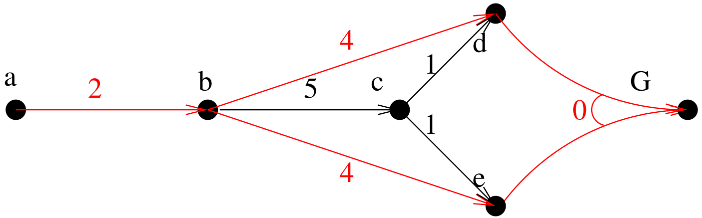


Figure: Plan computed by h_{FF}

Relaxed Plan Heuristics - Benefits

Generate explicit plans π whose cost can be used as the heuristic value, rather than only numeric estimates

Advantages:

- Explicit representation of plan – **no overcounting!**
- *Helpful actions*

Helpful Actions: Suggestions likely to decrease heuristic estimate

- Prune non-helpful actions – generate/evaluate fewer nodes

The h_{max} Heuristic

h_{max} replaces $\sum$ in h_{add} with max:

$$h_{max}(P; s) = \max_{p \in P}(h_{max}(p; s))$$

- Estimates cost of set as cost of most expensive fluent in set

Admissible

Turns out to be an instance of a more general formulation – more on this later

Relaxed Planning Graphs and h_{max}

If action costs are uniform, h_{max} and RPGs are related

If $p \in s$:

- $h_{max}(p) = 0$
- $\mathit{rpg\text{-}level}(p) = 0$

else:

- $h_{max}(p) = \min_{a \in O(p)} (1 + h_{max}(Pre(a)))$
- $\mathit{rpg\text{-}level}(p) = \min_{a \in O(p)} (1 + \mathit{rpg\text{-}level}(Pre(a)))$

Relaxed Planning Graphs and h_{max}

If action costs are uniform, h_{max} and RPGs are related

If $p \in s$:

- $h_{max}(p) = 0$
- $\mathit{rpg\text{-}level}(p) = 0$

else:

- $h_{max}(p) = \min_{a \in O(p)} (1 + h_{max}(Pre(a)))$
- $\mathit{rpg\text{-}level}(p) = \min_{a \in O(p)} (1 + \mathit{rpg\text{-}level}(Pre(a)))$

Theorem

When action costs are uniform, the relaxed planning graph level of a fact is equal to its h_{max} estimate

Relaxed Plans from h_{max}, h_{add}, etc.

h_{FF} uses uniform cost h_{max} supporters + plan extraction algorithm

Drawback: Uniform cost h_{max} not **cost-sensitive**

Solution: Cost-sensitive heuristic to choose best supporter ...

$$a_p(s) = \operatorname*{argmin}_{a \in O(p)} h(a; s)$$

... combined with generic plan extraction algorithm

- Construct π^+ by collecting the best supporters recursively backwards from the goal

Relaxed Plan Extraction Algorithm

Algorithm 2 Relaxed plan extraction

$\pi^+ \leftarrow \emptyset$
$\mathit{supported} \leftarrow s$
$\mathit{to\text{-}support} \leftarrow G$
while $\mathit{to\text{-}support} \neq \emptyset$ **do**
  Remove a fluent p from *to-support*
  if $p \notin \mathit{supported}$ **then**
    $\pi^+ \leftarrow \pi^+ \cup \{a_p(s)\}$
    $\mathit{supported} \leftarrow \mathit{supported} \cup \mathit{Add}(a_p(s))$
    $\mathit{to\text{-}support} \leftarrow \mathit{to\text{-}support} \cup (\mathit{Pre}(a_p(s)) \setminus \mathit{supported})$
  end if
end while

Relaxed Plans

... and estimate the cost of a state s as the cost of $\pi^+(s)$:

$$h(s) = Cost(\pi^+(s)) = \sum_{a \in \pi^+(s)} c(a)$$

Results in **cost-sensitive** heuristic with **no overcounting**

The Set–Additive Heuristic h_a^s

Different method for computing relaxed plans, sometimes with higher quality

Idea: Instead of costs, propagate *the supports themselves*

- For each fluent, maintain explicit relaxed plan
- Obtain plan for set as union of plans for each
- Seeds for computation are also sets:

$$\pi(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The cost of an undefined plan is ∞

The Set–Additive Heuristic h_a^s

$$h_a^s(s) = Cost(\pi(G; s))$$

$$\pi(P; s) = \bigcup_{p \in P} \pi(p; s)$$

where

$$\pi(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi(a_p(s); s) & \text{otherwise} \end{cases}$$

$$\begin{aligned} a_p(s) &= \operatorname*{argmin}_{a \in O(p)} [Cost(\pi(a; s))] \\ \pi(a; s) &= \{a\} \bigcup \{ \cup_{q \in Pre(a)} \pi(q; s)\} \\ Cost(\pi) &= \sum_{a \in \pi} cost(a) \end{aligned}$$

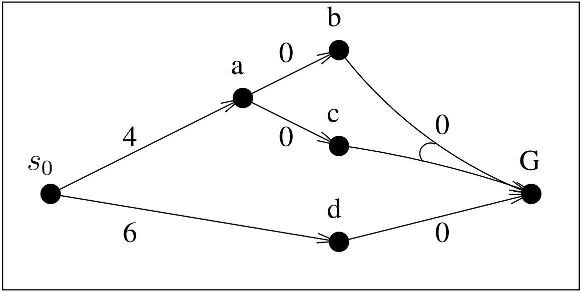
The Set–Additive Heuristic h_a^s

Intuition: Estimate the cost of a set taking into consideration the overlap between the plans for the individual fluents

- Potentially better estimates

Drawback: Requires the expensive $\cup$ operator to calculate the cost of a set of fluents, rather than the cheaper $\sum$ or *max* operators

Example



Example

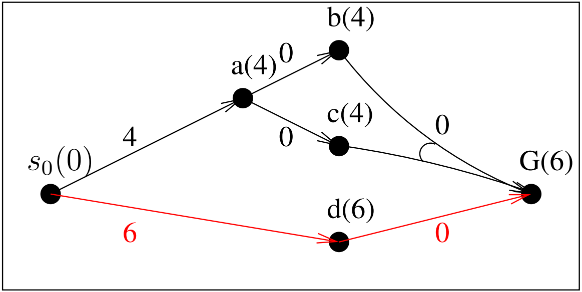
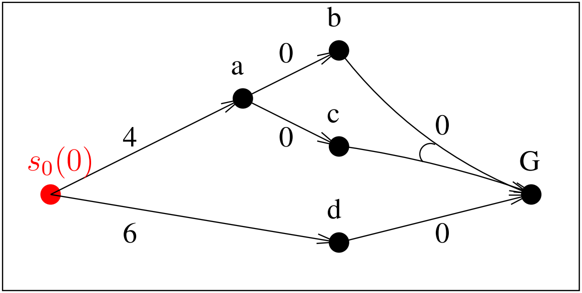


Figure: $\pi_{h_{add}} = \{s_0 \rightarrow d, d \rightarrow G\}, c(\pi_{h_{add}}) = 6$

h_{add} counts cost of action $s_0 \rightarrow a$ twice when computing cost of upper path

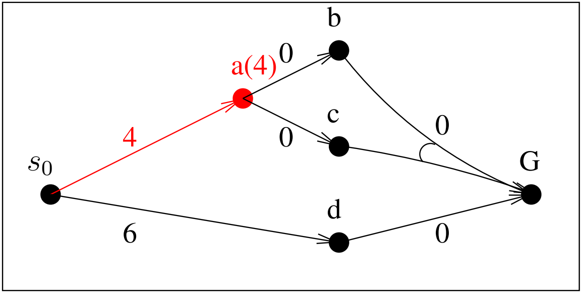
- Leads to suboptimal relaxed plan

Example



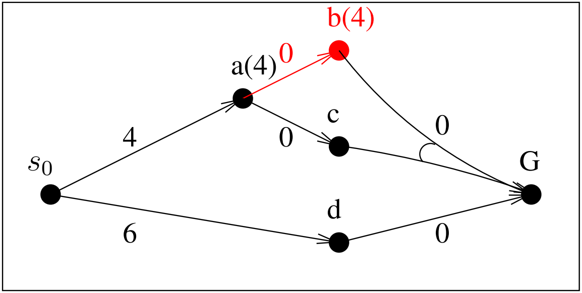
$\pi(s_0)$	$\{\}$
$\pi(a)$	undefined
$\pi(b)$	undefined
$\pi(c)$	undefined
$\pi(d)$	undefined
$\pi(G)$	undefined

Example



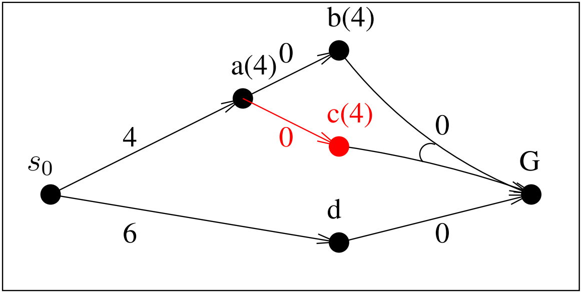
$\pi(s_0)$	$\{\}$
$\pi(a)$	$\{s_0 \rightarrow a\}$
$\pi(b)$	undefined
$\pi(c)$	undefined
$\pi(d)$	undefined
$\pi(G)$	undefined

Example



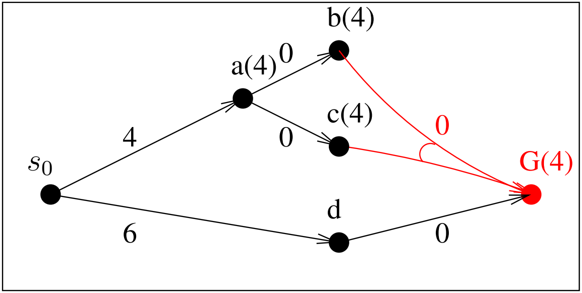
$\pi(s_0)$	$\{\}$
$\pi(a)$	$\{s_0 \longrightarrow a\}$
$\pi(b)$	$\{s_0 \longrightarrow a, a \longrightarrow b\}$
$\pi(c)$	undefined
$\pi(d)$	undefined
$\pi(G)$	undefined

Example



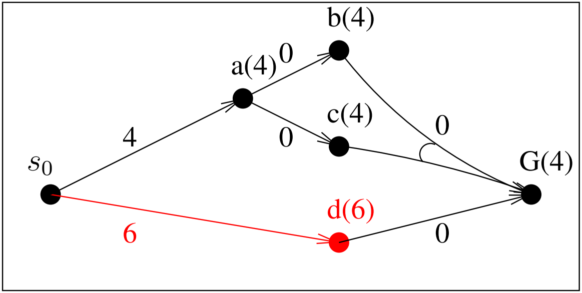
$\pi(s_0)$	$\{\}$
$\pi(a)$	$\{s_0 \rightarrow a\}$
$\pi(b)$	$\{s_0 \rightarrow a, a \rightarrow b\}$
$\pi(c)$	$\{s_0 \rightarrow a, a \rightarrow c\}$
$\pi(d)$	undefined
$\pi(G)$	undefined

Example



$\pi(s_0)$	$\{\}$
$\pi(a)$	$\{s_0 \longrightarrow a\}$
$\pi(b)$	$\{s_0 \longrightarrow a, a \longrightarrow b\}$
$\pi(c)$	$\{s_0 \longrightarrow a, a \longrightarrow c\}$
$\pi(d)$	undefined
$\pi(G)$	$\{s_0 \longrightarrow a, a \longrightarrow b, a \longrightarrow c, bc \longrightarrow G\}$

Example



$\pi(s_0)$	$\{\}$
$\pi(a)$	$\{s_0 \rightarrow a\}$
$\pi(b)$	$\{s_0 \rightarrow a, a \rightarrow b\}$
$\pi(c)$	$\{s_0 \rightarrow a, a \rightarrow c\}$
$\pi(d)$	$\{s_0 \rightarrow d\}$
$\pi(G)$	$\{s_0 \rightarrow a, a \rightarrow b, a \rightarrow c, bc \rightarrow G\}$

Example

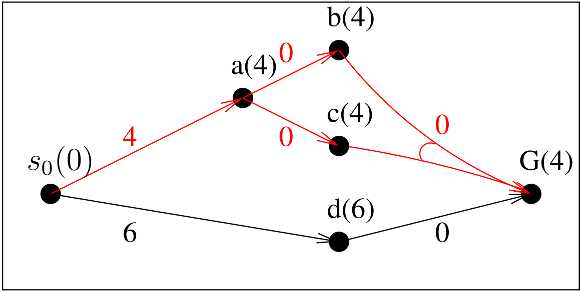


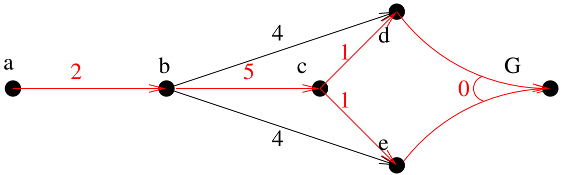
Figure: $\pi_{h_a^s} = \{s_0 \rightarrow a, a \rightarrow b, a \rightarrow c, bc \rightarrow G\}, c(\pi_{h_a^s}) = 4$

h_a^s stores explicit relaxed plans, realizing that the plans for b and c overlap

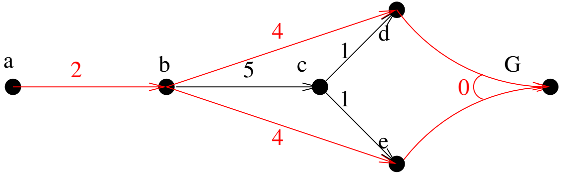
Remaining issues

Relaxed plan heuristics solve the **overcounting** issue by computing an explicit relaxed plan with no duplicate actions

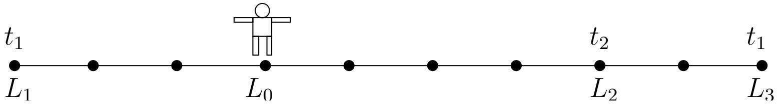
Independence assumption issues remain



vs.

