planning by forward search

start with an empty policy (no action assigned to states)

expand one state at time

empty policy, expanding a policy

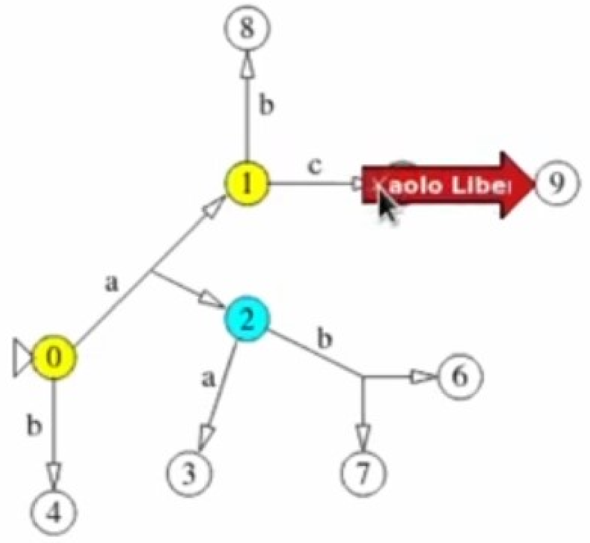
empty policy: `{}`

try an action for the initial state: `{(0,a)}`
repeat for the states resulting from executing `a` in `0`

try another action for the initial state: `{(0,b)}`

try yet another, etc.

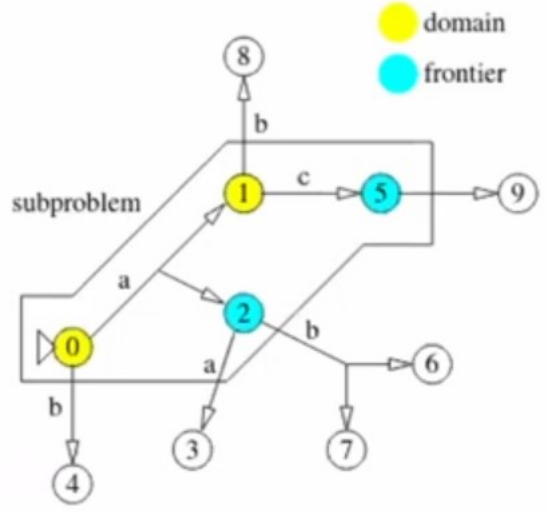
partially expanded policy



decisions taken so far:
in `0` execute `a`
in `1` execute `c`

policy: `{(0,a), (1,c)}`

domain, subproblem, frontier



yellow nodes 0 and 1

policy has decided what to do here

domain of the policy

polygon

states and actions in the policy, including resulting states

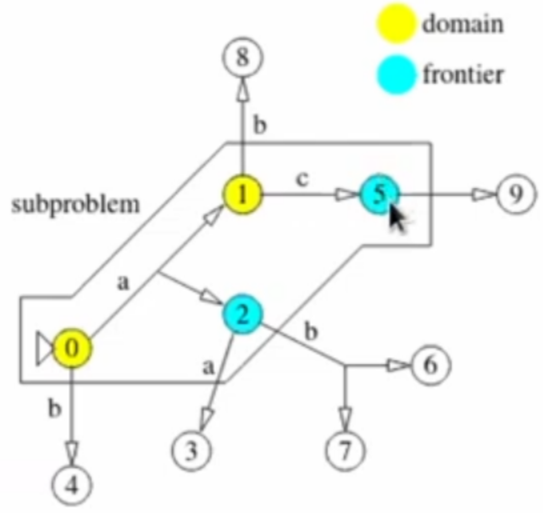
subproblem

cyan nodes 2 and 5

policy arrived there, but has not yet decided what to do there

frontier

where to expand a policy

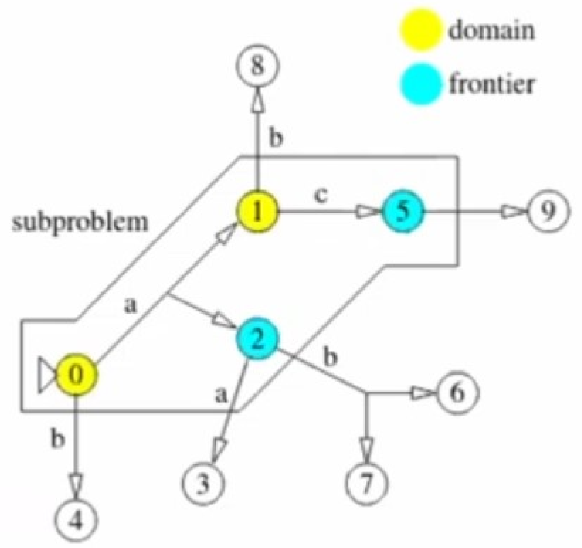


states in the domain (yellow): action to execute already decided

states outside the subproblem (white): still not known if policy will ever arrive there

remain: states in the frontier (cyan)

