delete relaxation

ignore negative effects
assume no negative precondition

optimally solving the resulting problem is still NP-hard

solve it *quickly* rather than *optimally*

---

relaxed problem

assume no negative precondition or goals
otherwise, rewrite the problem

remove delete effects

plans remain plans
not the other way around

new plans created by relaxing

original problem:

```
initial: -x-y
a:  ⇒  x
b:  x  ⇒  -xy
goal:  xy
```

optimal plan: `a,b,a`
states: `-x-y, x-y, -xy, xy`
`x` made true, then false: needs to be made true again

remove negative effects:

```
initial: -x-y
a:  ⇒  x
b:  x  ⇒  y
goal:  xy
```

once true, `x` remains true
new optimal plan: `a,b`

plans remains plans
but new, shorter ones introduced
admissible heuristics

optimally solve the relaxed problem

finding a plan is easy
accumulate variables until goal reached

finding an optimal plan is NP-hard
why: minimal way to cover the goal variables

admissibility requires optimal plans for the relaxed problem
why: their length is a lower bound for the optimal plans of the original problem

solution: approximate length of the optimal plans

solve the relaxed problem

`x` true in the initial state
⇒ cost to obtain `x=true` is zero

`y` is false in the initial state
made true by action `a` of cost `1` and precondition `x`
⇒ cost to obtain `y=true` is `1`

`z` is false in the initial state
made true by action `c` of cost `6` and precondition `y`
⇒ cost to obtain `z=true` is `6+1`

etc.

in general:

cost for executing action = cost of preconditions + cost of action

example

variables x_1, x_2, x_3, x_4, x_5, x_6, x_7

actions:

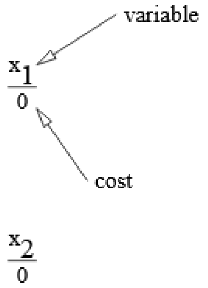
- a, cost 1, requires x_1, makes x_3 true
- b, cost 2, requires x_2, makes x_4 and x_5 true
- c, cost 4, requires x_3 and x_4, makes x_6 true
- d, cost 10, requires x_4 and x_5, makes x_7 true
- e, cost 3, requires x_6, makes x_8 true
- f, cost 1, requires either x_7, makes x_8 true

initially: x_1 and x_2 are true
goal: x_7 is true

problem already simplified (positive variables only)

[note] This description is given in full only for reference. In the following slides, the relevant parts are repeated when they are used.

