## Deterministic planning: problem instances

A *problem instance* is a 4-tuple $\langle P, I, O, G\rangle$ where

1. $P$ is a finite set of state variables,
2. $I$ is a state (a valuation of $P$) called the *initial state*,
3. $O$ is a finite set of operators over $P$, and
4. $G$ is a propositional formula over $P$ (the *goal*).



## Deterministic planning: plans

A *solution* of a problem instance $\langle P, I, O, G\rangle$ is a sequence $\pi = o_1, \ldots, o_n$ of operators (a *plan*) such that $\{o_1, \ldots, o_n\} \subseteq O$ and $s_0, \ldots, s_n$ is a sequence of states (the *execution* of $\pi$) so that

1. $s_0 = I$,
2. $s_i = \text{app}_{o_i}(s_{i-1})$ for every $i \in \{1, \ldots, n\}$, and
3. $s_n \models G$.

This can be equivalently expressed as

$$\text{app}_{o_n}(\text{app}_{o_{n-1}}(\cdots \text{app}_{o_1}(I)\cdots)) \models G$$



## Properties of plans

Let $\langle P, I, O, G\rangle$ be a problem instance.

1. There is a plan of length 0 iff $I \models G$.
2. Shortest plan may not be longer than $2^n - 1$: If a plan is longer, then it visits some state $s$ more than once and has the form $\sigma_1 \; {}^{s} \; \sigma_2 \; {}^{s} \; \sigma_3$: the plan $\sigma_1\sigma_3$ is shorter.
3. Shortest plan may have length $2^n - 1$: Reach the goal state $111\ldots1$ from the initial state $000\ldots0$ by an operator that increments the corresponding binary number $2^n - 1$ times.



## Deterministic planning: expressivity

The decision problem SAT: test whether a given propositional formula $\phi$ is satisfiable.

$$\begin{aligned} P &= \text{the set of propositional variables occurring in } \phi \\ I &= \text{any state, e.g. all state variables have value 0} \\ O &= (\{\top\} \times P) \cup (\{\langle \top, \neg p\rangle | p \in P\}) \\ G &= \phi \end{aligned}$$

The problem instance has a solution if and only if $\phi$ is satisfiable.

## Deterministic planning: expressivity

- Because we have a polynomial-time translation from SAT to deterministic planning, and SAT is an NP-complete problem, we have a polynomial time translation from **every decision problem in NP** to deterministic planning.
- Does deterministic planning have the power of NP, or is it still more powerful?



## Turing machines

A Turing machine $\langle \Sigma, Q, \delta, q_0, g \rangle$ consists of

1. an alphabet $\Sigma$ (a set of symbols),
2. a set $Q$ of internal states,
3. a transition function $\delta$ that maps $\langle q, s \rangle$ to a tuple $\langle s', q', m \rangle$ where $q, q' \in Q$, $s \in \Sigma \cup \{|, \Box\}$, $s' \in \Sigma$ and $m \in \{L, N, R\}$.
4. an initial state $q_0 \in Q$, and
5. a labeling $g : Q \rightarrow \{\text{accept}, \text{reject}, \exists\}$ of states.



## TMs, example

TM accepting strings $\epsilon, 1, 12, 121, 1212, \ldots$ is $\langle \Sigma, Q, \delta, q_1, g \rangle$ where

$$
\begin{array}{rcllrcl}
\Sigma & = & \{1, 2\}, & & & & \\
Q & = & \{q_1, q_2, q_3, q_4\}, & & & & \\
g(q_1) & = & \exists, & & g(q_2) & = & \exists, \\
g(q_3) & = & \text{accept}, & & g(q_4) & = & \text{reject} \\
\delta(q_1, 1) & = & \langle 1, q_2, R \rangle & & \delta(q_1, 2) & = & \langle 2, q_4, R \rangle \\
\delta(q_2, 2) & = & \langle 2, q_1, R \rangle & & \delta(q_2, 1) & = & \langle 1, q_4, R \rangle \\
\delta(q_1, \Box) & = & \langle 1, q_3, R \rangle & & \delta(q_2, \Box) & = & \langle 2, q_3, R \rangle \\
\delta(q, s) & = & \langle 1, q_4, R \rangle \text{ for all other } q, s & & & &
\end{array}
$$



