Principles of AI Planning

Strong cyclic planning with full observability

Malte Helmert   Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

January 24th, 2007

Planning objectives
Strong plans

- The simplest objective for nondeterministic planning is the one we have considered in the previous lecture: reach a goal state with certainty.
- With this objective the nondeterminism can also be understood as an opponent like in 2-player games or in n-player games in general.
  The plan guarantees reaching a goal state no matter what the opponent does: plans are winning strategies.

Planning objectives
Limitations of strong plans

- In strong plans, goal states can be reached without visiting any state twice.
- This property guarantees that the length of executions is bounded by some constant (which is smaller than the number of states.)
- Some solvable problems are not solvable this way.
  1. Action may fail to have any effect.
     Hit a coconut to break it.
  2. Action may fail and take us away from the goals.
     Build a house of cards.

  Consequences:

  1. It is impossible to avoid visiting some states several times.
  2. There is no finite upper bound on execution length.

- In strong plans, goal states can be reached without visiting any state twice.
- This property guarantees that the length of executions is bounded by some constant (which is smaller than the number of states.)
- Some solvable problems are not solvable this way.
  1. Action may fail to have any effect.
     Hit a coconut to break it.
  2. Action may fail and take us away from the goals.
     Build a house of cards.

Consequences:

1. It is impossible to avoid visiting some states several times.
2. There is no finite upper bound on execution length.

Planning objectives
Limitations of strong plans

- In strong plans, goal states can be reached without visiting any state twice.
- This property guarantees that the length of executions is bounded by some constant (which is smaller than the number of states.)
- Some solvable problems are not solvable this way.
  1. Action may fail to have any effect.
     Hit a coconut to break it.
  2. Action may fail and take us away from the goals.
     Build a house of cards.

  Consequences:
  1. It is impossible to avoid visiting some states several times.
  2. There is no finite upper bound on execution length.

Assumption

For any nondeterministic effect $e_1 \mid \dots \mid e_n$ the probability of every effect $e_1, \dots, e_n$ is greater than 0.

Alternatively: For any $s' \in \mathit{img}_o(s)$ the probability of reaching s' from s by o is greater than 0.

This assumption guarantees that a strong cyclic plan reaches the goal almost certainly (with probability 1).

This is not compatible with viewing nondeterminism as an opponent in a 2-player game: the opponent's strategy might rule out some of the choices $e_1, \dots, e_n$.

Planning objectives

When strong cyclic plans make sense

Assumption

For any nondeterministic effect $e_1 \,|\, \ldots \,|\, e_n$ the probability of every effect $e_1, \ldots, e_n$ is greater than 0.

Alternatively: For any $s' \in \mathit{img}_o(s)$ the probability of reaching s' from s by o is greater than 0.

This assumption guarantees that a strong cyclic plan reaches the goal almost certainly (with probability 1).

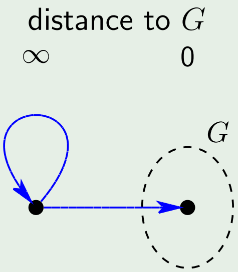
This is not compatible with viewing nondeterminism as an opponent in a 2-player game: the opponent's strategy might rule out some of the choices $e_1, \ldots, e_n$.

Need for strong cyclic plans
Example

Example (Breaking a coconut)

- Initial state: coconut is intact.
- Goal state: coconut is broken.
- On every hit the coconut may or may not break.
- There is no finite upper bound on the number of hits.

This is equivalent to coin tossing.



Need for strong cyclic plans

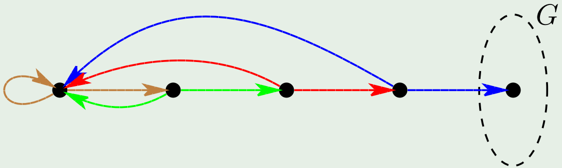
Example

Example (Build a house of cards)

- Initial state: all cards lie on the table.
- Goal state: house of cards is complete.
- At every construction step the house may collapse.