initially



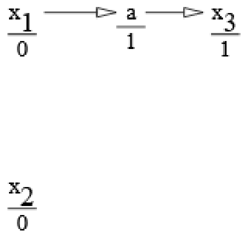
x_1 and x_2 initially true

making them true costs nothing

graphically: variable over cost of making it true

---

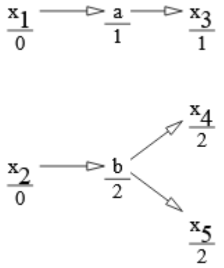
effect of action



cost of a is 1
effect is x_3=true

making x_3 true costs 1

multiple effects



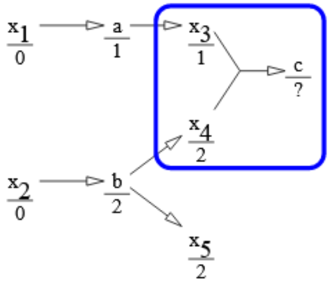
b costs 2
makes x_4 and x_5 true

making x_4 costs 2

same for x_5

---

multiple preconditions



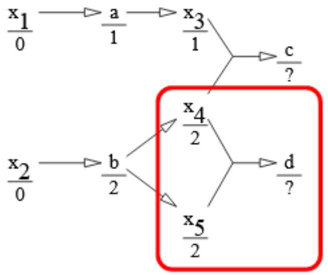
cost of c alone is 4

requires both x_3 and x_4 to be true
they cost 1 and 2

cost of executing c at this point is preconditions+action = $(1+2)+4$

look obvious, but…

multiple preconditions: a different case



d requires two variables to be true: x_4 and x_5

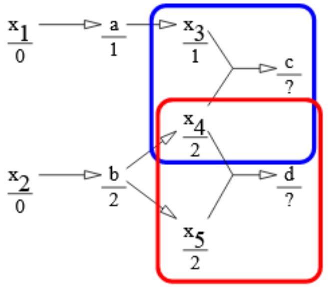
like in the previous case

but the actual cost is not $2+2$ as before

both variables generated by b at the same time:
cost is 2

---

multiple preconditions: the two cases

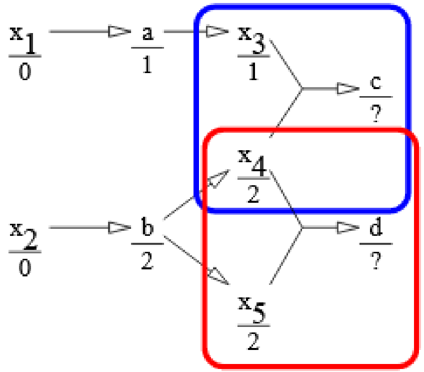


two preconditions may require two different actions
like x_3 and x_4, generated by a and b

or may be generated by the same action
like x_4 and x_5, generated both by b

cost of obtaining both: $1+2$ or 2?

how to combine the cost of preconditions



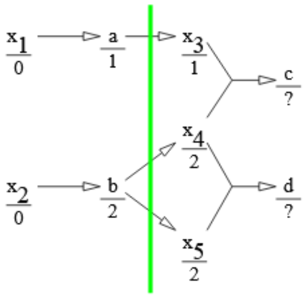
check which actions generates which variables is too costly

this case was simple, but preconditions may be generated in more complex ways
e.g., a caused x_2 and x_3, then b required x_3 and caused x_4 and x_5 and c required x_2, x_4 and x_5

heuristics will be computed many times during search
cannot spend too much time

[note] Removing negative effects and estimating the cost of reaching the goal is an heuristics. During the search it is calculated many time. Not much time can be spent on it.

do not look back



simplification:

cost of `c` is only function of cost of x_3 and x_4

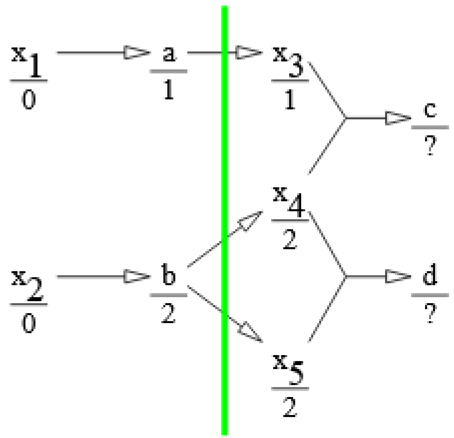
cost of `d` is only function of cost of x_4 and x_5

do not go back the vertical line

use only the cost of making each precondition true
disregard how it was obtained

[note] This simplification will of course result in an imprecision in the estimate of the cost of reaching the goal, but this is implicit because determining the optimal cost is NP-hard and the heuristics needs to be quick to determine.

optimistic and pessimistic attitude



do not go back the green line

pessimistic approach

the two preconditions are obtained by independent actions

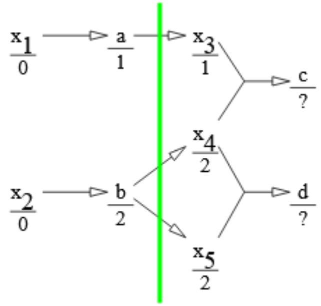
cost of making both true is the sum of making each true

optmistic approach

the two preconditions are obtained by the very same actions

cost of making both true is the maximal cost of making one of them true

pessimistic attitude



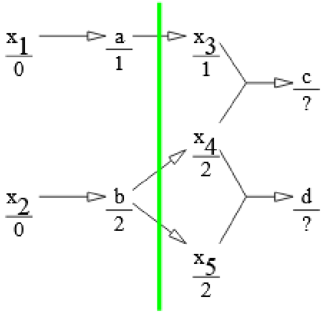
be pessimistic everywhere:

- cost of x_3 and x_4 is `1+2 = 3`
  add cost of `c`
- cost of x_4 and x_5 is `2+2 = 4`
  add cost of `d`

additive heuristics h^{add}

[note] This heuristics assumes that making two variables true can never be done with some common actions. The actions that make the first true and the actions that makes the second true are always different, with no action in common. Therefore, obtaining both variables can only be done by executing the actions that make the first variable true and the actions that make the second variable true.

optimistic attitude



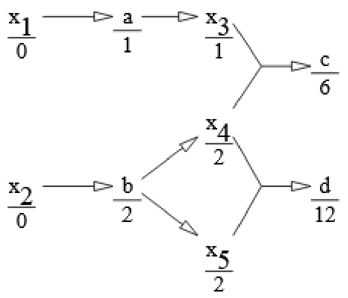
be optimistic everywhere:

- cost of x_3 and x_4 is `max(1,2)=2`
  add cost of `c`
- cost of x_4 and x_5 is `max(2,2)=2`
  add cost of `d`

maximum heuristics h^{max}

[note] This heuristics assumes that making two variables true can always be done with many common actions. In fact, it based on assuming that as many actions as possible contribute to making both variables true.

maximum heuristics, with cost of actions



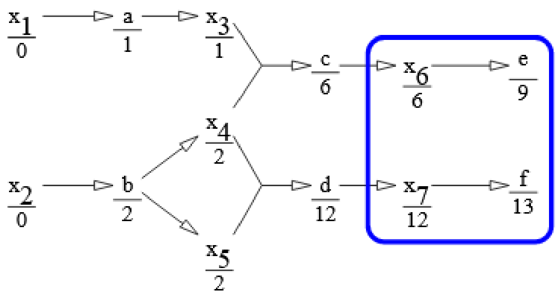
cost of action `c` alone: `4`
with preconditions: `4+max(1,2)=6`

cost of action `d` alone: `10`
with preconditions: `10+max(2,2)=12`

[note] As an example, this calculation is continued with the maximum heuristics, but the additive heuristics could have been used instead.

---

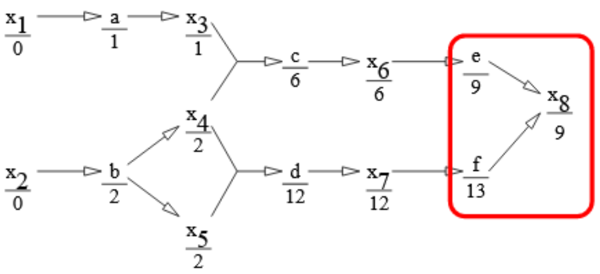
continue



x_6 makes `e` executable
x_7 makes `f` executable

add cost of actions: cost of `e` is `3` cost of `f` is `1`

alternative actions



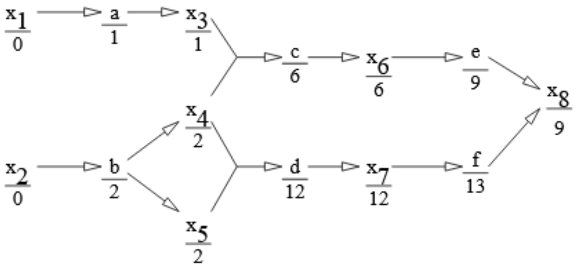
x_7 is made true by either `e` or `f`

cost is the *lowest* among them:
`min(9,13)=9`

[note] Contrarily to the optimistic/pessimistic policy, this is not a choice. The best way to make a variable true is always by executing the action that is cheapest in term of its overall cost (the cost of the action plus the cost of its preconditions).