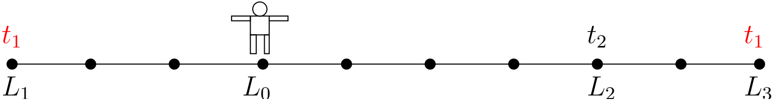


Another Example



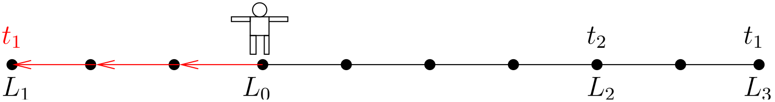
Agent at location L_0 must perform tasks t_1 and t_2

Another Example



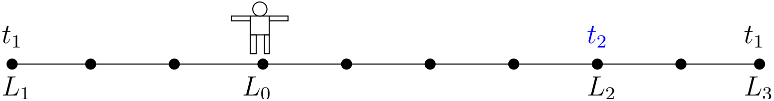
Task t_1 can be performed at L_1 and L_3

Another Example



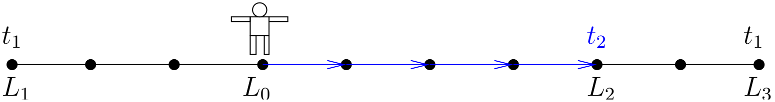
$$\pi^+(t_1) = \langle \textit{left}, \textit{left}, \textit{left}, \textit{do } t_1 \rangle$$

Another Example



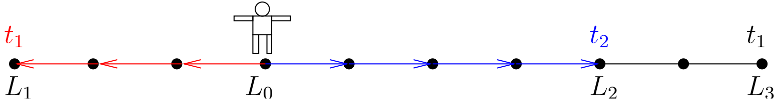
Task t_2 can be performed only at L_2

Another Example



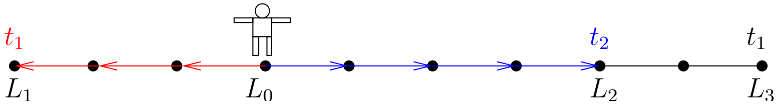
$$\pi^+(t_2) = \langle right, right, right, right, do\ t_2 \rangle$$

Another Example



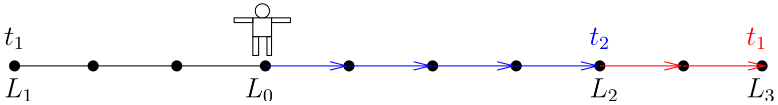
Delete relaxation heuristics such as h_{add}, h_a^s and h_{FF} compute plan for $\{t_1, t_2\}$ by combining plans for t_1 and t_2, giving a cost of 7 – in this case and many others, this is suboptimal

Another Example



Plan suggests actions *left* and *right*, neither of which decreases heuristic estimate

Another Example



Optimal plan for $\{t_1, t_2\}$ in both P and P^+, *right* is only suggested action and decreases heuristic estimate

The Independence Assumption

Heuristics discussed previously implicitly or explicitly assume *independence* to make P^+ tractable:

The Independence Assumption for Relaxed Plan Heuristics

The relaxed plan for a **set of goals** G is the union of the relaxed plans **for each goal**

$$\pi^+(G) = \bigcup_{g \in G} \pi^+(g)$$

The limitations of this approach can be understood by considering the *Steiner Tree Problem*

The Steiner Tree Problem

The Steiner Tree Problem

Given a graph $G = \langle V, E \rangle$ and a set of *terminal nodes* $T \subseteq V$, find a minimum-cost tree S that spans all $t \in T$

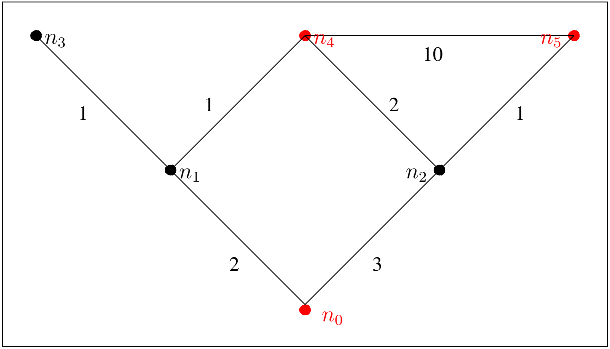


Figure: A Steiner Tree Problem, $T = \{n_0, n_4, n_5\}$

The Steiner Tree Problem

The Steiner Tree Problem

Given a graph $G = \langle V, E \rangle$ and a set of *terminal nodes* $T \subseteq V$, find a minimum-cost tree S that spans all $t \in T$

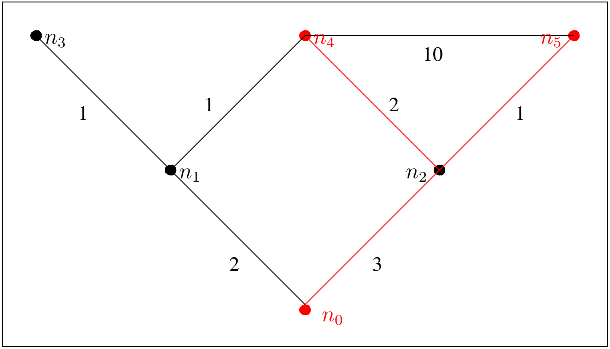


Figure: A Steiner Tree with cost 6

The Steiner Tree Problem

When $T = V$, this is the tractable Minimum Spanning Tree (MST) problem

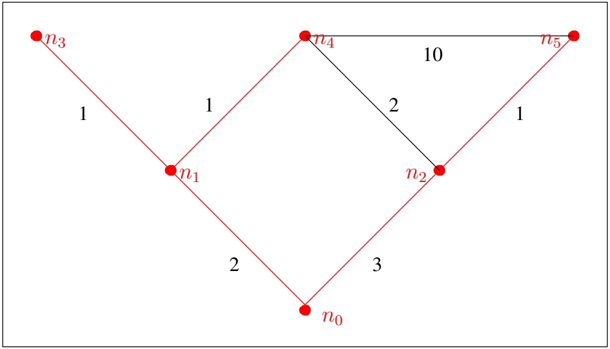


Figure: A Minimum Spanning Tree

The Steiner Tree Problem

Otherwise, equivalent to finding best set of non-terminal nodes P to span, known as *Steiner Points*

- Steiner Tree is then MST over $P \cup T$

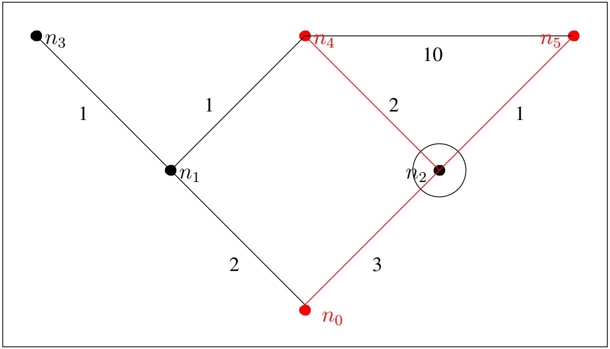


Figure: $P = \{n_2\}$

Equivalence of ST and P^+

Given a Steiner Tree Problem S with graph $W = \langle V, E \rangle$ and terminal set T, we construct an equivalent planning problem $P_S = \langle F, I, O, G \rangle$ with **no deletes**:

- $F = V$
- $I = \{n_0\}$ for some $n_0 \in T$
- $O = \{a_{n,n'} | \langle n, n' \rangle \in E\}$, $Pre(a_{n,n'}) = n$, $Add(a_{n,n'}) = n'$
- $G = T$

Theorem

The optimal plans for P_S encode the Steiner Trees for the problem $S = \langle W, T \rangle$, and $c(S) = c^*(P_S) = c^*(P_S^+)$

Question: What do heuristics such as h_{add}, h^s_a, and h_{FF} do on this problem?

Steiner Trees vs. Trees of Shortest Paths

Theorem

Given a Steiner Tree Problem P_S, independence-based relaxed plan heuristics compute tree-of-shortest-paths approximation

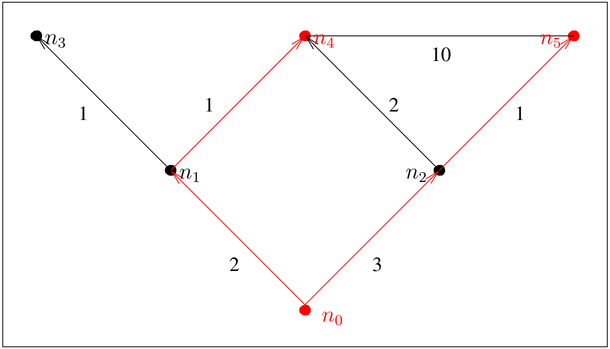


Figure: Tree of shortest paths rooted at n_0 with cost 7

Steiner Trees vs. Trees of Shortest Paths

Given a root node r and a set of terminal nodes T, a tree of shortest paths consists of the union of the paths from r to each $t \in T$

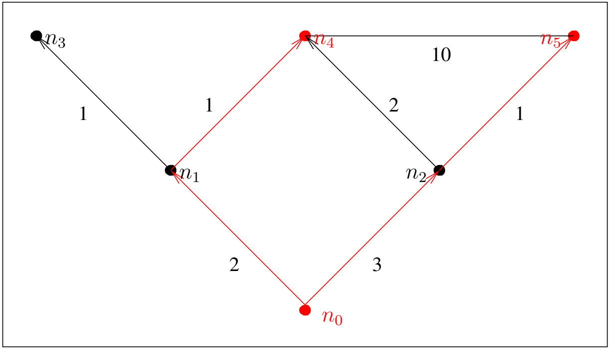


Figure: Tree of shortest paths rooted at n_0 with cost 7

Question: Can we do better?