distance to G

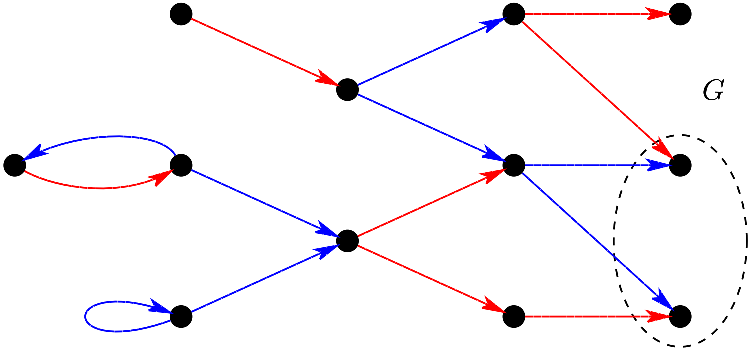
∞ 0



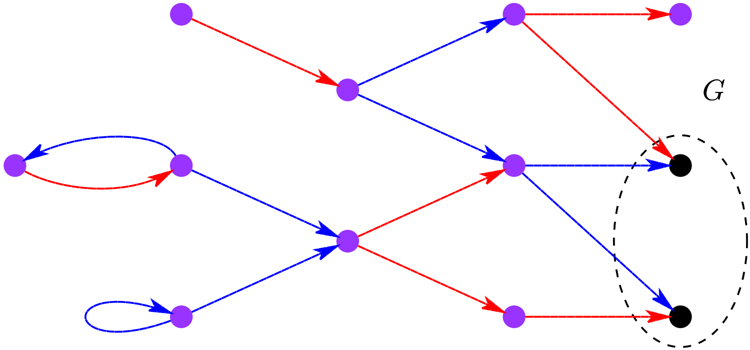
Strong cyclic planning algorithm
Idea

- We now present an algorithm that finds plans that may loop (strong cyclic plans).
- The algorithm is rather tricky in comparison to the algorithm for strong plans.
- Every state covered by a plan satisfies two properties:
  1. The state is good: there is at least one execution (= path in the graph defined by the plan) leading to a goal state.
  2. Every successor state is either a goal state or good.
- The algorithm repeatedly eliminates states that are not good.

Strong cyclic planning algorithm
Example

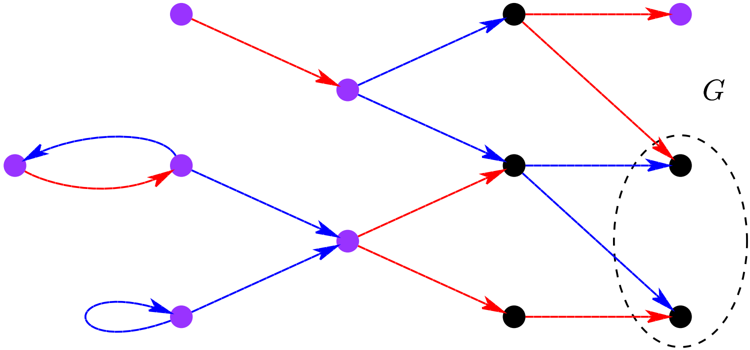


All states are candidates for being good.



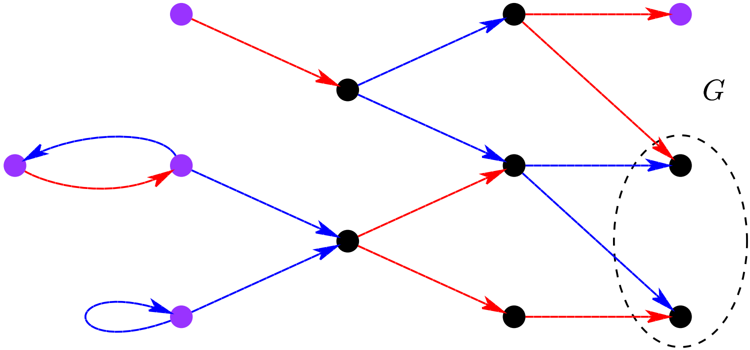
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 1 steps so that all immediate successors are possibly good.



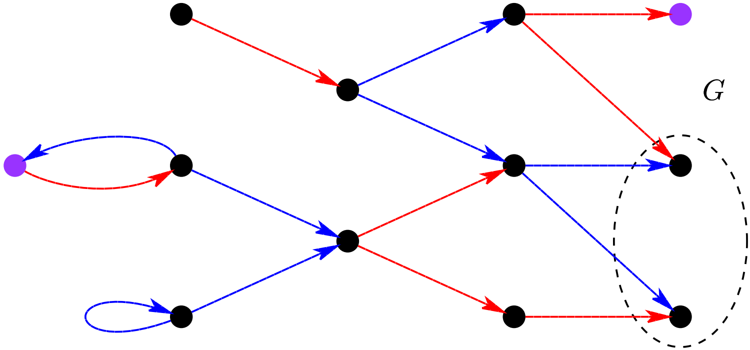
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 2 steps so that all immediate successors are possibly good.