## TMs, example: cont'd

What does the TM do with the string 12122?

$$
\begin{array}{ll}
q_1 & |\widehat{1}2122\Box \\
q_2 & |1\widehat{2}122\Box \\
q_1 & |12\widehat{1}22\Box \\
q_2 & |121\widehat{2}2\Box \\
q_1 & |1212\widehat{2}\Box \\
q_4 & |12122\widehat{\Box}
\end{array}
$$

The label $g(q_4) =$ *reject*. The TM does not accept the string.

## Simulation of PSPACE Turing machines

We show how polynomial-space Turing machines can be simulated by planning.

- contents of tape cells are encoded as state variables
- R/W head location is encoded as state variables
- internal state of the TM is encoded as state variables
- transitions are encoded as operators

A given Turing machine $M$ accepts an input string $\sigma$ if and only if a problem instance $T(M, \sigma) = \langle P, I, O, G \rangle$ has a plan.



## PSPACE simulation I

Simulate a TM = $\langle \Sigma, Q, \delta, q_0, g \rangle$ that needs at most $p(n)$ tape cells on an input string of length $n$.

State variables in the problem instance in planning are

1. $\{q_1, \ldots, q_{|Q|}\} = Q$ for denoting the current state of the TM,
2. $s_i$ for every symbol $s \in \Sigma \cup \{|, \Box\}$ and tape cell $i \in \{0, \ldots, p(n)\}$,
3. $h_i$ for every $i \in \{0, \ldots, p(n)\}$ (position of the R/W head).



## PSPACE simulation II

1. $I(q_0) = 1$ and $I(q) = 0$ for all $q \in Q \backslash \{q_0\}$.
2. $I(s_i) = 1$ if $i < n$ and input symbol $i$ is $s$.
3. $I(s_i) = 0$ if $i < n$ and $s \in S$ and symbol $i$ is not $s$.
4. $I(\Box_i) = 1$ iff $i \in \{n, \ldots, p(n) - 1\}$
5. $I(|_i) = 1$ iff $i = 0$
6. $I(h_i) = 1$ iff $i = 1$



## PSPACE simulation III

Goal of the problem instance is to reach an accepting state.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}.$$

## PSPACE simulation IV

For all $s \in \Sigma \cup \{|, \Box\}$ and $q \in Q$ and $i \in \{0, \ldots, p(n)\}$ with $\delta(q, s) = \langle s', q', m \rangle$ and $m \neq R$ or $i < p(n)$), define

$$o_{s,q,i} = \langle h_i \wedge s_i \wedge q, \nu \wedge \chi \wedge \mu \rangle$$

where

$\nu$ is

$\top$ if $s \in \{|, s'\}$, and $\neg s_i \wedge s'_i$ otherwise,



## PSPACE simulation IV, cont'd

$\chi$ is

$\neg q \wedge q'$ if $q \neq q'$, and $\top$ otherwise, and

$\mu$ is

$\top$ if $m = N$

$\neg h_i \wedge h_{i-1}$ if $i > 0$ and $m = L$, and $\top$ if $i = 0$ and $m = L$

$\neg h_i \wedge h_{i+1}$ if $i < p(n)$ and $m = R$, and $\top$ if $i = p(n)$ and $m = R$



## Example: a Turing machine

Turing machine $\langle \{A, B\}, \{q_1, q_2, q_{acc}\}, \delta, q_1, g \rangle$ where $\delta$ is

| | $A$ | $B$ | $\vert$ | $\Box$ |
|---|---|---|---|---|
| $q_1$ | $\langle A, q_1, R \rangle$ | $\langle B, q_2, N \rangle$ | $\langle \vert, q_2, R \rangle$ | $\langle B, q_1, N \rangle$ |
| $q_2$ | $\langle A, q_1, L \rangle$ | $\langle A, q_{acc}, N \rangle$ | $\langle \vert, q_1, R \rangle$ | $\langle A, q_2, L \rangle$ |
| $q_{acc}$ | $-$ | $-$ | $-$ | $-$ |

and $g(q_acc)$ = accept, $g(q_1) = \exists$ and $g(q_2) = \exists$.