---

goal



goal reached

the initial state was `{`x_1`=true, `x_2`=true}`

estimated cost of reaching the goal from the state `{`x_1`=true, `x_2`=true}` is `9`

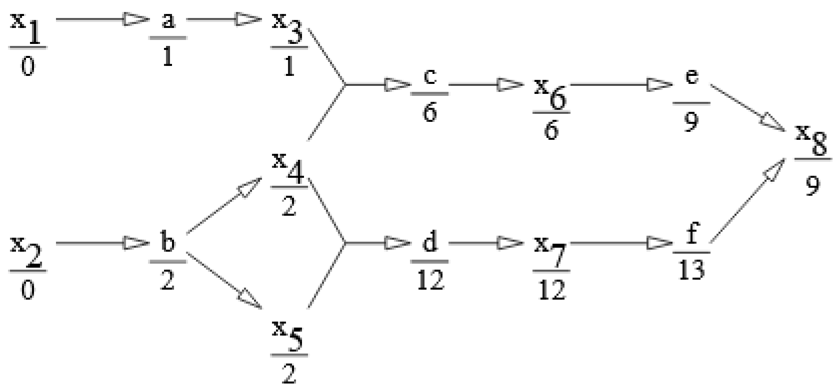
other states

estimated cost of reaching the goal from other states: same way

example: state `{x2=true, x5=true}`

build a similar graph with x_3 and x_5 as its first level

about the example



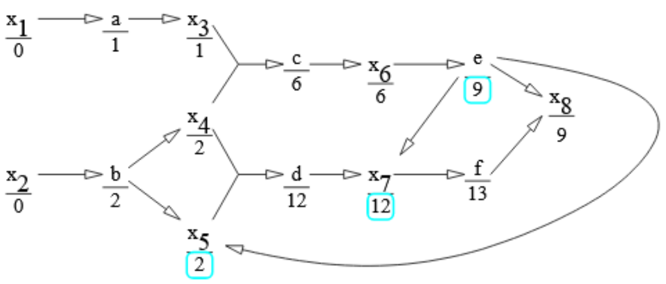
simple for the sake of explanation
variables and actions in levels
not so in general

actions may have preconditions from different levels
example: `f` requires both x_7 and x_3

may also make variables from previous levels true
example: `e` makes x_5 and x_7 true

first case easy
what to do in the second case?

diagonal effects

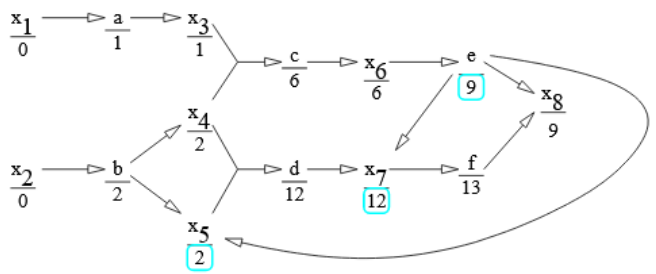


this is a different problem
previously; e made only x_8 true
now: e makes x_8, x_5 and x_7 true

keep into account these new effects

[note] The previous example did not contain such “diagonal” effects. It was built this way for the sake of the simplicity. This new example instead contains effects from a “line” to another, and also effects that go “backwards”.

cost updating



cost of executing e (including preconditions): 9

effects: x_8 (as before), x_5 and x_7

x_5

previously know: it can be achieved with cost 2
new way to achieve it (by action e) has cost 9
old way better: minimal cost 2

x_7

previously known: it can be achieved with cost 12
new way to achieve it (by action e) has cost 9
new way better: minimal cost 9

note: cost of x_7 changed, update f as well!

[note] This is a different problem, where action e has also effects x_5 and x_7.