Strong cyclic planning algorithm
Example

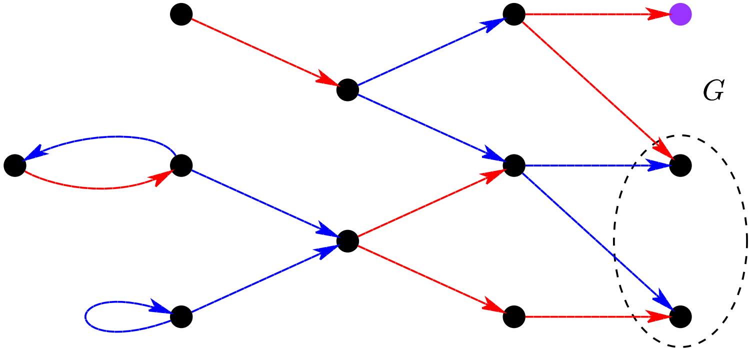
States from which goals are reachable in ≤ 3 steps so that all immediate successors are possibly good.



Strong cyclic planning algorithm

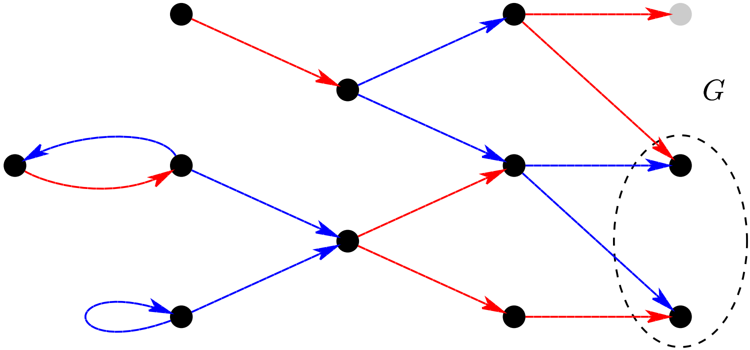
Example

States from which goals are reachable in ≤ 4 steps so that all immediate successors are possibly good.



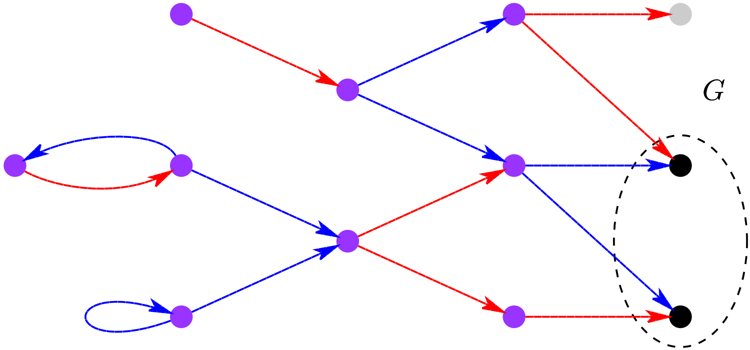
Strong cyclic planning algorithm
Example

Eliminate states that turned out not to be good.



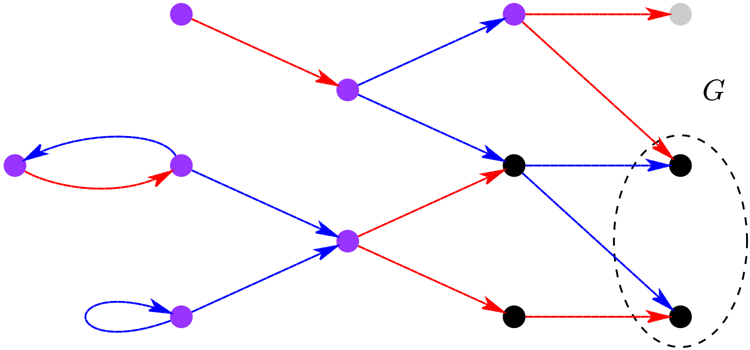
Strong cyclic planning algorithm
Example

The set of possibly good states is now smaller.



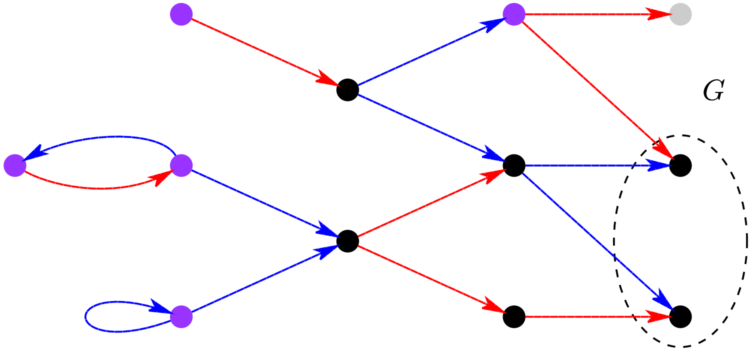
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 1 steps so that all immediate successors are possibly good.



