abstraction heuristics

pattern database: remove some variables
in the search space: group states

states differing only on the removed variables are made one

do better groups exist?
can be determined efficiently (time/space)?

a challenge for pattern database

```
a: ¬x¬y¬z ⇒ x
b: ¬x¬y¬z ⇒ y
c: ¬x¬y¬z ⇒ z
d: x¬y¬z ⇒ w
e: ¬xy¬z ⇒ w
f: ¬x¬yz ⇒ w
goal: w
```

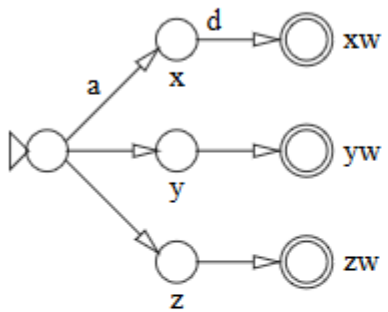
state ¬x¬y¬z¬w

actions a,b,c make one variable among x,y,z true
actions d,e,f make the goal true if one variable is true

compute heuristics for state ¬x¬y¬z¬w

[note] The nature of this planning is much clearer when seen in the search space (next slide), rather than from the definition of the actions.

looking at the search space

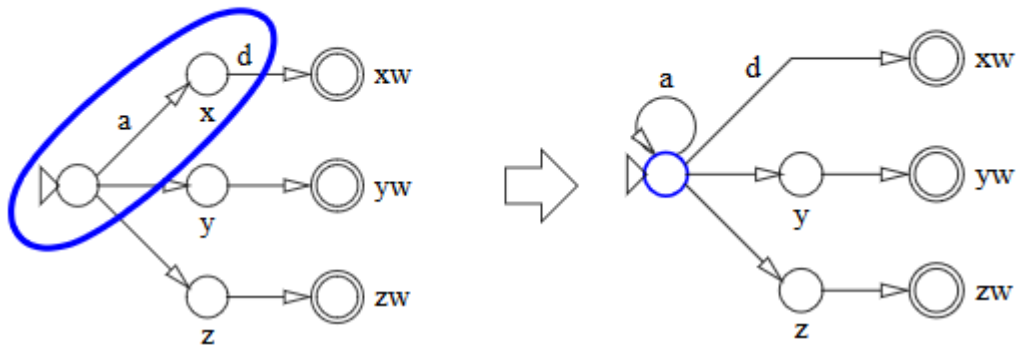


an optimal plan is `a;d`
length `2`

what happens when removing variables?

[note] This diagram is simplified by omitting the variables that are false in the states. Some actions have also been omitted.

remove x

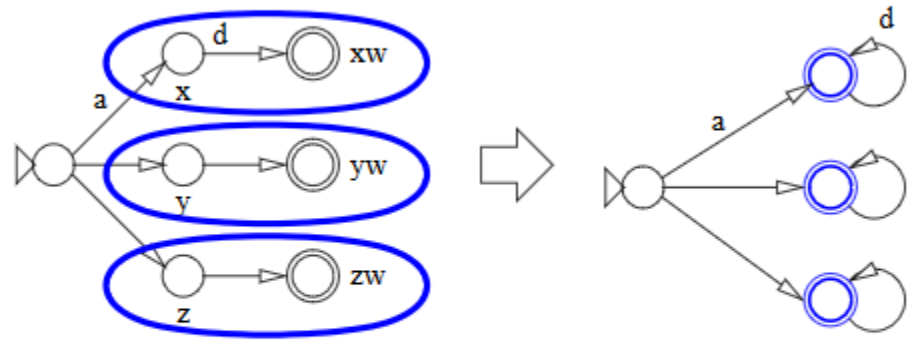


group states differing only on x
in this case: states blue oval
make them one

optimal plan: d alone
h_{yzw}(-x-y-z-w) = 1

same for deleting y alone or z alone

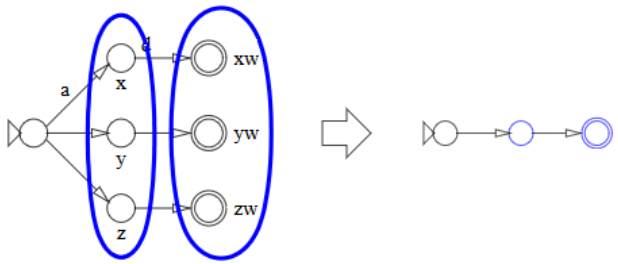
remove w



each blue oval becomes a single state
why: for example, x and xw differ only on w

