Artificial Intelligence 15-381
Search Methods

Jaime Carbonell jgc@cs.cmu.edu
20-January-2000

Today's Angeda

- Depth-first and breadth-first search
- Bi-directional and island search
- Heuristics and evaluation functions
- Hueristic search (HC, best-first)
- Admissible HS (A*, B*)

More on the State Space

Useful functions:

$\text{Succ}(s_i) = \{s_k \mid o_j(s_i) = s_k\text{D}\}$

$\text{Reachable}(s_i) = \{\text{U}\{s_k\} \mid \text{Succ}^*(s_i)\}$

$\text{Succ}^{-1}(s_i) = \{s_k \mid o_j(s_k) = s_i)$

$\text{Reachable}^{-1}(s_i) = \{\text{U}\{s_k\} \mid (\text{Succ}^{-1})^*(s_i)\}$

$\text{s-Path}(s_{a_0}, s_{a_n}) = (s_{a_0}, s_{a_1}, \ldots, s_{a_n})$
...such that for all s_{a_i} exists $o_j(s_{a_i}) = s_{a_{i+1}}$

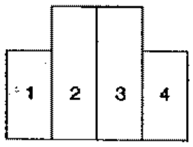
$\text{o-Path}(s_{a_0}, s_{a_n}) = (o_{j_0}, o_{j_1}, \ldots, o_{j_{n-1}}$
)
...such that for all s_{a_i} exists $o_j(s_{a_i}) = s_{a_{i+1}}$

Useful concepts:

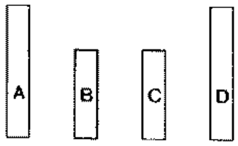
- Solution = o-Path(s_0, s_G) [or s-Path]
- cost(Solution) = Σcost(o_j) ... (often cost(o_j) = 1)
- P is solvable if at least one o-Path(s_0, s_G) exists
- Solutions may be constructed forward, backward or any which way.
- State spaces may be finite, infinite, implicit or explicit

AN R1 COMPUTER CONFIGURATION TASK

There is a backplane that has four slots — 1 and 4 are of width 4, 2 and 3 are of width 6

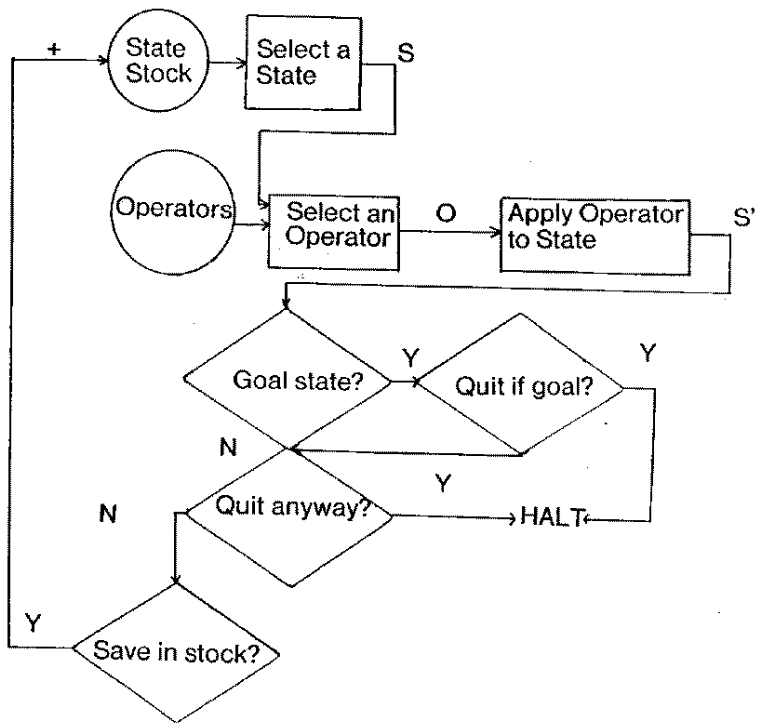


There are four boards to be placed into the backplane — A of width 6, B of width 4, C of width 4 but must precede A, and D of width 6 but must go in slot 2



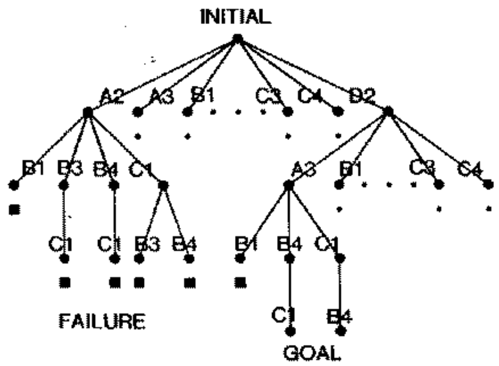
B-22

A FLOW CHART FOR ALL SEARCHES



B-27

AN R1 TREE

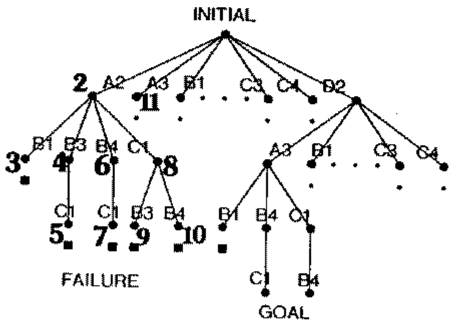


B-23

Start with the initial state.