Several approximation algorithms proposed previously (Charikar *et al.* [1998], Zelikovsky [1997]) but not readily adaptable to our problem

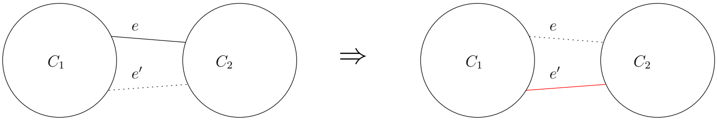
Improving Candidate Steiner Trees

A necessary property of Steiner trees: A Steiner Tree S is an MST over the set of nodes that it spans

Idea: Ensure that the tree satisfies this criterion

For each edge e in the tree, check whether there is a cheaper edge e' between the two connected components C_1 and C_2 linked by e
$\rightarrow$ If so, replace e with e'

When no more edges can be replaced, tree is an MST



Example

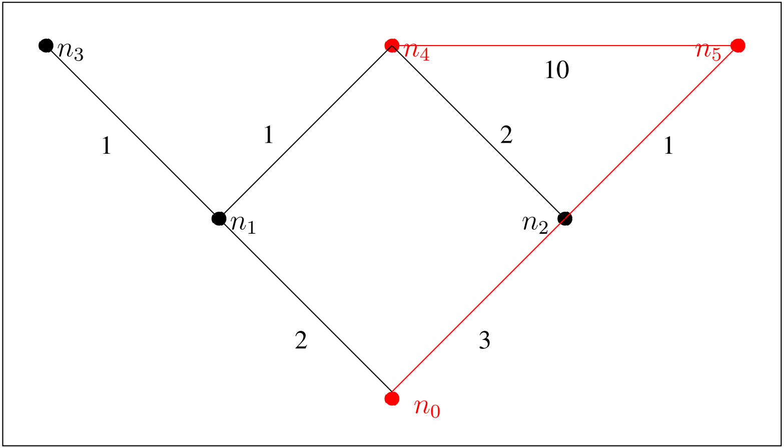


Figure: A candidate Steiner Tree with cost 14

Example

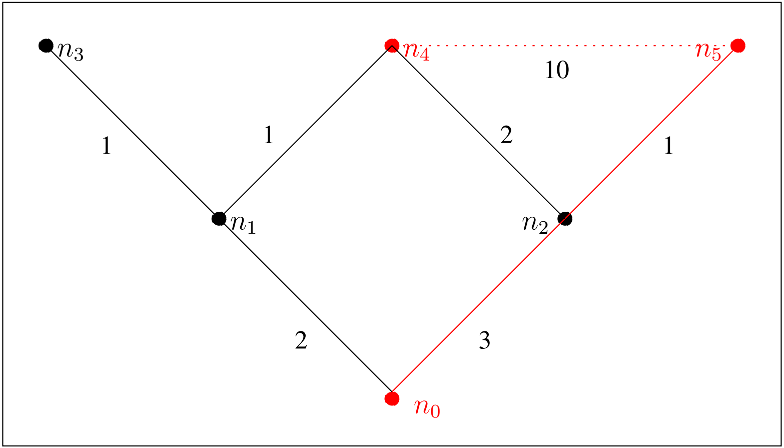


Figure: Removing edge $\langle n_4, n_5 \rangle$ leaves connected components $\{n_0, n_2, n_5\}$ and $\{n_4\}$

Example

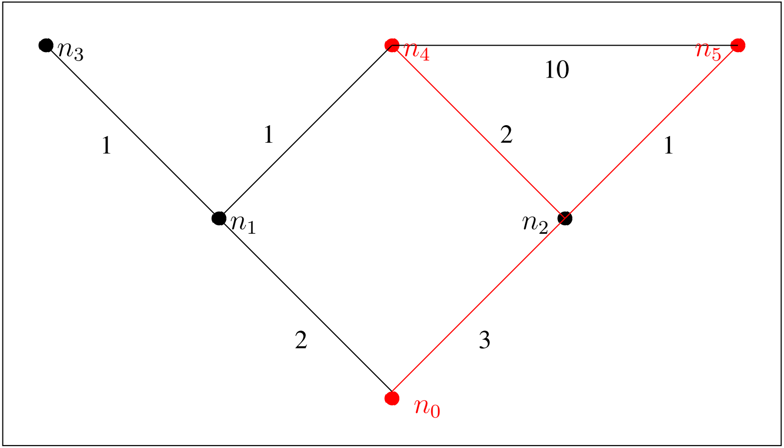


Figure: We reconnect the connected components with $\langle n_2, n_4 \rangle$

Example

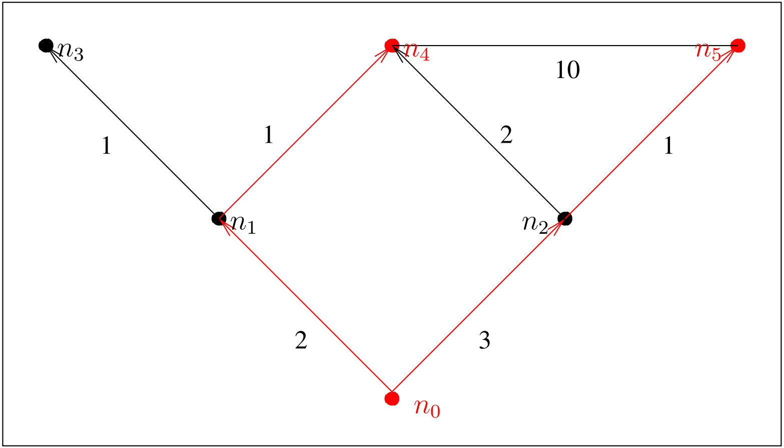


Figure: MST criterion is *necessary*, not *sufficient*

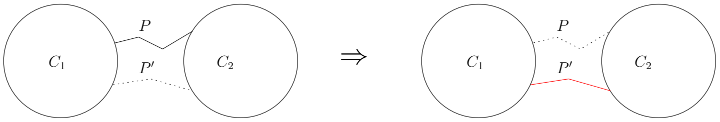
Replacing Paths

An improvement: Replace *paths* rather than edges

Replace a path P connecting two connected components C_1 and C_2 with a cheaper path P' that connects the same two connected components

P cannot cross any terminal node $t \in T$

Observation: This improvement leads to local search procedure that yields lower cost Candidate Steiner Trees with a possibly different set of Steiner Points P



Example

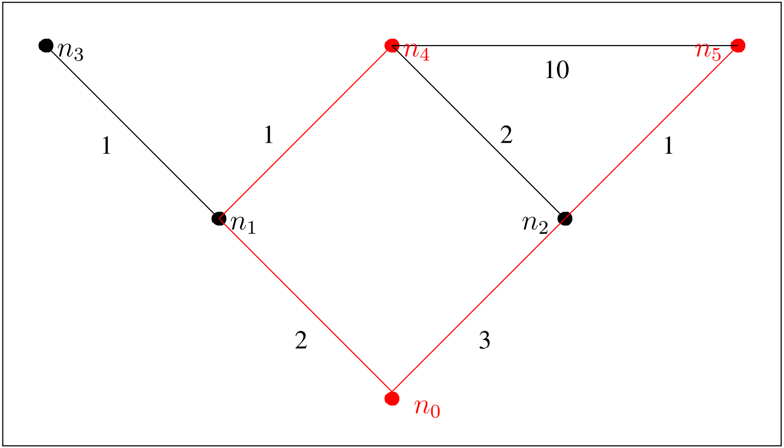


Figure: A candidate Steiner Tree

Example

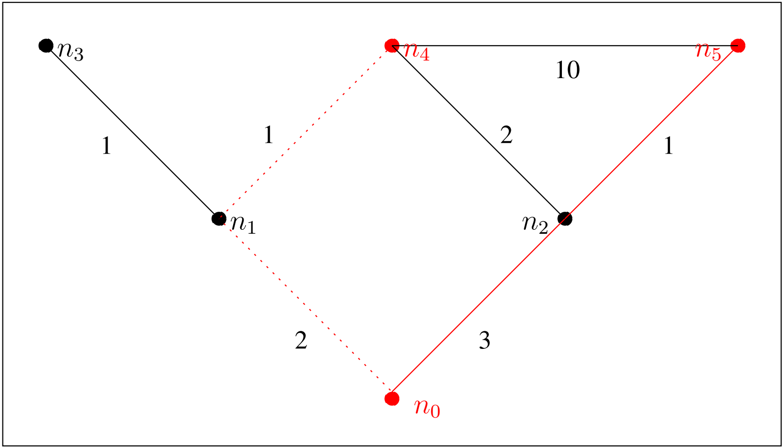


Figure: Removing path $P = \langle n_0, n_1, n_4 \rangle$ leaves connected components $\{n_0, n_2, n_5\}$ and $\{n_4\}$

Example

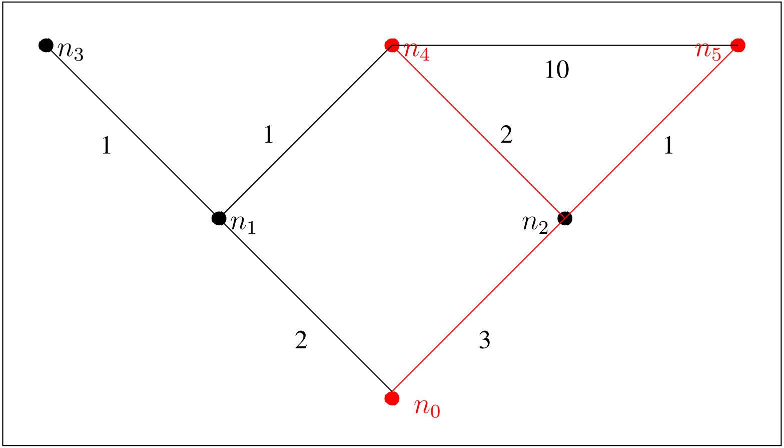


Figure: We reconnect the connected components with $P' = \langle n_2, n_4 \rangle$

Improving Relaxed Plans

Idea: Improve a relaxed plan $\pi^+(s)$ using edge/path replacement methods

Problem: Delete-relaxation graphs are *directed hypergraphs* rather than graphs

Solutions are *directed hyperpaths*

However, it turns out main ideas can be adapted to this setting

Improving Relaxed Plans

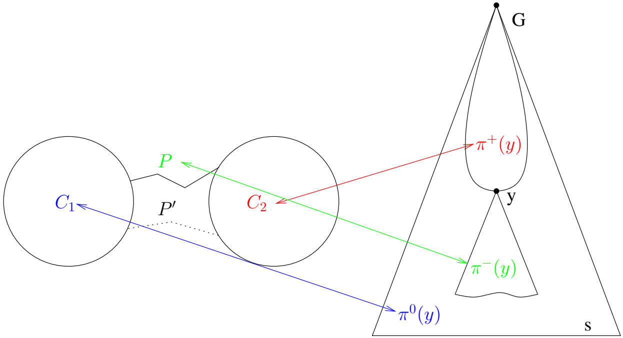
Given a fluent y, we partition $\pi(s)$ into three disjoint sets:

$\pi^-(y;s)$ Actions required *only* to achieve y

$\pi^+(y;s)$ Actions that depend on y being achieved

$\pi^0(y;s)$ All other actions

Improving Relaxed Plans



$\pi^0(y;s)$ and $\pi^+(y;s)$ correspond to connected components rooted at s and y respectively, while $\pi^-(y;s)$ corresponds to a path from $\pi^0(y;s)$ to y

Relaxed Plan Improvement Algorithm

Look for fluent y in $\pi(s)$ such that

$$cost(\pi(y; s')) < cost(\pi^{-}(y; s))$$

where

$$s' = s \cup \{p \mid a_p(s) \in \pi^0(y; s)\}$$

Replace $\pi^{-}(y; s)$ with $\pi(y; s')$ if such y is found:

$$\pi'(s) = \pi^0(y; s) \cup \pi^{+}(y; s) \cup \pi(y; s')$$

When no further improvement possible, heuristic is cost of resulting plan:

$$h_{lst}(s) = cost(\pi)$$

Example

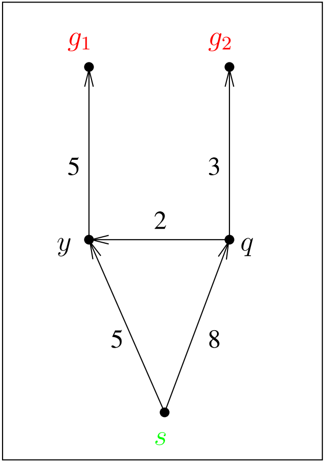


Figure: A planning problem P with $I = \{s\}$ and $G = \{g_1, g_2\}$

Example

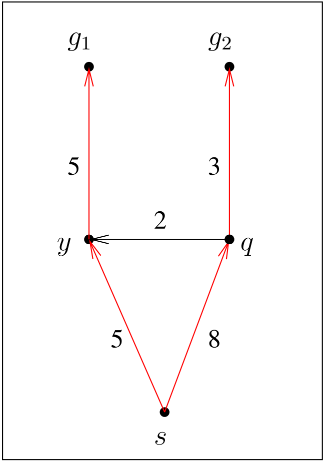


Figure: A shortest paths plan for P

Example

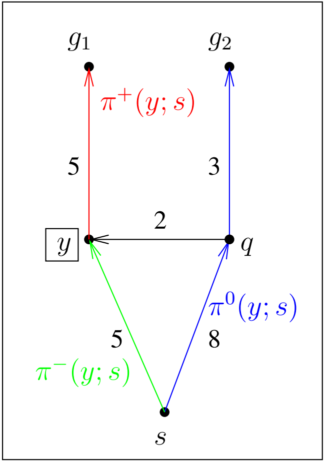


Figure: Plan partitioned for fluent y

Example

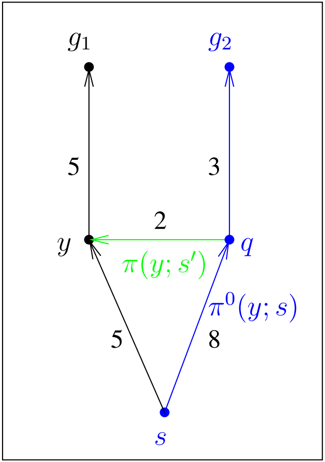


Figure: Calculation of $\pi(y; s')$ from $s' = \{s, q, g_2\}$

Example

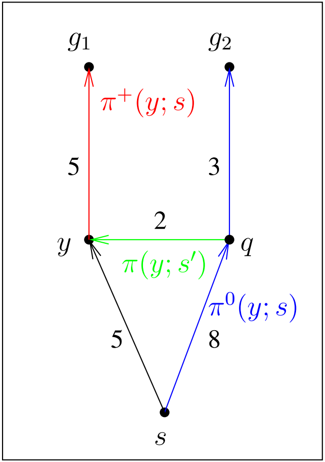


Figure: New plan $\pi'(s) = \pi^0(y;s) \cup \pi^+(y;s) \cup \pi(y;s')$

Conclusions

- The optimal delete relaxation heuristic h^+ is admissible and powerful, but calculating it is NP-hard
  - Sub-optimal solutions have proven to be effective heuristics
- h_{add} suffers from two problems:
  1. Over-counting of actions when combining estimates
  2. The independence assumption
- Overcounting problem can be solved with *relaxed plan heuristics*
- Steiner Tree heuristic is a preliminary attempt at improving over the independence assumption