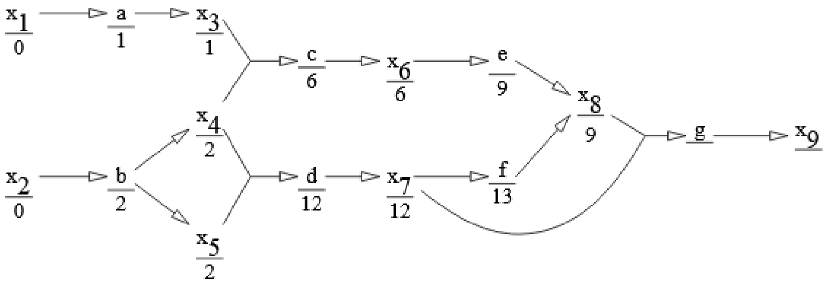
Before considering the consequences of e, the cost of achieving x_5 was 2. This means that x_5 can be obtained either the old way, with cost 2, or as a consequence of e, with cost 9. The old way is cheaper than the new, so the cost of x_5 is not changed.

Before expanding the consequences of e, the cost of achieving x_7 was 12. This variable is now found out to be obtainable as a consequence of e, with cost 9. The new way is cheaper than the old, so the cost of x_7 is lowered to 9.

The cost of f has to be updated as well. Previously, its cost was the sum of the cost of x_7 (12) plus the cost of the action alone (1). Since the cost of x_7 changed, the sum has to be recomputed: it is the new cost of x_7 (9) plus the cost of the action alone (1). The new cost of f is therefore 10.

This mechanism is similar to the reopening of nodes in search algorithms such as A*: when reaching a variable in a new way, the new path may be cheaper than the old or not; in the former case, the cost of the variable and its succesors is lowered. It is done in the graph of variables/action dependencies instead of the search space, in a manner similar to Dijstra algorithm.

multiple goals



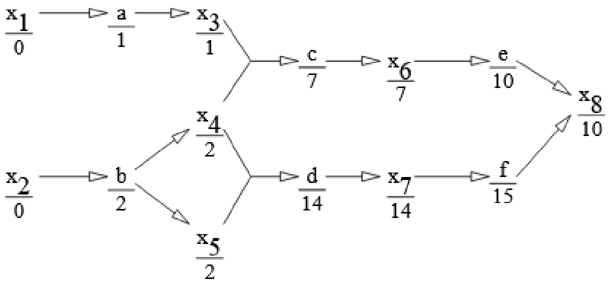
example: goal was x_7 and x_8

add an action with two preconditions x_7 and x_8 and zero cost

new variable as effect
or just use the cost of executing the action (including preconditions) as the value of the heuristic

---

additive heuristics

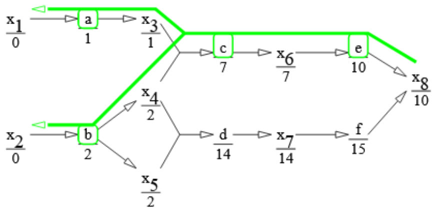


same progression
(variables - actions - variables - …)

cost of action = cost of action alone + sum of cost of preconditions

example: cost of `c` is cost of x_3 (1) plus cost of x_4 (2) plus cost of `c` alone (4)
total: 7

ff heuristics



use the additive heuristics

start from the goal
go back selecting the actions actually used to reach the goal

in x_7 select cheapest action: `e`

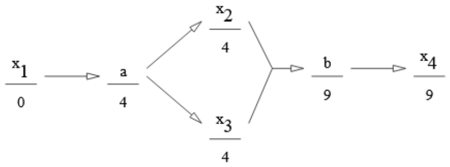
in `c`, both preconditions need to be true

sum cost of selected actions: `1+2+4+1 = 8`
cost of actions *alone*
without the cost of their preconditions

[note] In this case ff gives the same result as add. The next example shows that ff may be more accurate than ff.

---

the trouble with add



a different example
cost of `a` is 4, cost of `b` is 1

sum gives 9 = 1 + 4 + 4

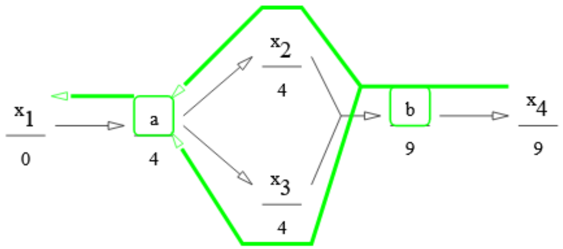
where do these numbers come from?