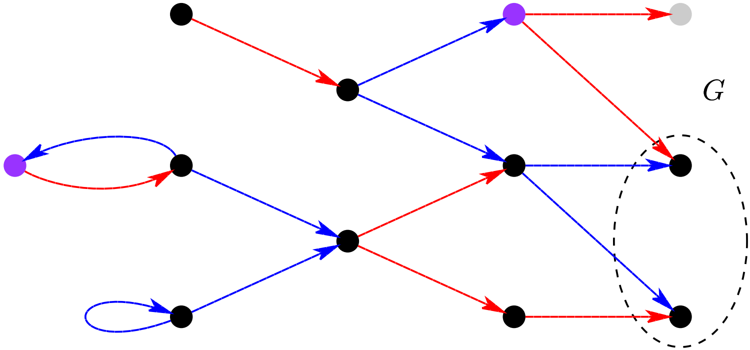
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 2 steps so that all immediate successors are possibly good.

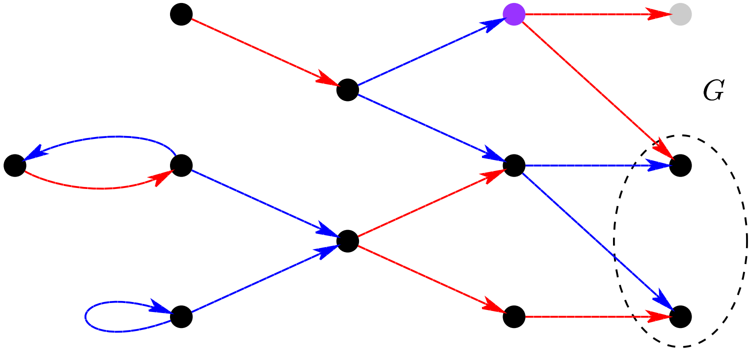


Strong cyclic planning algorithm
Example

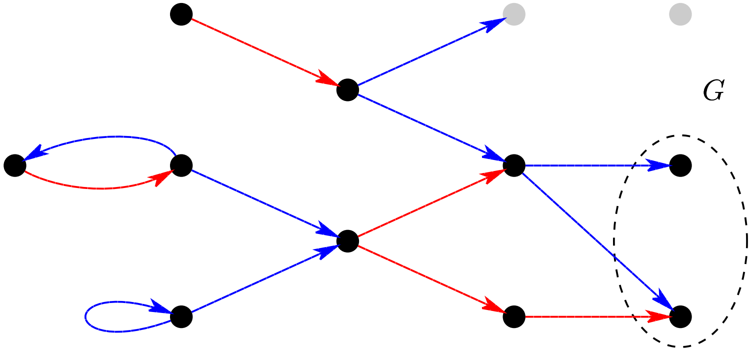
States from which goals are reachable in ≤ 3 steps so that all immediate successors are possibly good.



States from which goals are reachable in ≤ 4 steps so that all immediate successors are possibly good.

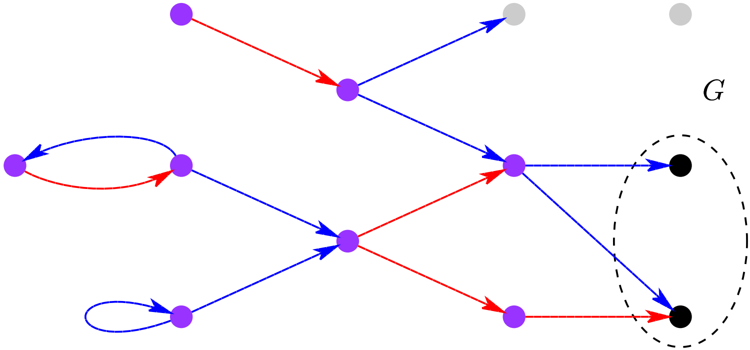


Eliminate states that turned out not to be good.



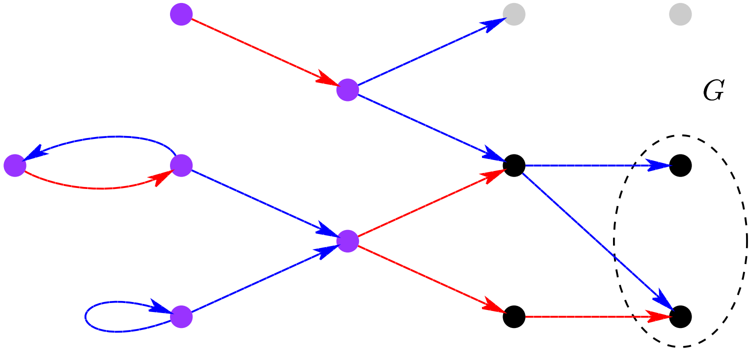
Strong cyclic planning algorithm
Example

The set of possibly good states is now smaller.