Part II

Exact Computation of h^+

Heuristic h^+

- Defined in terms of the delete relaxation P^+

$$h^+(P[I = s]) = h^*(P^+[I = s])$$

- Logical approach to compute h^+

Heuristic h^+

- Defined in terms of the delete relaxation P^+

$$h^+(P[I = s]) = h^*(P^+[I = s])$$

- Logical approach to compute h^+
- Delete relaxation encoded as a propositional theory $T(P^+)$
- We will show that $h^+(P[I = s])$ is equal to the **rank** $r^*(T(P^+) \wedge I(s))$ of the theory $T(P^+) \wedge I(s)$ where $T(P^+)$ encodes P^+ and $I(s)$ encodes the state s

Ranks of logical theories

- A literal-ranking function r maps literals into real numbers
- The rank of a model ω is the sum of the ranks of the literals it makes true:
$$r(\omega) = \sum_{\omega \models L} r(L)$$
- The rank of a theory Γ is the minimum rank of its models
$$r^*(\Gamma) = \min_{\omega \models \Gamma} r(\omega)$$

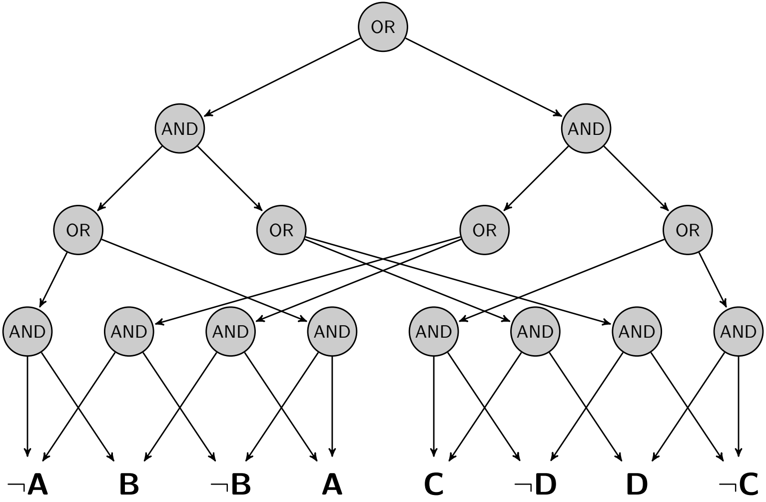
Ranks of logical theories

- A literal-ranking function r maps literals into real numbers
- The rank of a model ω is the sum of the ranks of the literals it makes true:
$$r(\omega) = \sum_{\omega \models L} r(L)$$
- The rank of a theory Γ is the minimum rank of its models
$$r^*(\Gamma) = \min_{\omega \models \Gamma} r(\omega)$$
- If the γ is in d-DNNF format, its rank can be computed in linear time for every literal-ranking function r

Negation Normal Form

- A formula is in NNF if it is made of conjunctions, disjunctions and negations, and the negations only appear at the literal level
- Alternatively, a NNF formula is a rooted DAG whose leaves are literals or the constants **true** and **false**, and whose internal nodes stand for conjunctions (AND nodes) or disjunctions (OR nodes)
- The NNF fragment is the collection of all formulas in NNF. It is a **complete** fragment for propositional logic

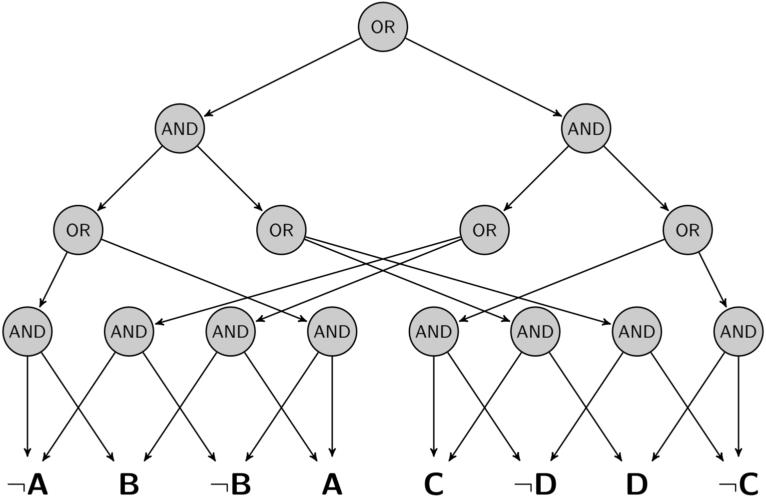
Example: NNF formula



Deterministic Decomposable Negation Normal Form

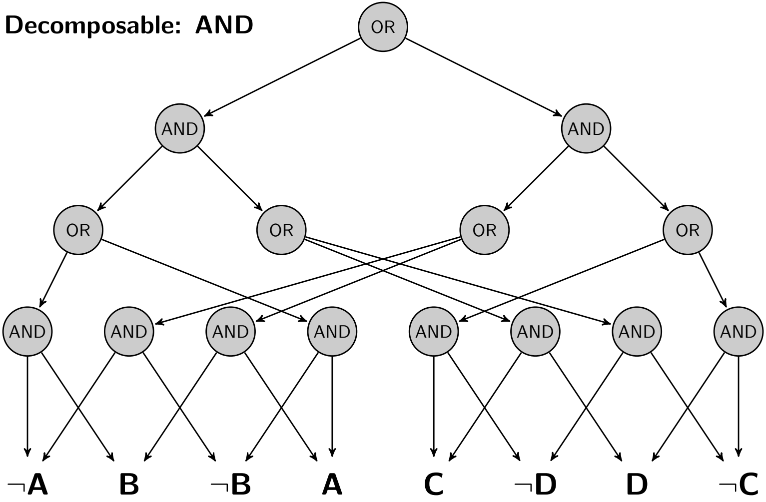
- A NNF is **decomposable** if the subformulas associated to the children of an AND node share no variables
- A NNF is **deterministic** if the subformulas associated to the children an OR node are pairwise inconsistent
- The d-DNNF fragment is the collection of all formulas that are NNF, decomposable and deterministic. It is a **complete** fragment for propositional logic