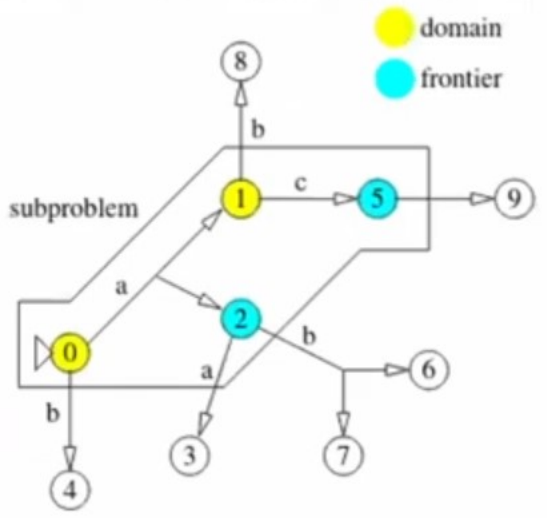
expanding a policy

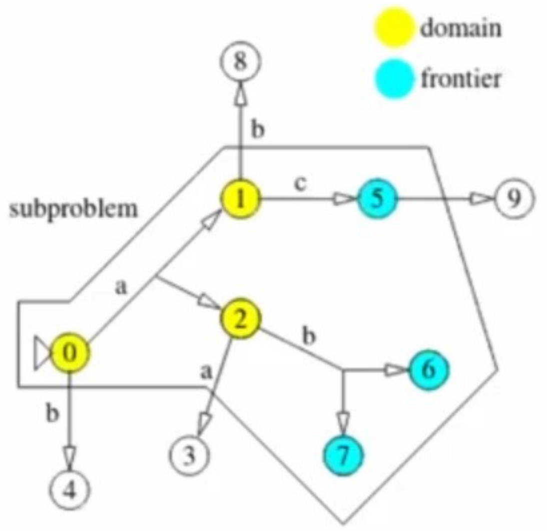


choose a node in the frontier, for example `2`
choose an action, in this case either `a` or `b`
for example, `b`

expanded policy

choose `b` as the action to execute in `2`:





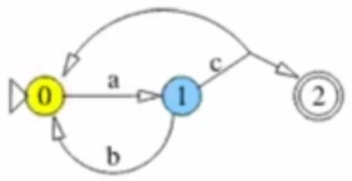
old policy: `{(0,a),(1,c)}`
new policy: `{(0,a),(1,c),(2,b)}`

when to stop?

the frontier includes a goal state
 weak solution
the frontier is made of all goal states
 strong solution, **maybe**

not all cycles are made equal

example of a partially expanded policy:



what to do in `0` is already decided (do `a`)
what to do in `1`?

do `b`
 the cycle can never be escaped
 $0 \to_a 1 \to_b 0 \to_a 1 \to_b 0 \to \dots$
do `c`
 the cycle may lead back or not
 for example:
 $0 \to_a 1 \to_c 0 \to_a 1 \to_c 2$ [goal]

what to do on cycles

overall algorihtm:

- choose an action for a frontier state
- expand the policy
- repeat

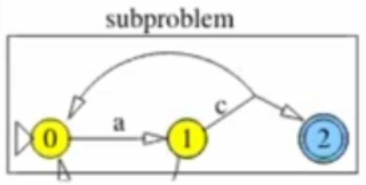
if the expansion makes an unescapable cycle enters the policy, choose another action
if no more actions to try, backtrack
if searching for a strong policy: do it also for escapable cycles

<-- -->

```
expand(policy) {
   choose a node N in the frontier
   for every action A executable in N {
      if (expand(policy + {N,A}) == found)
         return found;
   }
   return notfound;
}
```

cycles in the problem and cycles in the subproblem

do not worry about cycles in the problem
even if they are only made of states in the domain



the cycle `a-b-a-b-…` is irrelevant
not in the **subproblem**

the cycle `a-c-a-c-…` is relevant
is in the subproblem
the frontier `{3}` is made only of goal states, but the solution is only strong, not strong acyclic