Input string: ABAAB



## Example: translation to planning

Construct $\langle P, I, O, G \rangle$ where

1. $P = \{q_1, q_2, q_{acc}, h_0, \ldots, h_{p(5)}, A_0, \ldots, A_{p(5)}, B_0, \ldots B_{p(5)} \ldots\}$
2. $I \models |_0 \wedge A_1 \wedge B_2 \wedge A_3 \wedge A_4 \wedge B_5 \wedge \Box_6 \wedge \Box_7 \wedge \cdots \wedge \Box_{p(5)} \wedge \neg A_0 \wedge \neg B_0 \wedge \neg \Box_0 \wedge \cdots$
3. operators $O$ are on the next slide
4. $G = q_{acc}$

## Example: translation to planning

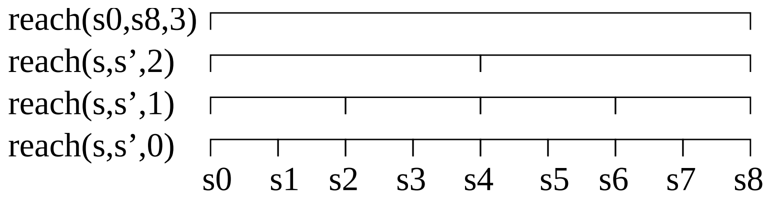
Only part of the about $|\{0,1,\ldots,p(5)\} \times |\{q_1,q_2\}| \times |\{A,B,|,\Box\}|$ operators are given below, for R/W head position 1 and input symbols A and B:

$$
\begin{aligned}
O = \{ & \langle h_1 \wedge A_1 \wedge q_1, \quad \neg h_1 \wedge h_2 \rangle, \ldots, \\
& \langle h_1 \wedge B_1 \wedge q_1, \quad \neg q_1 \wedge q_2 \rangle, \ldots, \\
& \langle h_1 \wedge A_1 \wedge q_2, \quad \neg q_2 \wedge q_1 \wedge \neg h_1 \wedge h_0 \rangle, \ldots, \\
& \langle h_1 \wedge B_1 \wedge q_2, \quad \neg B_1 \wedge A_1 \wedge \neg q_2 \wedge q_{acc} \rangle, \ldots \}
\end{aligned}
$$



## Deterministic planning can be solved in PSPACE

Recursive algorithm for testing $m$-step reachability between two states with $\log m$ memory consumption.






## Deterministic planning can be solved in PSPACE

Existence of plans of length $\leq 2^n$:

*PROCEDURE* reach($s$,$s'$,$n$)
*IF* $n = 0$ *THEN*
  *IF* s = s' OR $s' = \text{app}_o(s)$ for some $o \in O$ *THEN RETURN* true
  *ELSE RETURN* false;
*ELSE*
  *FOR* all states $s''$ *DO*
    *IF* reach($s$,$s''$,$n-1$) *AND* reach($s''$,$s'$,$n-1$) *THEN RETURN* true
  *END*
  *RETURN* false;



## Deterministic planning can be solved in PSPACE

CORRECTNESS:

For problem instance $N$ with $n$ state variables, $N$ has a plan if and only if reach($I$,$s'$,$n$) returns true for some $s'$ such that $s' \models G$.

MEMORY CONSUMPTION:

If number of states is $2^n$, then recursion depth is $n$. At each recursive call only one state $s''$ is represented, taking space $\mathcal{O}(n)$, which means that total memory consumption at any time point is $\mathcal{O}(n^2)$, which is polynomial in the size of the problem instance.

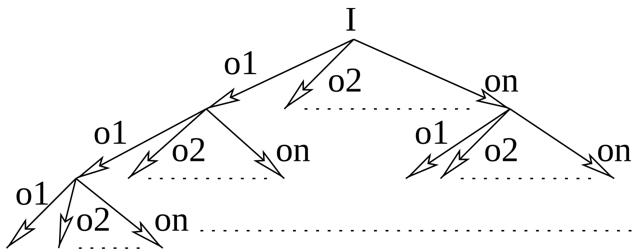
## Progression

- Progression is computing the successor state $\text{app}_o(s)$ of $s$ with respect to $o$.
- Used in *forward search* in a transition system: from the initial state toward the goal states.
- Efficient to implement.
- Only for deterministic planning: *nondeterministic operators* may produce a *set of states* from one state.