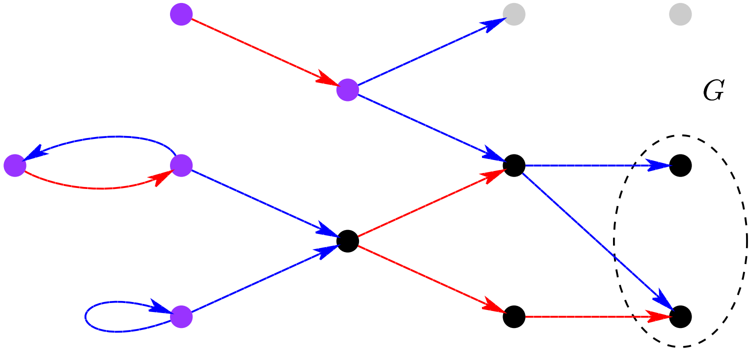
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 1 steps so that all immediate successors are possibly good.

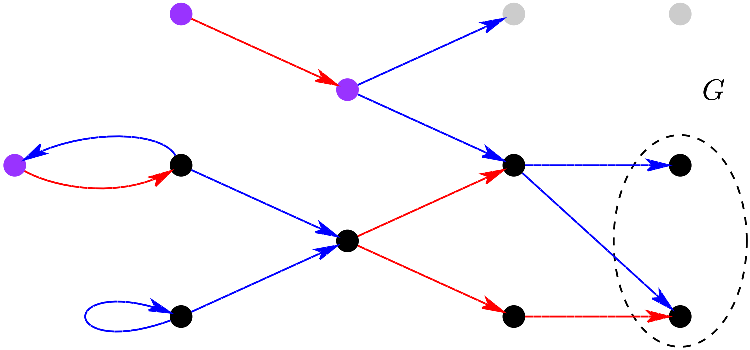


Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 2 steps so that all immediate successors are possibly good.

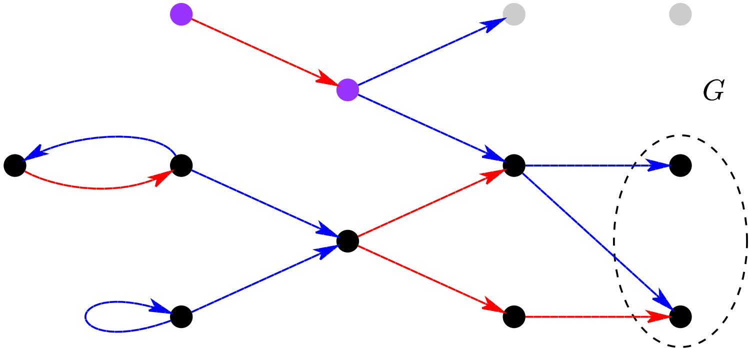


States from which goals are reachable in ≤ 3 steps so that all immediate successors are possibly good.



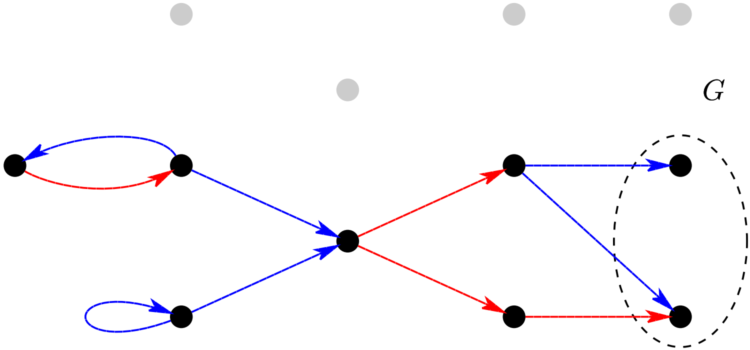
Strong cyclic planning algorithm
Example

States from which goals are reachable in ≤ 4 steps so that all immediate successors are possibly good.



Remaining states are all good.

A further iteration would not eliminate more states.

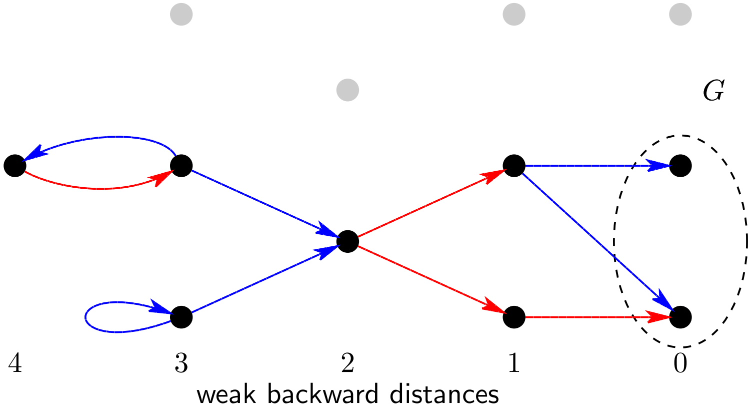


Strong cyclic planning algorithm
Example

Assign each state an operator so that the successor states are goal states or good, and some of them are closer to goal states.