Example: d-DNNF formula



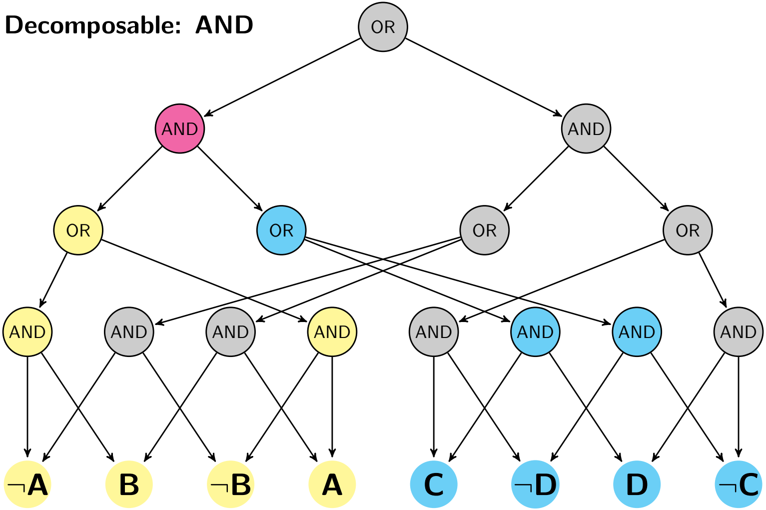
Example: d-DNNF formula

Decomposable: AND



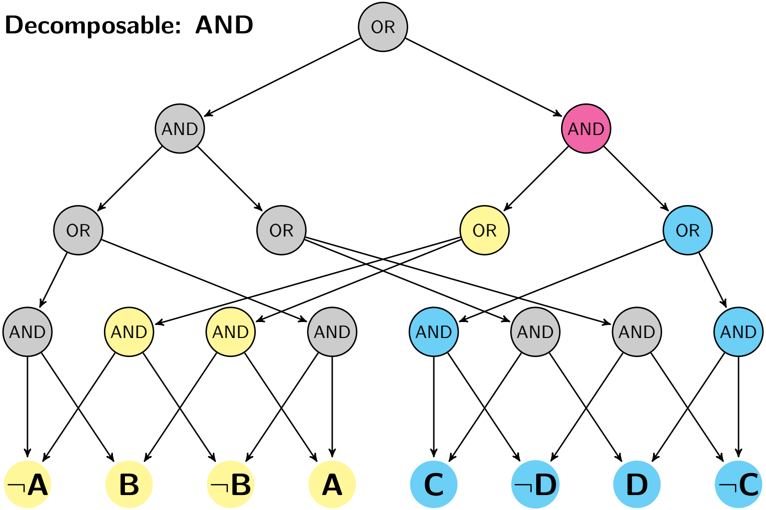
Example: d-DNNF formula

Decomposable: AND



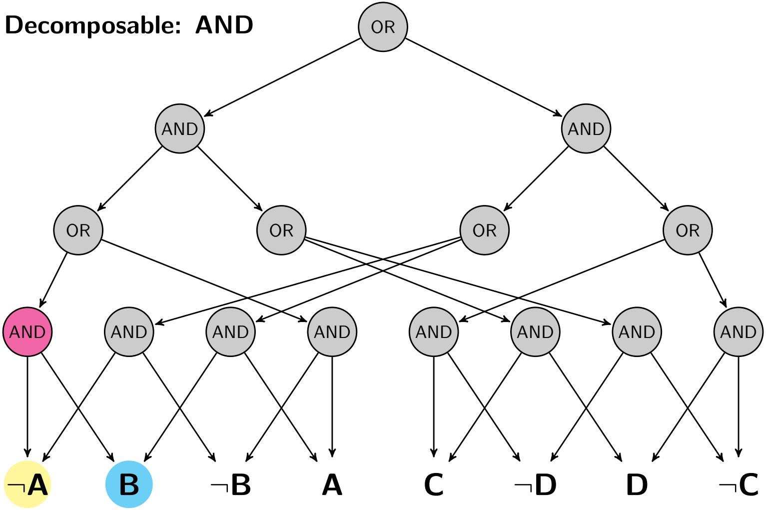
Example: d-DNNF formula

Decomposable: AND



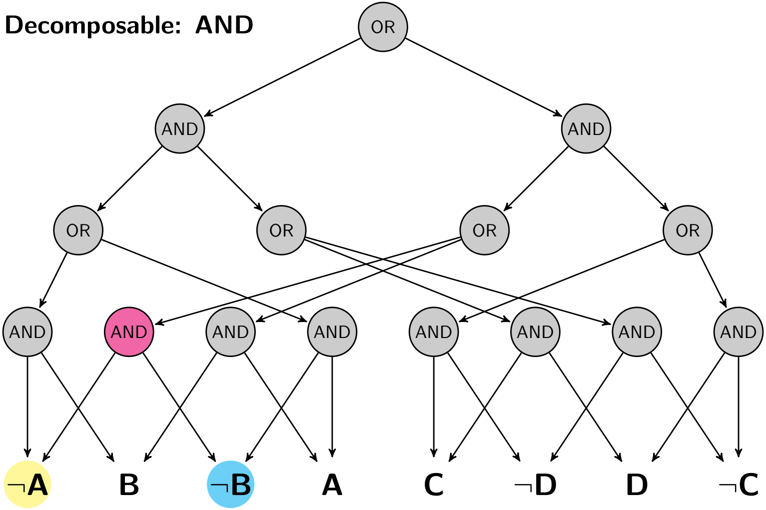
Example: d-DNNF formula

Decomposable: AND



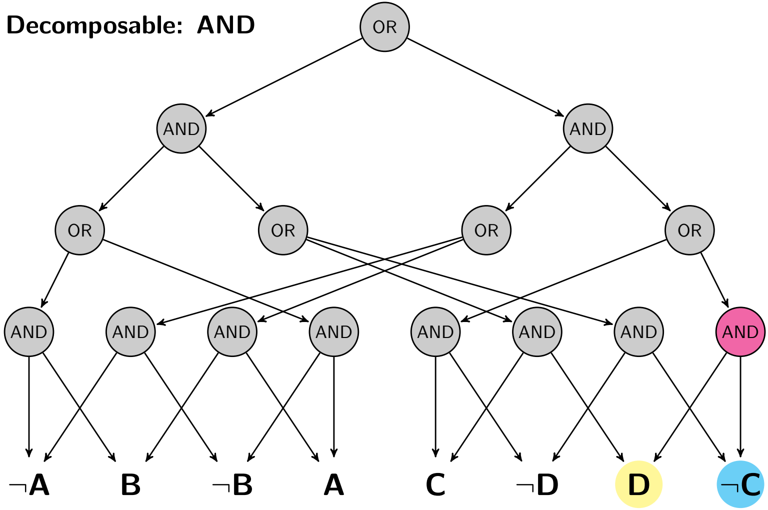
Example: d-DNNF formula

Decomposable: AND



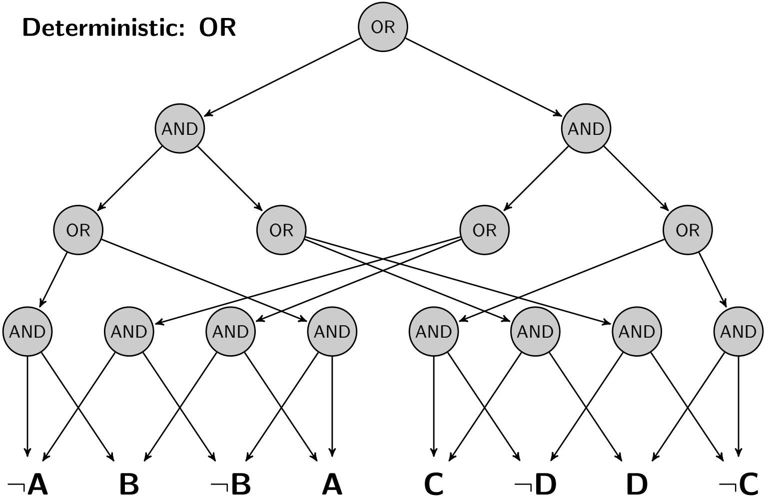
Example: d-DNNF formula

Decomposable: AND



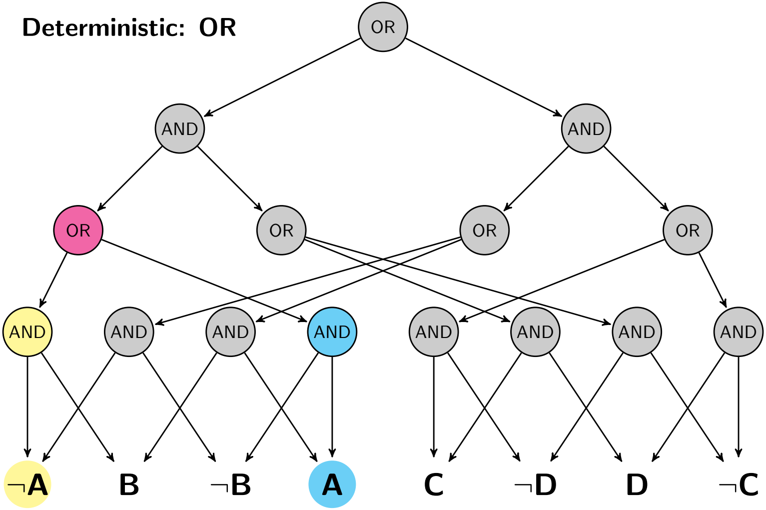
Example: d-DNNF formula

Deterministic: OR



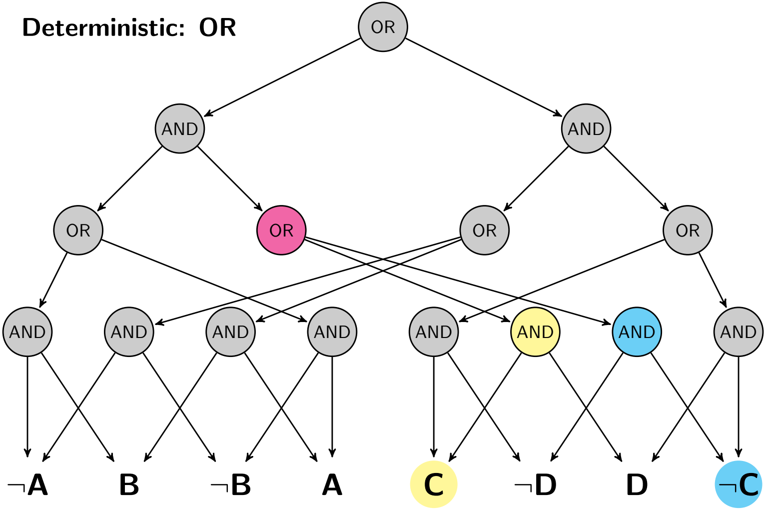
Example: d-DNNF formula

Deterministic: OR



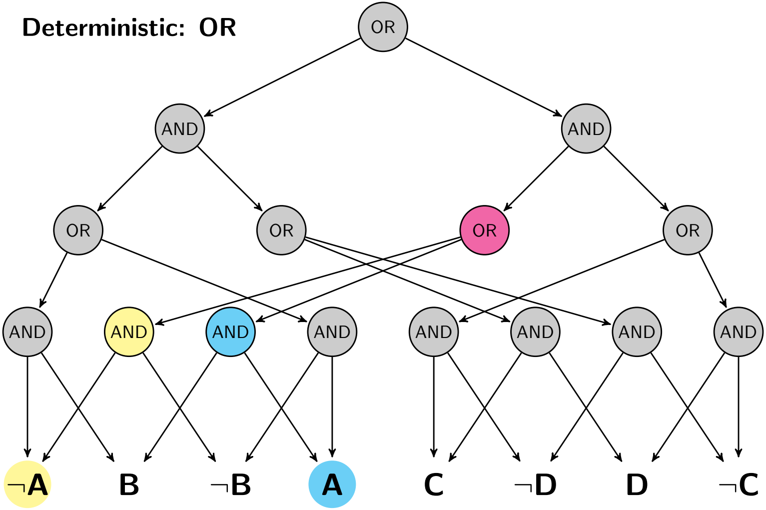
Example: d-DNNF formula

Deterministic: OR



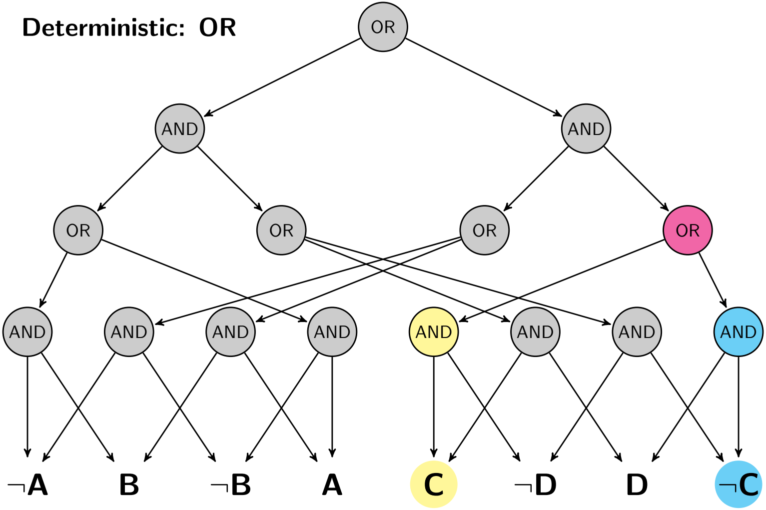
Example: d-DNNF formula

Deterministic: OR



Example: d-DNNF formula

Deterministic: OR

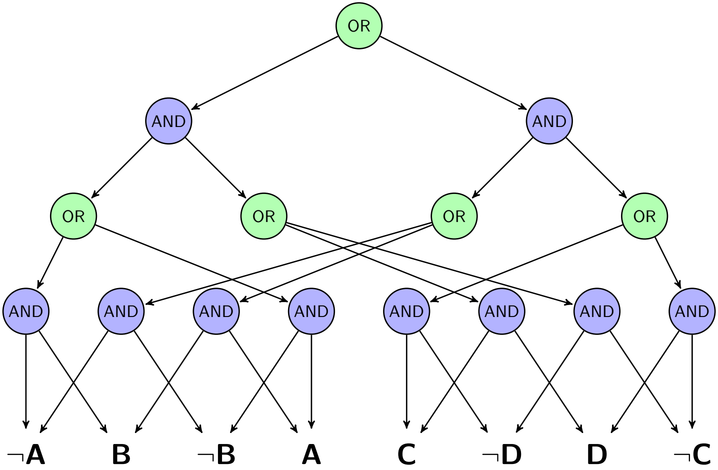


d-DNNF: Operations

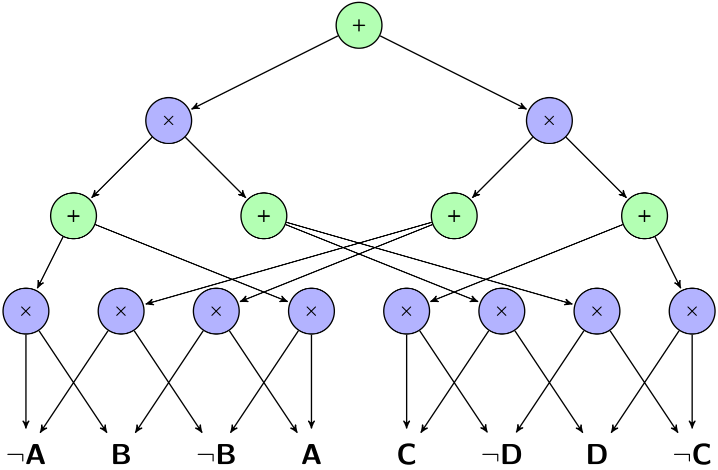
d-DNNFs support a number of operations in polynomial time:

1. Satisfiability
2. Validity
3. Clause entailment
4. Model counting
5. Computation of ranks
6. others

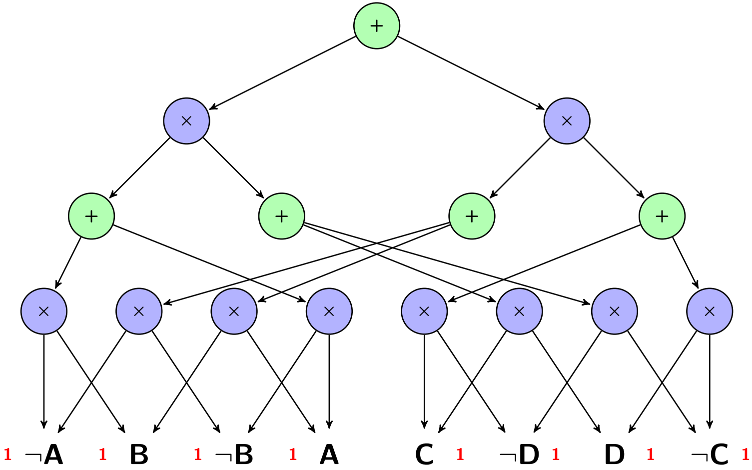
Example: Model Counting



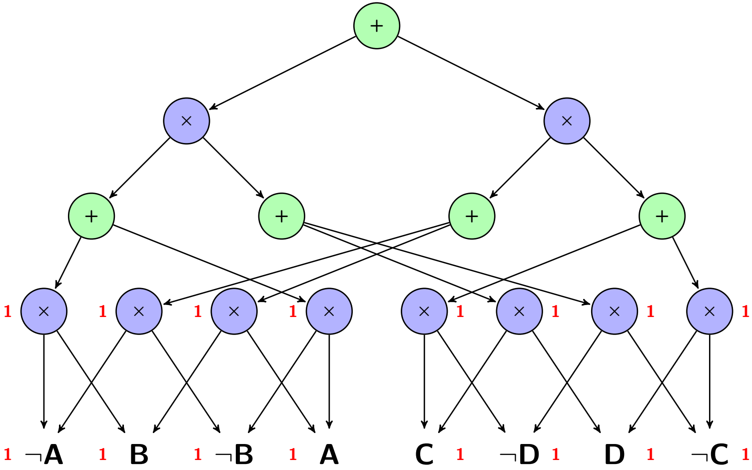
Example: Model Counting



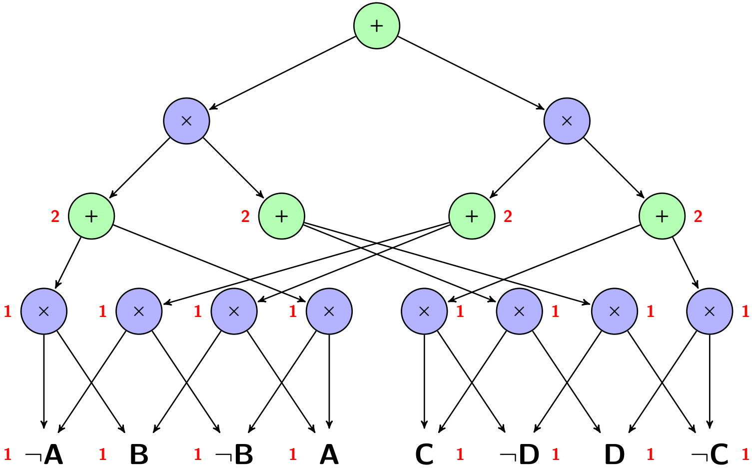
Example: Model Counting



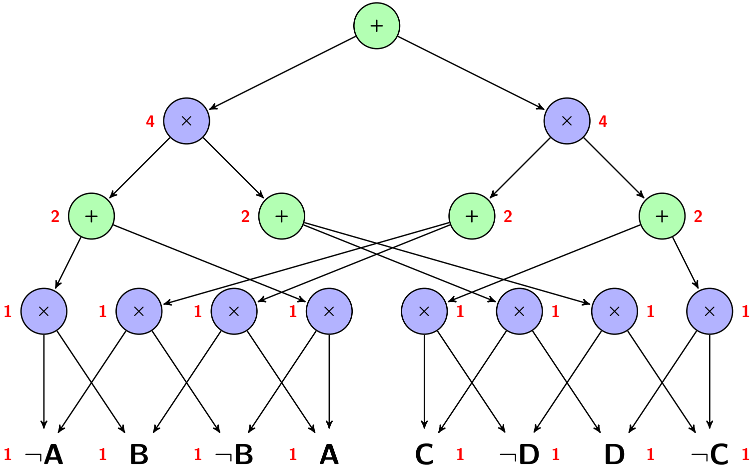
Example: Model Counting



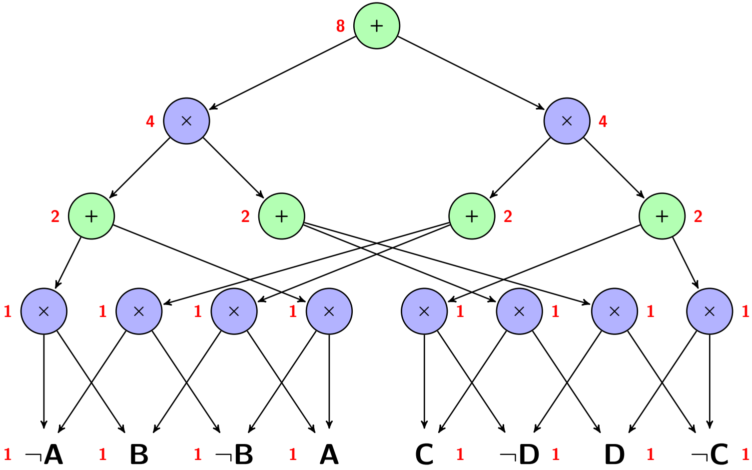
Example: Model Counting



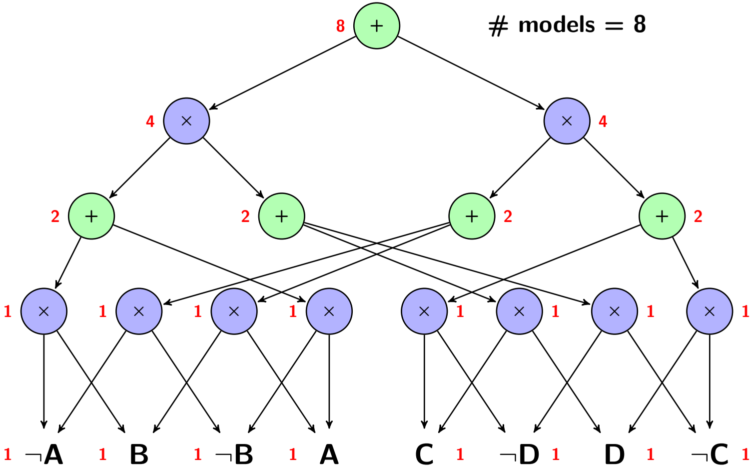
Example: Model Counting



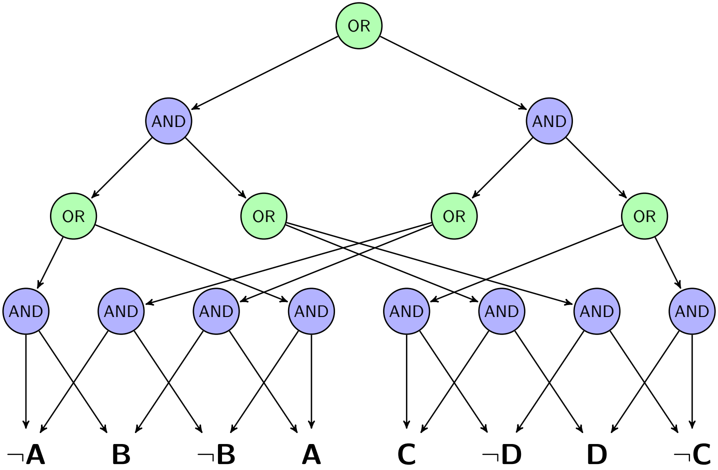
Example: Model Counting



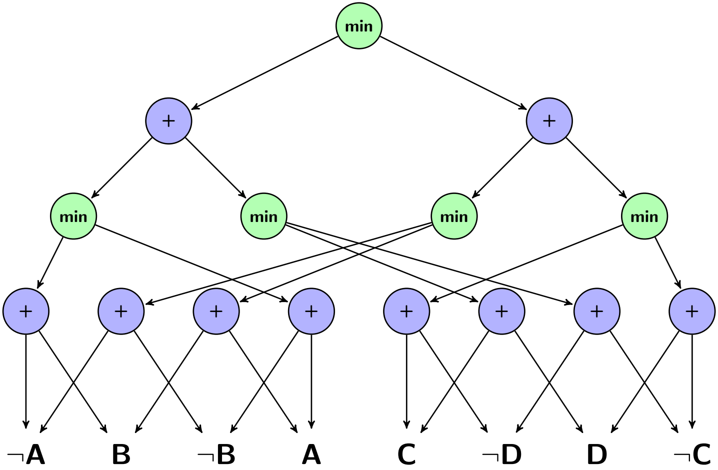
Example: Model Counting



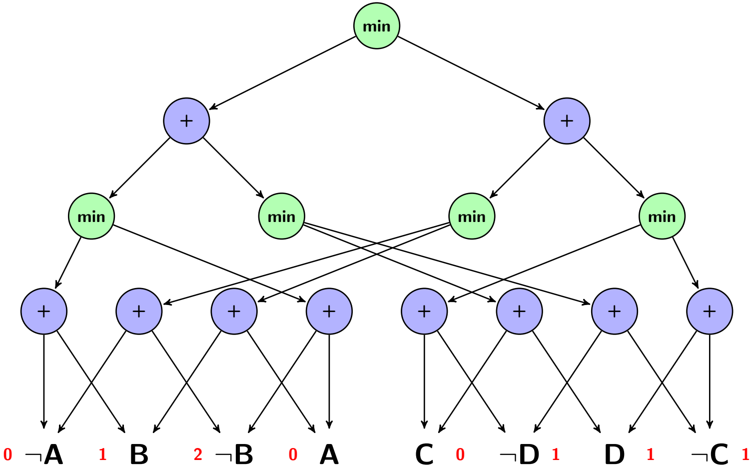
Example: Rank Computation



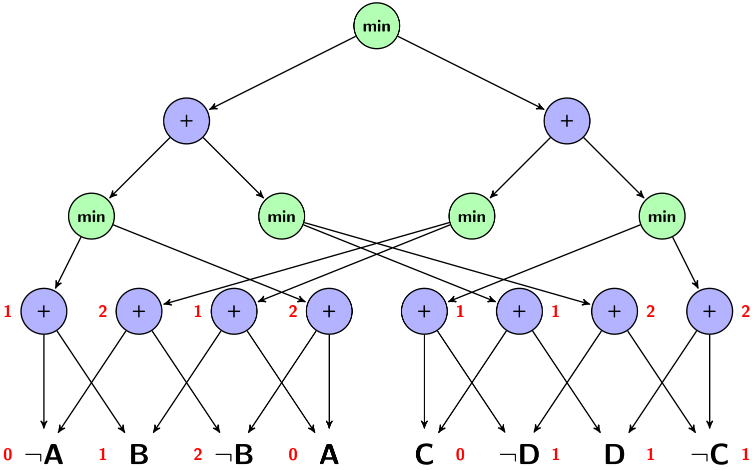
Example: Rank Computation



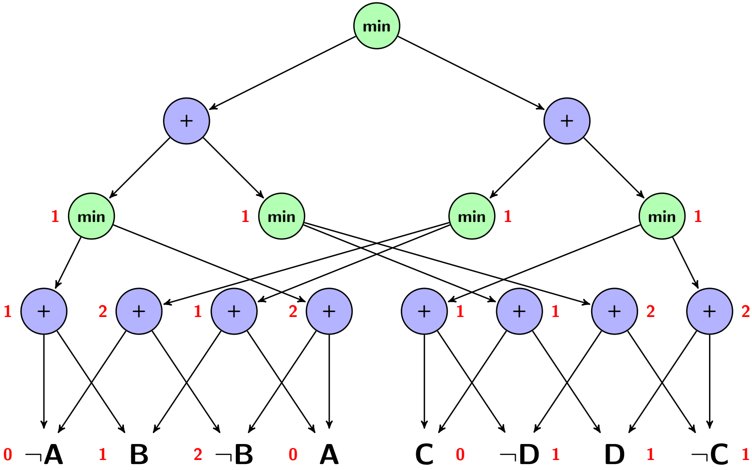
Example: Rank Computation



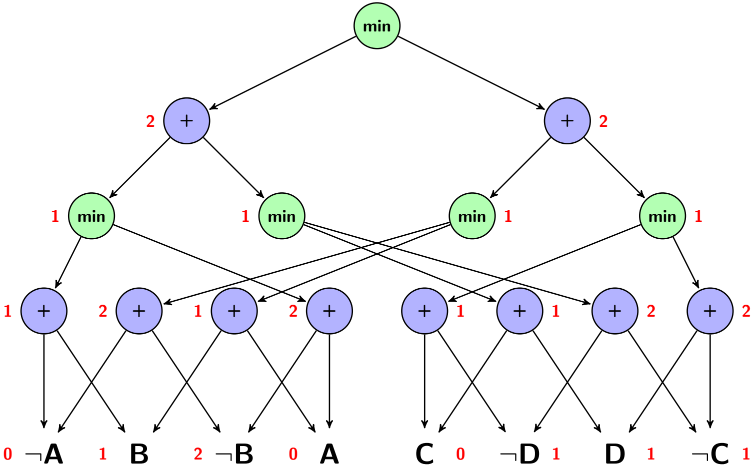
Example: Rank Computation



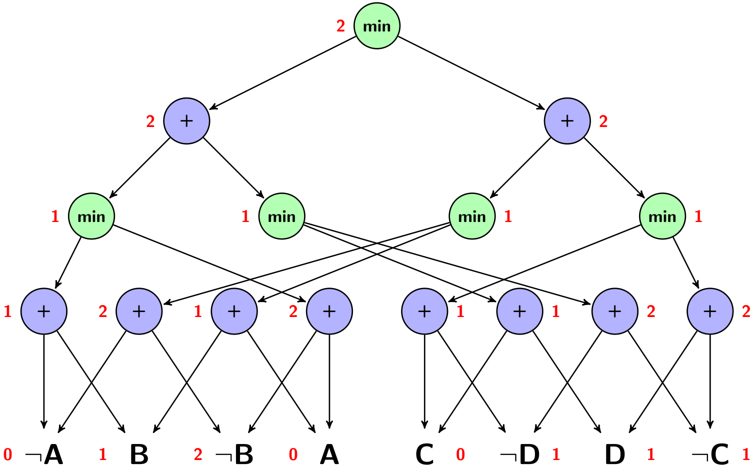
Example: Rank Computation



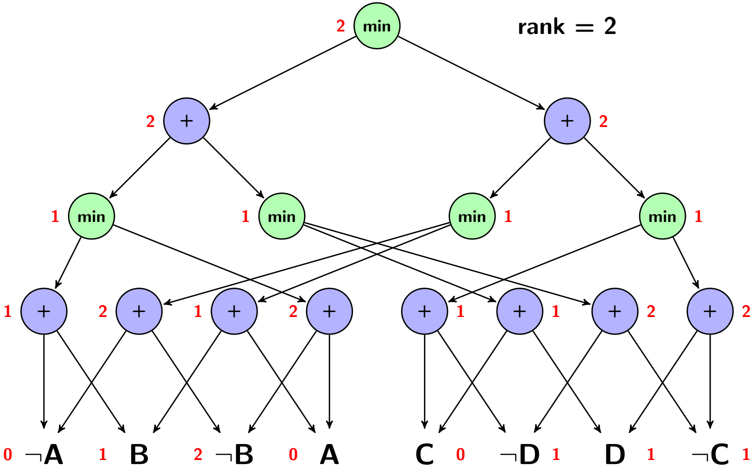
Example: Rank Computation



Example: Rank Computation



Example: Rank Computation



Ranks and d-DNNF

Theorem

Let Γ be a d-DNNF formula and r a literal ranking function. Then, the rank $r^*(\Gamma)$ can be computed in linear time in the size of Γ.