## Search algorithms 1: Search with progression

depth-first search, breadth-first search, iterative deepening, informed search, ...






## Search algorithms: systematic vs. local

Systematic algorithms:

- Keep track of all the states already visited.
- Memory consumption may be high.
- Always find a plan if one exists.
- depth-first, breadth-first, A$*$, IDA$*$, WA$*$, best-first, ...



## Search algorithms: systematic vs. local

Local search algorithms:

- Keep track of only one search state at a time.
- Find a plan with a high probability (given enough time...).
- Cannot determine that no plans exist.
- hill-climbing, simulated annealing, tabu search, ...

## Search algorithms: A$*$

Use the function $f(s) = g(s) + h(s)$ to guide search:

- $g(s)$ = cost so far (number of operators)
- $h(s)$ = estimated remaining cost (estimated distance)

  $h(s)$ must be less than or equal the real remaining cost (distance): otherwise A$*$ is not guaranteed to find an optimal solution. (*admissibility of* $h(s)$).

(IDA$*$ improves A$*$ on memory consumption.)



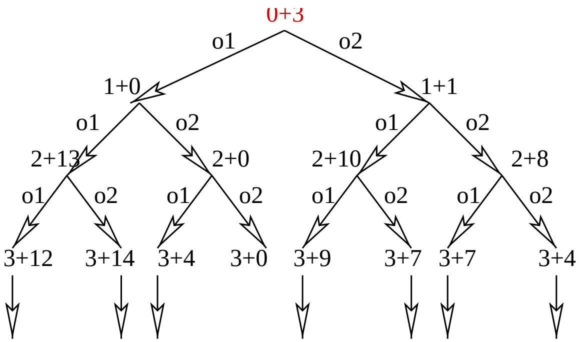
## Search algorithms: A$*$, cont'd

The algorithm tries to reach a state in $G$ from $I$ as follows.

1. OPEN := $\{I\}$, CLOSED := $\emptyset$.
2. If some state in $G$ is in OPEN, then stop: solution found.
3. If OPEN = $\emptyset$, then stop: no solution.
4. Choose an element $s \in$ OPEN with the least $f(s)$.
5. OPEN := OPEN$\backslash\{s\}$, CLOSED := CLOSED$\cup\{s\}$.
6. OPEN := OPEN $\cup(\{\text{app}_o(s)|o \in O\}\backslash$CLOSED$)$.
7. Go to 2.