Use weak distances computed with weak preimages.

For this example this is trivial.



Procedure *prune*

- The procedure prune finds a maximal set of states for which reaching goals with looping is possible.
- It consists of two nested loops:
  1. The outer loop iterates through $i = 0, 1, 2, \ldots$ and produces a shrinking sequence of candidate good state sets $C_0, C_1, \ldots, C_n$ until $C_i = C_{i+1}$.
  2. The inner loop identifies growing sets W_j of states from which a goal state can be reached with j steps without leaving the current set of candidate good states C_i. The union of all $W_0, W_1, \ldots$ will be C_{i+1}.

Procedure *prune*
Definition

Procedure prune

def prune(S, O, G):
    $C_0 := S$
    for each $i \in \mathbb{N}_1$:
        $W_0 := G$
        for each $j \in \mathbb{N}_1$:
            $W_j := W_{j-1} \cup \bigcup_{o \in O}(\mathit{preimg}_o(W_{j-1}) \cap \mathit{spreimg}_o(C_i))$
            if $W_j = W_{j-1}$:
                break
        $C_i := W_j$
        if $C_i = C_{i-1}$:
            return $\langle C_i, \langle W_0, \ldots, W_{j-1} \rangle \rangle$

Procedure *prune*
Correctness

Lemma (Procedure prune)

Let S and $G \subseteq S$ be sets of states and O a set of operators Then prune(S, O, G) terminates after a finite number of steps and returns $C \subseteq S$ such that there is $\pi : C \setminus G \to O$ with the following properties:

- Hope: *For every $s \in C$ there is an execution $s_0, \ldots, s_n$ of π such that $s = s_0$ and $s_n \in G$.*
- Safety: *For every $s \in C \setminus G$, img$_{\pi(s)}(s) \subseteq C$.*
- Maximality: *There is no set $C' \not\subseteq C$ and $\pi' : C' \setminus G \to O$ satisfying the hope and safety properties.*

- The sets W_j also returned by *prune* encode weak distances and can be used to define the strong cyclic plan π.

Strong cyclic planning algorithm
Main algorithm

The planning algorithm

def strong-cyclic-plan($\langle A, I, O, G\rangle$):
    $S := A \rightarrow \{0, 1\}$
    $S_G := \{s \in S \mid s \models G\}$
    $\langle S^*, (W_j)_{j=0,1,2,\ldots}\rangle = \mathit{prune}(S, O, S_G)$
    if $\exists s \in S : s \models I \wedge s \notin S^*$:
        return no solution
    for each $s \in S^*$:
        $\delta(s) := \min\{j \in \mathbb{N}_0 \mid s \in W_j\}$
    for each $s \in S^* \setminus S_G$:
        $\pi(s) :=$ some operator $o \in O$ with $\mathit{img}_o(s) \subseteq S^*$
            and $\min\{\delta(s') \mid s' \in \mathit{img}_o(s)\} < \delta(s)$
    return π

Strong cyclic planning algorithm
Complexity

- The procedure *prune* runs in polynomial time in the number of states because the number of iterations of each loop is at most n – hence there are $O(n^2)$ iterations – and computation on each iteration takes polynomial time in the number of states.
- Finding strong cyclic plans for full observability is in the complexity class EXPTIME.
- The problem is also EXPTIME-hard.
- Similar to strong planning, we can speed up the algorithm in many practical cases by using a symbolic implementation (e. g. with BDDs).

Maintenance goals

- In this lecture, we usually limit ourselves to the problem of finding plans that reach a goal state.
- In practice, planning is often about more general goals, where execution cannot be terminated.
  1. An animal: find food, eat, sleep, find food, eat, sleep, ...
  2. Cleaning robot: keep the building clean.
- These problems cannot be directly formalized in terms of reachability because infinite (unbounded) plan execution is needed.
- We do not discuss this topic in full detail. However, to give at least a little impression of planning for temporally extended goals, we will discuss the simplest objective with infinite plan executions: maintenance.