Compilaton into d-DNNF 1/2

- If the theory is not in d-DNNF, it needs to be first compiled into d-DNNF using a compiler such as Darwiche's c2d compiler
- The compilation takes exponential time and space in the worst case. Otherwise, some important complexity classes would collapse to P
- However, the compilation needs to be computed only once in order to compute any number of rank computations with respect to different literal-ranking functions

Compilaton into d-DNNF 2/2

- If the compilation succeeds with a “small” d-DNNF. The compilation time can be traded off when computing a large number of rank operations
- The compilation time and space is exponential in a parameter known as the treewidth of the theory
- If the formula is not compilable due to high treewidth, it can be relaxed into a simpler one whose would be lower bounds on the rank of the original formula

Encodings of delete relaxations

We now see how to encode a delete-relaxation P^+ into a logical theory $T(P^+)$ that allow us to compute h^+ exactly as a rank operation. We consider two encodings:

- Stratified encodings that use a time horizon
- LP encodings that use no time horizon

Stratified encodings of P^+ 1/2

- Plans for a STRIPS problem $P^+ = \langle F, I, O, G\rangle$ with horizon n can be obtained from models of propositional theory $T_n(P^+)$:

1. **Actions:** For $i = 0, \ldots, n-1$ and all action a:

$$a_i \supset p_i \text{ for } p \in Pre(a)$$
$$C_i \wedge a_i \supset p_{i+1} \text{ for each effect } a : C \longrightarrow p$$

2. **Frame:** For $i = 0, \ldots, n-1$ and all fluent p:

$$p_i \supset p_{i+1}$$
$$\neg p_i \wedge (\textstyle\bigwedge_{a:C\rightarrow p}(\neg a_i \vee \neg C)) \supset \neg p_{i+1}$$

3. **Seriality:** For $i = 0, \ldots, n-1$ and $a \neq a'$, $\neg(a_i \wedge a'_i)$
4. **Goals and Init:** free, defined by formulas I_0 and G_n

Stratified encodings of P^+ 2/2

- Heuristic $h^+(P[I = s, G = g]) = h^*(P^+[I = s, G = g])$ can be defined as the rank $r^*(T_n(P^+) \wedge I_0 \wedge G_n)$ where:

  1. Horizon n is equal to $\min\{\#\text{actions}, \#\text{fluents}\}$
  2. Literal ranking function:

$$r(L) = \begin{cases} c(a) & \text{if } L = a_i \\ 0 & \text{otherwise} \end{cases}$$

Theorem

Let $\Pi_n(P^+)$ be the compilation of theory $T_n(P^+)$ in d-DNNF where n is a sufficiently large horizon. Then, the heuristic values $h^+(P[I = s, G = g])$ for any initial and goal situation s and g, and any cost function c, can be computed from $\Pi_n(P^+)$ in linear time.

Horizons

- The SAT encoding of a STRIPS problem P requires an exponential horizon in the worst case
- The SAT encoding of the delete-relaxation P^+ requires a linear horizon, yet in most applications this horizon is still too large to compile the theory $T_n(P^+)$
- However, we can achieve a more compact encoding of P^+ that **requires no time horizon**
- This encoding is called the LP encoding as it is obtained from a set of **positive Horn clauses**

The LP encoding of P^+ 1/2

- Obtained from LP rules of the form:

$$p \leftarrow Pre(a), a$$

  for each (positive) effect $p \in Add(a)$

- Additionally, we consider rules of the form:

$$p \leftarrow set(p)$$

- Focus is on a class of minimal models (stable models) that have an **implicit stratification** in correspondence with the **temporal stratification**

The LP encoding of P^+ 2/2

- Models are **grounded on the actions** as all fluents are required to have well-founded support on them
- Furthermore, actions do no imply their preconditions. Not a problem, since cost of actions are positive, and they require their preconditions to have an effect
- Models that make actions true without their preconditions are not preferred

SAT encoding of the LP 1/2

- Let $L(P)$ be the LP encoding of planning problem P^+
- Let $wffc(L(P))$ be the **well-founded fluent completion** of $L(P)$: a completion formula that forces each fluent p to have a well-founded support
- Then,

$$h^+(P[I = s, G = g]) = r^*(wffc(L(P)) \cup I(s) \cup g)$$

where

$$I(s) = \{set(p) : p \in s\}$$

SAT encoding of the LP 2/2

- $wffc(L(P))$ **picks up** the models of $L(P)$ in which each fluent has a non-circular support that is based on the actions made true in the model
- Let's say that $L(P)$ is acyclic if the directed graph, formed by connecting the atoms in the body of a rule to the head, is acyclic
- If $L(P)$ is acyclic, $wffc(P)$ is Clark's completion applied to the fluent literals

Clark's completion of the LP

- For each fluent p with rules $p \leftarrow B_i$ for $i = 1, \ldots, n$, add the formula

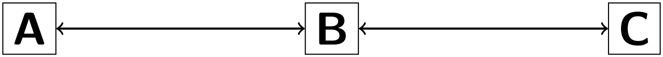
$$\begin{aligned} p &\supset B_1 \vee \cdots \vee B_n \\ B_i &\supset p \end{aligned}$$

- If there are no rules for p, add the formula $\neg p$
- In the presence of cycles, Clark's completion is not enough

Well-founded fluent completion

- From Answer Set Programming
- Completion adds new atoms and rules to the LP, providing a consistent and partial ordering of the fluents
- Then, Clark's completion of the extended LP is computed

Example 1/3



- Actions $move(x, y)$ and fluents $at(x)$. LP rules:

$$at(y) \leftarrow at(x), move(x, y)$$
$$at(y) \leftarrow set(at(y))$$

- Let $s = \{at(A)\}$ and $g = \{at(C)\}$ be init and goal states, and let all actions have unit cost except $c(move(A, B)) = 10$
- Best plan (in P and P^+) is $\pi = \{move(A, B), move(B, C)\}$ so that $h^+(P[I = s, G = g]) = h^*(P^+[I = s, G = g]) = 11$

Example 2/3

- We have $I(s) = \{set(at(A)), \neg set(at(B)), \neg set(at(C))\}$
- Clark's completion is

$$
\begin{aligned}
at(A) &\equiv (at(B) \wedge move(B, A)) \vee set(At(A)) \\
at(B) &\equiv (at(A) \wedge move(A, B)) \vee (at(C) \wedge move(C, B)) \vee set(At(B)) \\
at(C) &\equiv (at(B) \wedge move(B, C)) \vee set(At(C))
\end{aligned}
$$

- Best model corresponds to actions

$$\{move(B, C), move(C, B)\}$$

which has circular support for $at(C)$ and has rank equal to 2

Example 3/3

- The $wffc(L(P))$ is Clark's completion of the modified LP in which each rule

$$at(y) \leftarrow at(x), move(x,y)$$

is replaced by the rules:

$$\begin{aligned} r_k &\leftarrow NOT\ \ at(y) \prec at(x), at(x), move(x,y) \\ at(y) &\leftarrow r_k \\ at(x) \prec at(y) &\leftarrow r_k \\ at(z) \prec at(y) &\leftarrow r_k, at(z) \prec at(x) \end{aligned}$$

where $z \in \{A, B, C\}$ and NOT is negation as failure

- $wffc(L(P))$ is Clark's completion of the modified LP

Heuristic Computation

Theorem

Let $\Pi(P)$ *be the compilation of theory wffc*$(L(P))$ *in d-DNNF. Then, for any initial and goal situation* s *and* g*, and any cost function* c*, the heuristic* $h^+(P[I = s, G = g])$ *can be computed from* $\Pi(P)$ *in linear time.*

Extended Planning Model

- This framework allow us to extend the planning model by considering positive or negative costs $c(p)$ for fluents, in addition to the positive action costs $c(a)$
- Given a planning problem P and plan π, the cost $c(\pi)$ of a plan is given by the cost of actions in π and the cost of the atoms $F(\pi)$ **made true by π (at any time)**

$$c(\pi) = \sum_{a \in \pi} c(a) + \sum_{p \in F(\pi)} c(p)$$

- The cost of a problem P is defined as

$$c^*(P) = \min_{\pi} c(\pi)$$

- This model extends classical planning by allowing to express non-trivial preferences . . .

Scope of the model

The model is simple and flexible, and can represent:

- **Terminal costs:** a fluent p can be rewarded or penalized if true at the end of the plan, by means of a new atom p' initialized to false, and conditional effect $p \rightarrow p'$ for action *End*
- **Goals:** not strictly required since can be modeled as a sufficiently high terminal reward
- **Soft Goals:** modeled as terminal rewards
- **Rewards on Conjunctions:** using actions $Collect(p_1, \ldots, p_n)$

Not so simple to represent repeated costs or rewards, penalties on sets of atoms, partial preferences, ...

Ranking Function

- The only fix required is to use a ranking function of the form:

$$r(L) = \begin{cases} c(a) & \text{if } L = a \\ c(p) & \text{if } L = p \\ 0 & \text{otherwise} \end{cases}$$

- Then, the LP encodings still works and the heuristic $h^+(P[I = s, G = g])$ is the rank $r^*(T(P^+) \wedge I(s) \wedge g)$

Conclusions

- h^+ efficiently computable when $T(P^+)$ is in d-DNNF
- Can compute exact heuristics for more general planning models

Part III

The h^m Heuristics

h_{max} Heuristic 1/2

Obtained by replacing the $\sum$ in h_{add} with max:

$$h_{max}(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ \min\limits_{a \in O(p)} cost(a) + h_{max}(Pre(a); s) & \text{otherwise} \end{cases}$$

where

$$O(p) = \text{"operators that add } p\text{"}$$

$$h_{max}(P; s) = \max_{p \in P} h_{max}(p; s)$$

h_{max} Heuristic 2/2

- $h_{max}(P; s)$ is cost to achieve a costliest atom in P from s
- It is computed using Dijkstra algorithm, and provides admissible estimates that can be used for optimal planning, yet its values are often low and non-informative
- This heuristic is also referred as the h^1 heuristic

h^2 Heuristic 1/2

Haslum and Geffner observed that h_{max} can be modified to compute costs for **pairs of atoms**:

- $h^2(P; s)$ estimates the cost to achieve a costliest pair $\{p, q\} \subseteq P$ from s
- If $\{p, q\} \subseteq s$, then the cost of $\{p, q\}$ is zero as both atoms are already achieved
- Otherwise, need to consider ways to **achieve** $\{p, q\}$ from s
- Let $O(pq)$ be the set of operators that add p and q, and $O(p|q)$ be the set of operators that add p and don't delete q

h^2 Heuristic 2/2

We can achieve $\{p, q\}$ by either:

1. applying an operator in $O(pq)$ that achieves p and q simultaneously

h^2 Heuristic 2/2

We can achieve $\{p, q\}$ by either:

1. applying an operator in $O(pq)$ that achieves p and q simultaneously
2. applying an operator in $O(p|q)$ that achieves p and doesn't delete q from a state that already contains q, or

h^2 Heuristic 2/2

We can achieve $\{p, q\}$ by either:

1. applying an operator in $O(pq)$ that achieves p and q simultaneously
2. applying an operator in $O(p|q)$ that achieves p and doesn't delete q from a state that already contains q, or
3. applying an operator in $O(q|p)$ that achieves q and doesn't delete p from a state that already contains p

h^2 Regression

Formula that expresses the different ways to achieve (regress) the pair $\{p, q\}$ thru action a:

$$R(\{p,q\},a) = \begin{cases} Pre(a) & \text{if } a \in O(pq) \\ Pre(a) \cup \{q\} & \text{if } a \in O(p|q) \\ Pre(a) \cup \{p\} & \text{if } a \in O(q|p) \\ undefined & \text{otherwise} \end{cases}$$

h^2 Equation

Defined by generalizing h_{max} fix-point equation to pairs of atoms:

$$h_{max}(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ \min_{a \in O(p)} \; cost(a) + h_{max}(Pre(a); s) & \text{otherwise} \end{cases}$$

h^2 Equation

Defined by generalizing h_{max} fix-point equation to pairs of atoms:

$$h_{max}(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ \min_{a \in O(p)} \; cost(a) + h_{max}(Pre(a); s) & \text{otherwise} \end{cases}$$

$$h^2(\{p, q\}; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \{p, q\} \subseteq s \\ \min_a \; cost(a) + h^2(R(\{p, q\}, a); s) & \text{otherwise} \end{cases}$$

where

$$h^2(P; s) = \max\{h^2(\{p, q\}; s) : p, q \in P\}$$

h^2 Computation

- h^2 computed with Dijkstra algorithm seeded at
  1. $h^2(\{p, q\}; s) = 0$ if $\{p, q\} \subseteq s$
  2. $h^2(\{p, q\}; s) = \infty$ if $\{p, q\} \not\subseteq s$
- In practical applications, h^2 is too expensive to compute for **forward-search** planners; used by **backward-search** planners
- However, the values $h^2(\cdot; s_0)$ are computed by almost all planners to obtain the **mutexes** of the planning problem

h^2 vs h^+

- Both h^2 and h^+ are admissible heuristics but neither dominates the other
- Therefore, h^2 is **not a delete-relaxation heuristic**
- Indeed, h^2 values for problem P and P^+ may differ

Mutexes 1/2

- A mutex (relation) between a pair of atoms p and q (with respect to initial state s_0) specifies that there is no reachable state from s_0 that makes both p and q true
- The pairs $\{p, q\}$ such that $h^2(\{p, q\}; s_0) = \infty$ are indeed in mutex relation
- Yet there are other mutex pairs whose h^2-value is less than ∞

Mutexes 2/2

- The **mutex graph** is an undirected graph defined over the atoms in which there is an edge $\{p, q\}$ iff the pair $\{p, q\}$ is mutex
- The **maximal cliques** of the graph define the **implicit** multi-valued variables of the problem
- A maximal clique C can be thought as a variable X with domain $D_X = C \cup \{\perp\}$ since
  1. no state makes two values of D_X true
  2. every state makes one value of D_X true

Example: Implicit Multi-valued Variables

Blocksworld with 4 blocks $\{A, B, C, D\}$:

$$\text{top-of-}A = \{\texttt{on(B,A)}, \texttt{on(C,A)}, \texttt{on(D,A)}, \texttt{clear(A)}\}$$

$$\text{bot-of-}A = \{\texttt{on(A,B)}, \texttt{on(A,C)}, \texttt{on(A,D)}, \texttt{table(A)}\}$$

$$\text{holding} = \{\texttt{hold(A)}, \texttt{hold(B)}, \texttt{hold(C)}, \texttt{hold(D)}, \perp\}$$

h^m Heuristic

- Same idea for h^2 can be generalized to subsets of size $\leq m$
- $h^m(P; s)$ is the cost to achieve a costliest subset of size $\leq m$ from s
- This time need to consider all possible ways to achieve (regress) a subset of size at most $\leq m$

h^m Regression

- For subset P of atoms of size at most m, define the regression thru a as

$$R(P,a) = \begin{cases} (P \backslash Add(a)) \cup Pre(a) & \text{if } P \cap Del(a) = \emptyset \\ undefined & \text{otherwise} \end{cases}$$

- This formula generalizes the h^2 regression; indeed, for $m = 2$ both regressions coincide!

h^m Equation

$$h^m(P;s) = \begin{cases} 0 & \text{if } P \subseteq s \\ \min_a \, cost(a) + h^m(R(P,a)) & \text{if } |P| \leq m \\ \max\{h^m(X;s) : X \subset P, |X| \leq m\} & \text{otherwise} \end{cases}$$

h^m Computation

- h^m computed using Dijkstra's seeded at, for $|P| \leq m$,
  1. $h^m(P; s) = 0$ if $P \subseteq s$
  2. $h^m(P; s) = \infty$ if $P \nsubseteq s$
- Up to our knowledge, only up to h^3 has been computed in real planners
- There exist m such that $h^m = h^*$
- The general computation of h^m is **exponential in** m

Higher-Order Mutexes 1/2

- A mutex of order m is a subset M of atoms, $|M| = m$, for which there is no reachable state s from s_0 that contains M
- The sets M such that $h^m(M; s_0) = \infty$ are mutex of order m
- A mutex M of order m may impose contraints on the simultaneous achievement of values for **different variables**
- Therefore, they can be used to improve the value of other heuristics such as Pattern Database heuristics

Higher-Order Mutexes 2/2

- For $m > 2$, it is possible that $h^m(\{p, q\}; s_0) = \infty$ whereas $h^2(\{p, q\}; s_0) < \infty$
- For example, consider atoms p, q, r, x, y such that the pairs $\{p, q\}$, $\{p, r\}$ and $\{q, r\}$ are reachable (non-mutex), $h^3(\{p, q, r\}; s_0) = \infty$ (mutex), and the action

$$a : Pre = \{p, q, r\}, Add = \{x, y\}, Del = \{\}$$

  Then, $R(\{x, y\}, a) = \{p, q, r\}$ and we have

  1. $h^2(\{x, y\}; s_0) < \infty$
  2. $h^3(\{x, y\}; s_0) = cost(a) + h^3(\{p, q, r\}; s_0) = \infty$

- Similar for higher-order mutexes and values of m

Conclusions

- h^m heurisitcs are powerful but expensive to compute
- Not an instance of delete-relaxation heuristics
- Can be used to boost other heuristics such as Pattern Database heuristics and even h^+

Part IV

The Context-Enhanced Additive Heuristic

The Context-Enhanced Additive Heuristic

h^{cea} extends the **causal graph heuristic** h^{CG} for SAS$^+$ domains (used in Fast Downward) by recasting it as a variation of h_{add}

h^{CG}

- Procedurally defined
- Certain problem structures must be simplified by removal of preconditions for computation

h^{cea}

- Mathematically defined
- Computable on all problem structures

h^{cea} vs. h^{CG}

- If h^{CG} is computable, $h^{CG}(s) = h^{cea}(s)$
- Otherwise, h^{cea} is expected to be more informative since no simplification required
- This is confirmed by empirical results

Notation

- x, x', x'' etc. are different values of the same multi-valued variable
- For a partial or complete variable assignment A, x_A is the value of x in A
- For a state s and a partial assignment P, $s[P]$ is identical to s except has the values that appear in P for all $var(P)$

h_{add} in SAS$^+$

For SAS$^+$ planning, h_{add} can be rewritten as follows:

$$h_{add}(s) \stackrel{\text{def}}{=} \sum_{x_g \in G} h_{add}(x_g | x_s)$$

$$h_{add}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min_{o:P \to x} c(o) + \sum_{y \in P} h_{add}(y|y_s) & \text{otherwise} \end{cases}$$

h_{add} in SAS$^+$

For SAS$^+$ planning, h_{add} can be rewritten as follows:

$$h_{add}(s) \stackrel{\text{def}}{=} \sum_{x_g \in G} h_{add}(x_g | x_s)$$

$$h_{add}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:P \to x} c(o) + \sum\limits_{y \in P} h_{add}(y|y_s) & \text{otherwise} \end{cases}$$

Cost of preconditions always evaluated from initial state

h^{cea}

$$h^{cea}(s) \stackrel{\text{def}}{=} \sum_{x_g \in G} h^{cea}(x_g|x_s)$$

$$h^{cea}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',P\rightarrow x} c(o) + h^{cea}(x''|x') + \sum\limits_{y\in P} h^{cea}(y|y_{s(x''|x')}) & \text{o.w.} \end{cases}$$

h^{cea}

$$h^{cea}(s) \stackrel{\text{def}}{=} \sum_{x_g \in G} h^{cea}(x_g | x_s)$$

$$h^{cea}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o: x'', P \to x} c(o) + h^{cea}(x''|x') + \sum\limits_{y \in P} h^{cea}(y | y_{s(x''|x')}) & \text{o.w.} \end{cases}$$

Intuition: Starting at x', x is achieved with o, where $x'' \in Pre(o)$:

$$x' \to \cdots \to x'' \xrightarrow{o} x$$

Achieve precondition x'' of o first, evaluate cost of $P = Pre(o) \setminus \{x''\}$ given the resulting **context** $s(x''|x')$

Contexts 1/2

$s(x''|x')$ is **projected state** after achieving x'' from x'

How to calculate $s(x''|x')$?

- Use the actions that result in the minimum values for the equation above:

$$x' \ldots \rightarrow x''' \xrightarrow{o'} x'' \xrightarrow{o} x$$

- Define $s(x''|x')$ *recursively*:

$$s(x''|x') \stackrel{\text{def}}{=} \begin{cases} s & \text{if } x'' = x' \\ s(x'''|x')[Pre(o')][Eff(o')] & \text{otherwise} \end{cases}$$

Context states and heuristic values are computed in parallel and are mutually dependent

Contexts 2/2

Ideally, use **full** contexts $s(x''|x')$, e.g. $h^{cea}(y|s(x''|x'))$

- **Problem:** Exponential number of context states

Idea: Information about other variables is discarded

- Approximate cost of precondition y from context state s' as $h^{cea}(y|y_{s'})$
- Information about other variables in s' is discarded

$$\ldots + \sum_{y \in P} h^{cea}(y|y_{s(x''|x')})$$

Example 1/4

Let $P = \langle V, O, I, G, c \rangle$ be an SAS$^+$problem with

- V $X = \{x_0, \ldots, x_n\}$
  $Y = \{true, false\}$
- O $a : \{\neg y\} \rightarrow \{y\}$ $b_i : \{y, x_i\} \rightarrow \{\neg y, x_{i+1}\}$
- I $\{x_0, y\}$
- G $\{x_n\}$
- c $c(a) = c(b_i) = 1$

The optimal plan is then

$$\pi^* = \langle b_0, a, \ldots, a, b_{n-1} \rangle$$

containing $n \times b_i + (n-1) \times a = 2n - 1$ actions

Example 2/4

$$x_0 \xrightarrow{b_0} \ldots \xrightarrow{b_{n-2}} x_{n-1} \xrightarrow{b_{n-1}} x_n$$

What is the value of $s(x_i|x_0)$?

- Base case: $s(x_0|x_0) = s = \{x_0, y\}$
- $s(x_1|x_0)$:

$$\begin{aligned} = \quad & s(x_0|x_0)[Pre(b_0)][Eff(b_0)] \\ & \{x_0, y\}[Pre(b_0)][Eff(b_0)] \\ & \{x_0, y\}[Eff(b_0)] \\ & \{x_1, \neg y\} \end{aligned}$$

- Recursive case: $s(x_i|x_0) = \{x_i, \neg y\}$

Example 3/4

What is the value of $h^{cea}(x_i|x_0)$?

$$
\begin{aligned}
h^{cea}(x_0|x_0) &= 0 \\
h^{cea}(x_1|x_0) &= c(b_0) + h^{cea}(x_0|x_0) + h^{cea}(y|y_{s(x_0|x_0)}) \\
&= 1 + 0 + 0 \\
h^{cea}(x_i|x_0) &= c(b_{i-1}) + h^{cea}(x_0|x_{i-1}) + h^{cea}(y|y_{s(x_{i-1}|x_0)}) \\
&= c(b_{i-1}) + h^{cea}(x_0|x_{i-1}) + h^{cea}(y|y_{\{x_0,\neg y\}}) \\
&= c(b_{i-1}) + h^{cea}(x_0|x_{i-1}) + h^{cea}(y|\neg y) \\
&= 1 + h^{cea}(x_0|x_{i-1}) + 1
\end{aligned}
$$

Since $s(x_{i-1}|x_0) = \{x_{i-1}, \neg y\}$, y evaluated from value $\neg y$

Example 4/4

We have:

$$\begin{aligned} h^{cea}(x_1|x_0) &= 1 \\ h^{cea}(x_n|x_0) &= h^{cea}(x_{n-1}|x_0) + 2 \end{aligned}$$

h^{cea} gives optimal solution to this problem:

$$\begin{aligned} h^{cea}(x_n|x_0) &= 2(n-1)+1 = 2n-1 \\ h^{cea}(s) &= h^*(s) \end{aligned}$$

Conclusions

- Context-enhanced heuristics generalize the concept of causal graph heuristic to problems with cyclic causal graphs
- Can be very informative in some cases in which h^+ is not
- Not comparable to delete relaxation heuristics