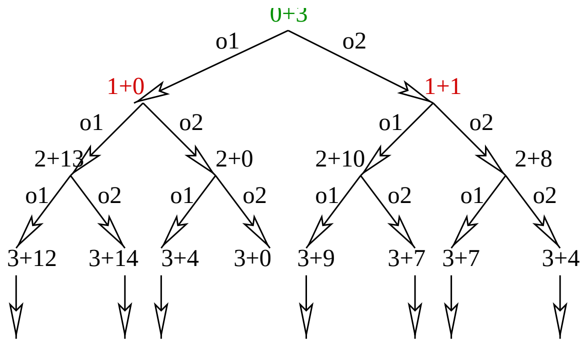


## Search algorithms: A$*$, example

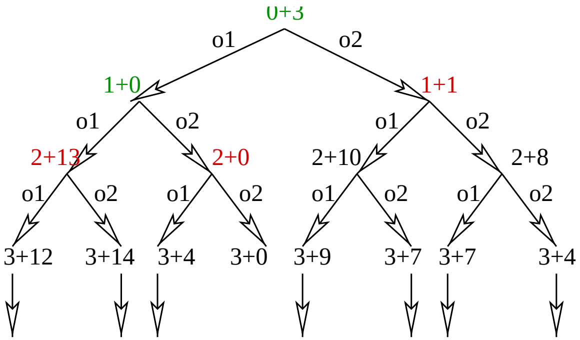





## Search algorithms: A$*$, example

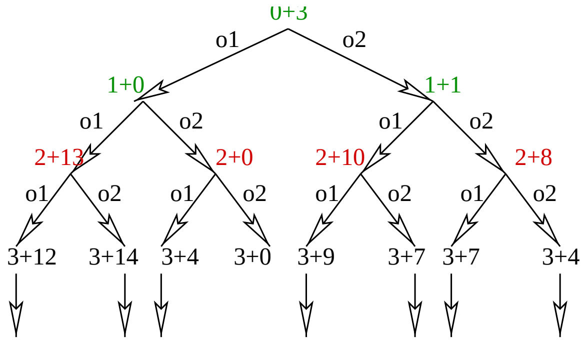


## Search algorithms: A∗, example



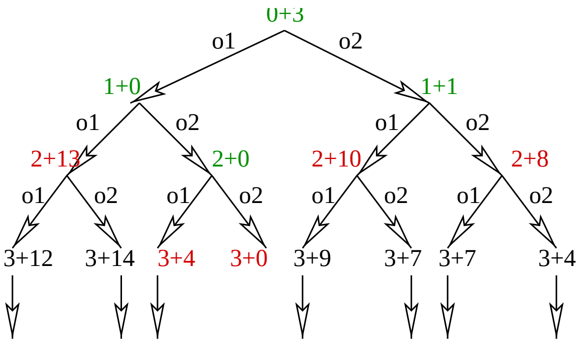



## Search algorithms: A∗, example






## Search algorithms: A∗, example






## Search algorithms: WA∗

A general property of (planning) algorithms: finding optimal solutions is **much** more difficult than finding any solution.

- By sacrificing optimality of A∗, plans can be found faster.
- WA∗ uses $f(s) = g(s) + Wh(s)$ for $W \geq 1$.
- With $W = 1$ we have WA∗ = A∗.
- With $W > 1$ search will be suboptimal and faster.
- Plan length may be $W$ times the optimum.

## Search algorithms: best-first search

- Like WA$*$, but the cost-so-far is ignored completely.
- Best-first search uses $f(s) = h(s)$ for $W \geq 1$.
- No guarantees on plan length.



## Search space vs. state space

Search space does not in general coincide with state space.

Exception: forward search with a systematic search algorithm, because the systematic search algorithm can be implemented so that it keeps track of the sequence of actions that have been taken.



## Plan search: search states for progression

For progression, the search state is represented as a sequence of operators and associated states.

$$s_I, o_1, s_1, o_2, s_2, \ldots, o_n, s_n$$

The neighbors of the state are those obtained by progression with respect to one of the operators or by dropping out some of the last operators and associated states:

1. $s_I, o_1, s_1, o_2, s_2, \ldots, o_n, s_n, o, \text{app}_o(s_n)$ for some $o \in O$
2. $s_I, o_1, s_1, o_2, s_2, \ldots, o_i, s_i$ for $i < n$ (for local search only)



## Local search: random walk

1. $s := I$
2. If $s \in G$, stop: goal state has been reached.
3. Randomly choose a neighbor $s'$ of $s$.
4. $s := s'$
5. Go to 2.

## Local search: steepest descent hill-climbing

1. $s$ := $I$
2. If $s \in G$, stop: goal state has been reached.
3. Randomly choose neighbor $s'$ of $s$ with the least $h(s')$.
4. $s$ := $s'$
5. Go to 2.

Problem: The algorithm gets stuck in local minima.



## Local search: simulated annealing

1. $s$ := $I$
2. If $s \in G$, stop: goal state has been reached.
3. Randomly choose a neighbor $s'$ of $s$.
4. If $h(s') < h(s)$ go to 7.
5. With probability $\exp(-\frac{h(s')-h(s)}{T})$ go to 7.
6. Go to 3.
7. $s$ := $s'$
8. Decrease $T$. (There are many strategies for doing this!!)
9. Go to 2.

The temperature $T$ is initially high and then gradually decreased.



## Local search: simulated annealing, picture