an optimal plan is a alone
h_{xyz}(-x-y-z-w) = 1

an optimal grouping



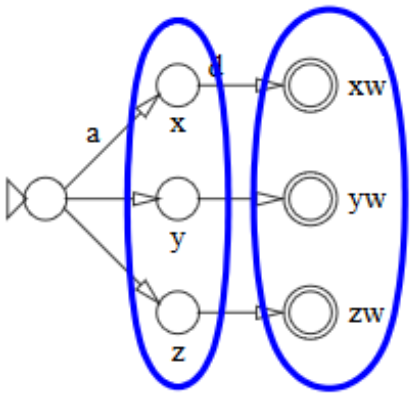
another way of grouping states
each blue oval becomes a single state

optimal heuristics:
a shortest plan is `a,d`
length 2

these groups cannot be obtained by removing some variables

[note] In this example, the optimal plan from the initial state has length 2, but removing `x` alone shortens the optional plan to 1, and the same for `y` alone, `z` alone or `w` alone. Instead, the groups in the figure leads to optimal plans of length 2.

how to group states?

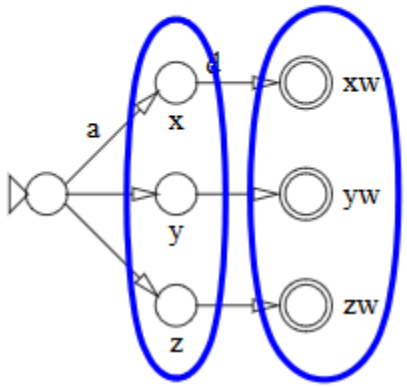


groups induced by variable removal may give a poor estimate

better estimates are obtained by other groups of states

how to obtain good groups?

exponentiality



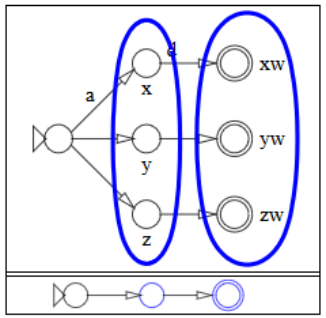
this example: seven states

in general: the number of states is exponential in the number of variables

analyzing the full search space for choosing the groups takes too long

solution: work on *abstract search spaces*
definition: next slide

definition: abstract search space



search space + some groups of states = abstract search space

each group becomes one state

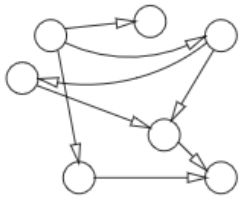
[note] For example, the search space of a problem after deleting some variables is an abstract search space. But in general the groups can be arbitrary. More precisely, the groups can be arbitrary non-overlapping sets of states.

merge and shrink

principles:

- always work on abstract search spaces that are small enough to store and analyze
  abstract search space = each group of state is replaced by a single state
- merge groups of states to reduce size

merge and shrink: concrete search state

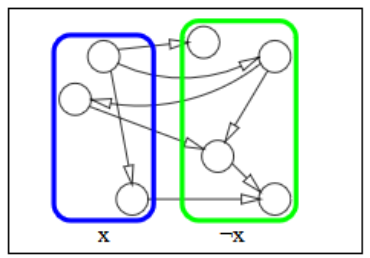


just the search space of the instance

in general: too large to inspect for good groups

[note] This is the concrete search space used in the next examples. This one has been chosen small for the sake of clarity, but in general the concrete search space may be too large to be represented in full.

merge and shrink: a single variable abstraction



concrete search space
with two groups of states

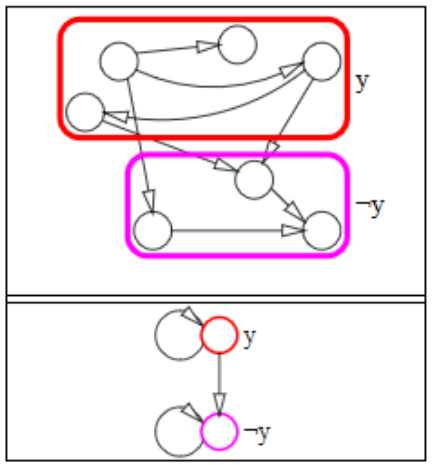
abstract search space

remove all variables but x

equivalently: two groups
first group: all states in which x is true
second group: all states in which x is false