Definition

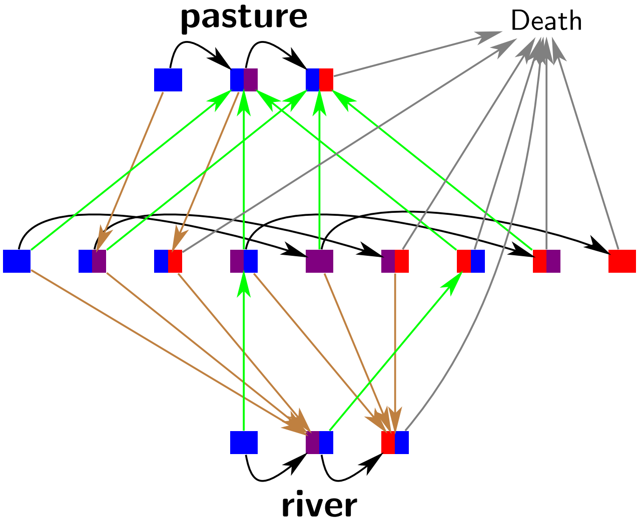
Let $\mathcal{T} = \langle A, I, O, G, V \rangle$ be a planning task.
A strategy π for $\mathcal{T}$ is called a plan for maintenance for $\mathcal{T}$ iff

- it contains no leaf nodes,
- all cycles contain at least one operator node, and
- $b(n) \models G$ for all nodes n of the strategy.

Maintenance goals
Example

- The state of an animal is determined by three state values: hunger (0, 1,2), thirst (0, 1, 2) and location (river, pasture, desert). There is also a special state called death.
- Thirst grows when not at river; at river it is 0.
- Hunger grows when not on pasture; on pasture it is 0.
- If hunger or thirst exceeds 2, the animal dies.
- The goal of the animal is to avoid death.

Maintenance goals
Transition system for the example



We can infer rules backwards starting from the death condition.

1. If in desert and $thirst = 2$, must go to river.
2. If in desert and $hunger = 2$, must go to pasture.
3. If on pasture and $thirst = 1$, must go to desert.
4. If at river and $hunger = 1$, must go to desert.

If the above rules conflict, the animal will die.

Algorithm for maintenance goals
Idea

Summary of the algorithm idea

Repeatedly eliminate from consideration those states that in one or more steps unavoidably lead to a non-goal state.

- A state is i-safe iff there is a plan that guarantees "survival" for the next i actions.
- A state is safe (or ∞-safe) iff it is i-safe for all $i \in \mathbb{N}_0$.
- The 0-safe states are exactly the goal states: maintenance objective is satisfied for the current state.
- Given all i-safe states, compute all $i+1$-safe states by using strong preimages.
- For some $i \in \mathbb{N}_0$, i-safe states equal $i+1$-safe states because there are only finitely many states and at each step and $i+1$-safe states are a subset of i-safe states. Then i-safe states are also ∞-safe.

Planning for maintenance goals

def maintenance-plan($\langle A, I, O, G \rangle$):

$S := A \rightarrow \{0, 1\}$

$\mathit{Safe}_0 := \{s \in S \mid s \models G\}$

for each $i \in \mathbb{N}_1$:

$\mathit{Safe}_i := \mathit{Safe}_{i-1} \cap \bigcup_{o \in O} \mathit{spreimg}_o(\mathit{Safe}_{i-1})$

if $\mathit{Safe}_i = \mathit{Safe}_{i-1}$:

break

if $\exists s \in S : s \models I \wedge s \notin \mathit{Safe}_i$:

return no solution

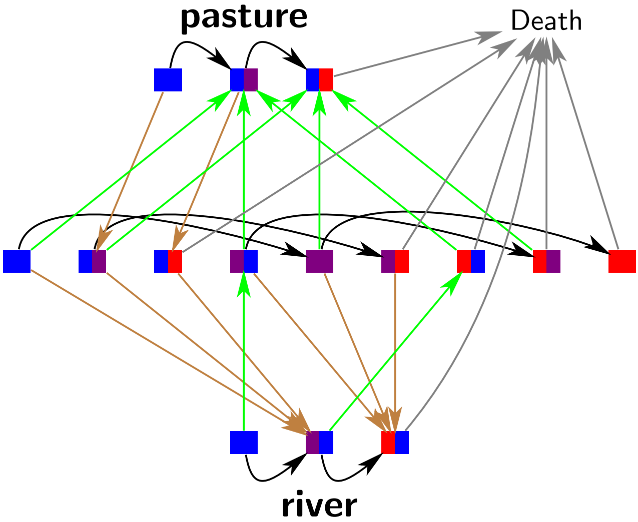
for each $s \in \mathit{Safe}_i$:

$\pi(s) :=$ some operator $o \in O$ with $\mathit{img}_o(s) \subseteq \mathit{Safe}_i$