- 1: cost of `b` alone
- 4: cost of x_2
  is a consequence of `a`
  and `a` costs `4`
- 4: cost of x_3
  is a consequence of `a`
  and `a` costs `4`

total is: cost of `b` + cost of `a` + cost of `a`

`a` is counted twice

why ff instead of add



same as add when going forward

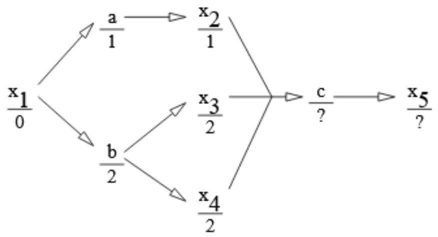
but then, goes back and collect actions used

total cost of `a` and `b`: 4+1 = 5

actions are never counted more than once

---

add, maximum and ff differ from each other



cost of executing `c` alone is `0`
when counting preconditions:

maximum
  cost of executing `c` is `max(1,2,3) = 2`
  underestimate: both `a` and `b` are required
add
  cost of executing `c` is `1+2+2 = 5`
  overestimate: is like `b` were executed twice
ff
  going back from x_5: actions required are `a`, `b` and `c`
  correct (in this case): actual cost `2+1+0=3`

ff is more precise than add, in practice better than maximum
still an estimate of the cost, not the exact value

[note] This is another problem, wiith different actions, variables and goal. Is used to show that the three heuristics may give three different results on the same problem.

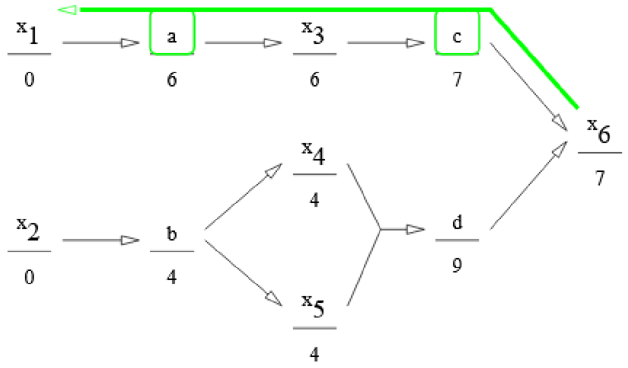
admissibility

maximal:
cost of action = maximal cost of a precondition
correct: each precondition needs to be achieved
admissible

sum and FF:
cost of action = sum cost of preconditinos
imprecise: some actions may contribute to more than one precondition
in such cases, cost is overestimated
not admissible

---

add and ff are not admissible



cost of `a` is `6`
cost of `b` is `4`
cost of `c` and `d` is `1`

add and ff: cost of action = sum of cost of preconditions
ff: go back from the goal and sum cost of actions

both add and ff evalaute cost as `7`
same as plan `a;c`

actual cost is `5`
optimal plan: `b;d`

[note] This is a different problem where ff and add both evaluate the cost of reaching the goal as `7`, while the actual cost was `5`. This means that neither heuristics is admissible.

non-admissible heuristics

non admissible ≠ unusable

example: a problem where h_{ff} is not admissible but never off by more than 1%

compare with $h_0(s)=0$ for all states s

h_0 admissible

- A* has the optimal plan when it first reaches the goal
- but: same as Dijstra, large frontier

h_{ff} not admissible

- first plan may not be optimal
- but: goal reached quickly; optimal plan found soon afterwards

[note] This is a possible scenario, where h_{ff} is assumed to be accurate on a particular problem. This is of course something that cannot be guaranteed in all cases. Yet, it shows that an accurate but non-admissible heuristics may be better than an inaccuarate but admissible heuristics.

---

non-boolean problems

delete effect = variable made false
defined only if variables are true/false

otherwise: map `x=value` into true/false
ignore delete effects: `x=value` never becomes false
like `x` accumulates all values it had