two states in the abstract search space

merge and shrink: another single variable abstraction

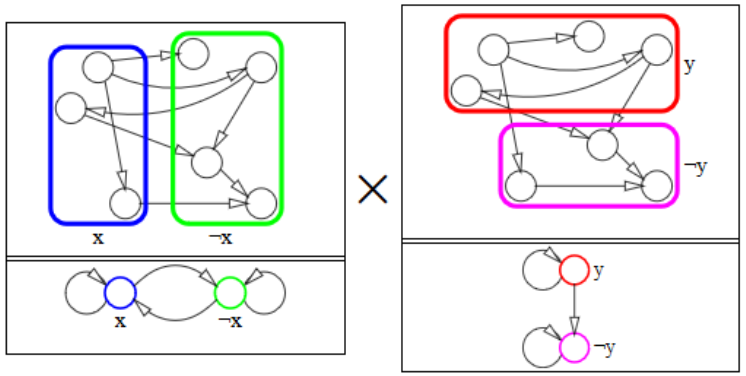


abstract search space for y

no arrow from $\neg y$ to y in this case

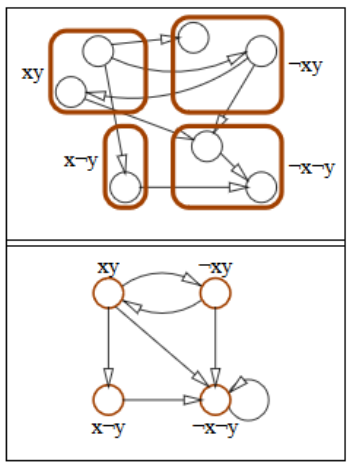
[note] Whether or not an arrow goes from a state to another in the abstract search space depends on the actual actions in the domain. This is detailed in a following slide.

merge and shrink: join two single-variable abstractions



×

=



"multiply" the two single-state abstractions

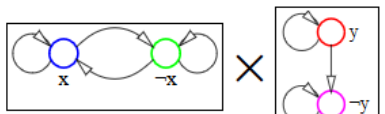
like splitting the `x` state in two: `xy` and `x¬y`
same for the `¬x` state

how to do it?
never look at the concrete search space: too big

[note] In the figures, each abstract search space is shown in two forms: above is the the concrete search space with the groups of states, below is the search space obtained by making each group a single state.

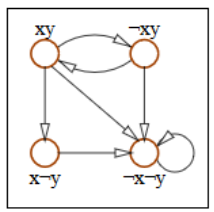
The abstract search space is the second only, the first is shown only for illustration. Joining is performed using only the abstract search space, not the concrete search space with the groups.

how to join two single-variable abstractions



×

=



the x state is split in xy and x¬y
same for ¬x

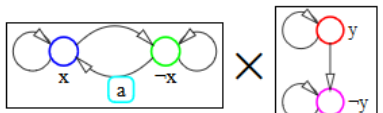
work only on the abstractions
not on the concrete search spaces with groups

omitted in the figure:
arrows are labeled with actions
including loops

used to determine arrows in the result

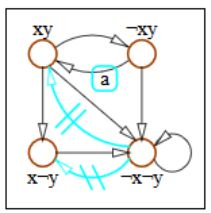
[note] This is the same figure as before, but only the abstract search spaces are shown, not the corresponding concrete search spaces with groups. The latter were only shown for clarity, but the algorithm can only work on abstract search spaces since the concrete search space is of exponential size.

where to place the arrows



×

=



example: action `a: ¬x,y ⇒ x`
on the arrow from `¬x` to `x`

`a` turns `¬xy` into `xy`
⇒ place arrow with `a` from `¬xy` to `xy`

`a` does not turn `¬x¬y` into `x¬y`
⇒ do not place arrow

`a` does not turn `¬x¬y` into `xy`
⇒ do not place arrow

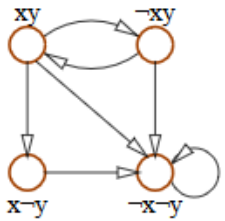
do the same for all actions and all pairs of states

[note] The concrete search space is too large for an exhaustive analysis. Therefore, all operations have to be performed in the abstract search spaces.

The starting points are the two single-variable search spaces, which are obtained from the original planning problem by removing all variables but one. This search space comprises two states only for binary variables, or a number of states equal to the possible values for the variables. Arrows are labeled with the actions that move from a state to another.

When joining two search states, the result has a state for each pair of states of them. The actions on the arrows of the search spaces that are joined are used to determine which arrows and label are on the resulting search space.

merge and shrink: join again



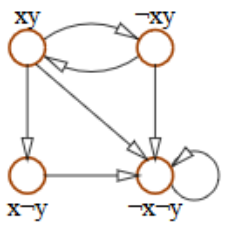
so far: joined single-variable abstractions for `x` and `y`
same as the pattern database for `xy`

join the result with the single-variable abstraction for `z`
result: same as the pattern database for `xyz`

or, instead…

---

merge and shrink: so what?



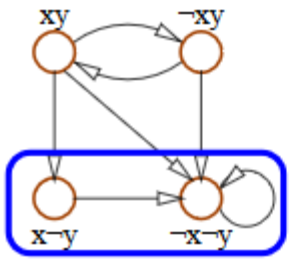
result so far could have been obtained by deleting all variables but `x` and `y`

so what?