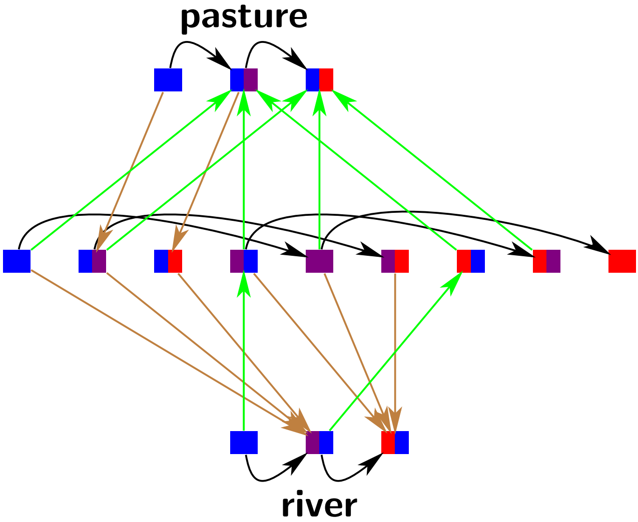
return π

Maintenance goals

Transition system for the example

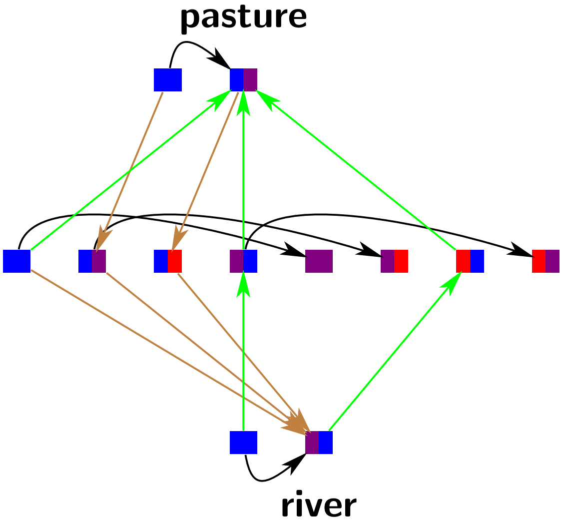


Maintenance goals
0-safe states



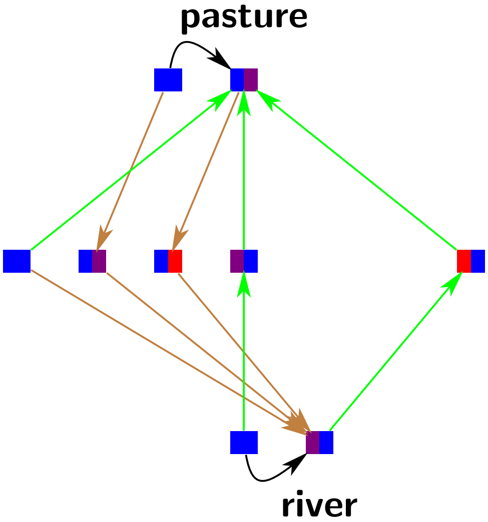
Maintenance goals

1-safe states



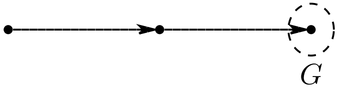
Maintenance goals

i-safe states for all $i \geq 2$

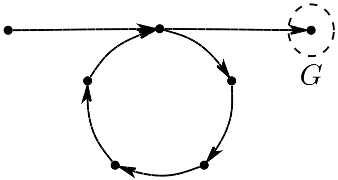


Different planning objectives

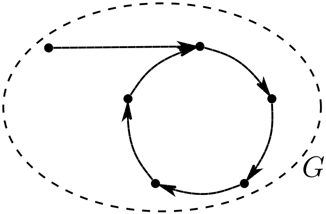
Strong planning



Strong cyclic planning



Maintenance



Outlook: Computational tree logic

- We have considered different classes of solutions for planning tasks by defining different planning problems.
  - strong planning problem: find a strong plan
  - strong cyclic planning problem: find a strong cyclic plan
  - ...
- Alternatively, we could allow specifying goals in a modal logic like computational tree logic to directly express the type of plan we are interested in using modalities such as A (all), E (exists), G (globally), and F (finally).
  - Weak planning: $\mathrm{EF}\varphi$
  - Strong planning: $\mathrm{AF}\varphi$
  - Strong cyclic planning: $\mathrm{AGEF}\varphi$
  - Maintenance: $\mathrm{AG}\varphi$
  - Strong recoverability: $\mathrm{AGAF}\varphi$

Summary

- We have extended our earlier planning algorithm from strong plans to strong cyclic plans.
- The story does not end there: When considering infinitely executing plans, many more types of goals are feasible.
- We considered maintenance as a simple example of a temporally extended goal.
- In general, temporally extended goals be expressed in modal logics such as computational tree logic (CTL).
- We presented dynamic programming (backward search) algorithms for strong cyclic and maintenance planning.
- In practice, one might implement both algorithms by using binary decision diagrams (BDDs) as a data structure for state sets.