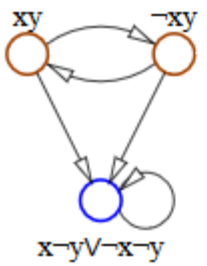
search space is *small*

can be inspected for states to be merged

merge and shrink: shrink



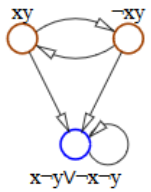
group two states



this reduces the size of the current search space

then…

merge and shrink: after shrinking



then proceed:
join this with the single-variable abstraction for `z`

result is different than removing all variables but `xyz`

[note] Without shrinking, joining single-variable abstractions would be pointless: the result of joining the abstractions for `x` and `y` would be the same as removing all variables but these two from the original instance.

Shrinking is necessary to reduce the search space. The resulting set of states may not be obtainable by simply removing some variables.

merge and shrink: the bound

- start with a single-variable abstraction
  a very small search space
- join with another single-variable abstraction
  enlarges the search space
- continue until the search space grows too large
- when it's becoming too big:
  group states
  then continue joining with single-variable abstractions

which states to merge?

merge and shrink: the improvement

merge and shrink may obtain better abstractions than pattern database

how: instead of removing altogether some variables,
make arbitrary groups of states in the current abstraction

better = ?

---

merge and shrink: the aim

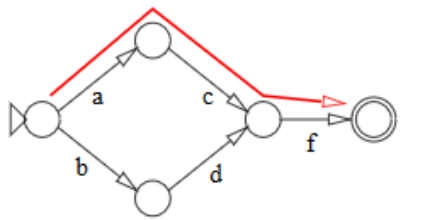
obtain a better heuristics
admissible, so larger is better

estimate = length of optimal plans

optiomal plans in the abstraction are better **longer** than shorter!

group states trying not to shorten plans

abstractions and plans



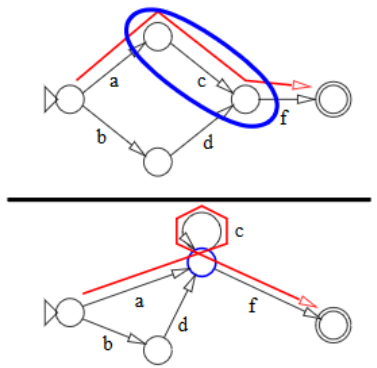
some states grouped

what happens to plans?

example: plan `a,c,f`

[note] The consideration made in this and the following slides are about abstractions in general. They are particularily relevant to grouping states in the merge-and-shrink method.

still a plan



plan `a,c,f` is still a plan

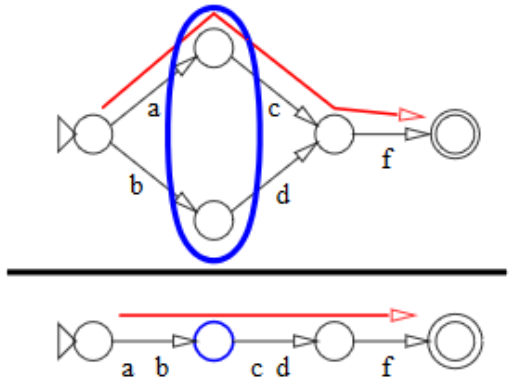
same actions can be executed in the abstraction
same effect

but: loop avoidable
shorter plan `a,f`

[note] The search space over the line is the original one. Below is the result of abstracting the two states in the group.

In the context of estimating the plan length, introducing a shorter plan is a bad thing. The abstraction has a plan of length 2 while the actual problem has only plans of length 3. This means that the abstracion underestimates the length of the plans.

same length



different grouping

`a,c,f` still a plan

with this grouping, also optimal
no shorter plans

[note] This is a different abstraction for the same problem. It is better than the previous one, since it estimates the length of the optimal plan exactly.

comparison of heuristics

admissible = never overestimates the length of optimal plans

if admissible, larger is better (more accurate)

longer plans in the abstraction = better heuristics

---

how shrinking affects the heuristics

shrink: group of states ⇒ single state
always admissible

try to shrink so that plans are not shortened
results in more accurate estimates

merge and shrink: summary

start with a single-variable abstraction

alternate:

merge
    join the current abstract search space with another single-variable abstraction
shrink
    if the current abstract search space is becoming too big, group some states
    choose groups that keep plans long

this way:
abstraction may contain all variables
size of abstract search space is bounded
built trying to obtain large estimates of plan length

alternative

method above
    join the current abstract search space with another single-variable search space
alternative: balanced tree
    join `x` with `y`
    then `z` with `w`
    join the two results

---

why "abstraction"?

why the name "abstraction" heuristics?

set of states ⇒ state

like an abstract concept represents a set of concrete objects