Repeat until a goal state is found: If the current state = goal state, return the path as the solution.

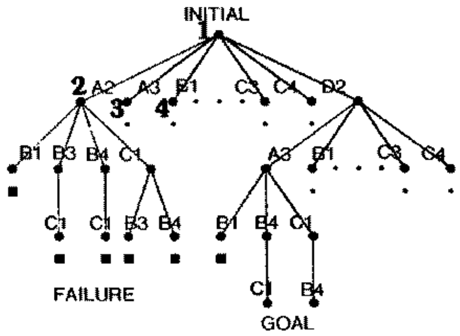
If the state has an unexamined successor state then go to it, else *backtrack*



Start with the initial state.

Repeat until a goal state is found:

Expand all of the nodes at the next depth,
stopping when one node = goal state.

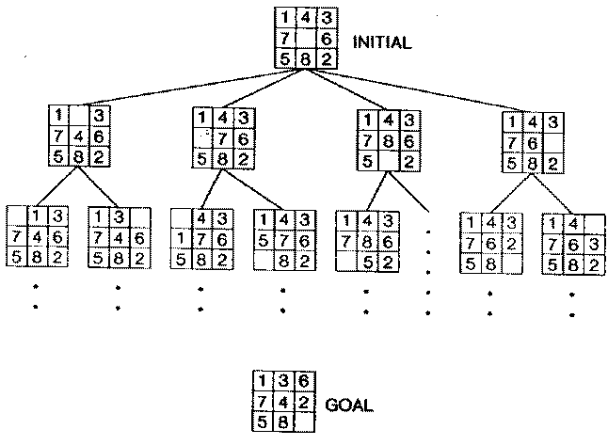


PROPERTIES OF BREADTH-FIRST SEARCH

- Same decisions as depth-first except extend least recent first (FIFO)
- This is a forward full-width search.
- It is guaranteed to find the shortest path to the goal.
- It works best when the goal is known to be at a shallow depth.
- It can require a lot of memory to store all the nodes at the previous depth.

B-33

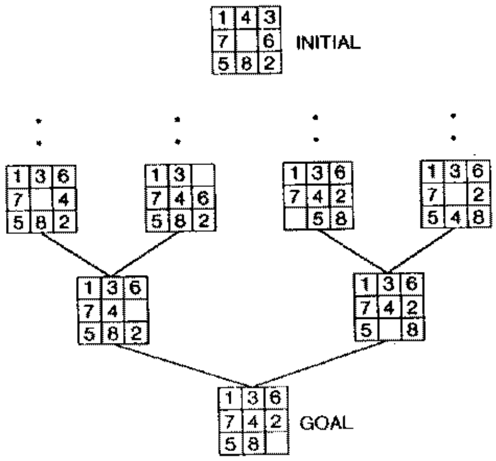
AN EIGHT-PUZZLE TREE



BACKWARD SEARCH

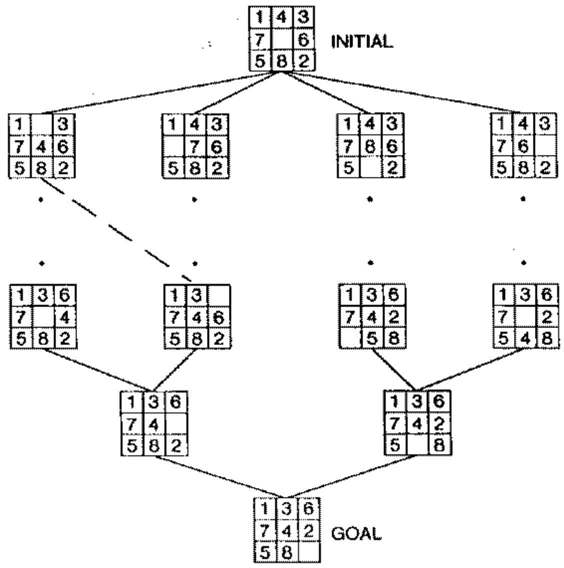
- Start at the goal state and go backwards.

- Requires that an explicit goal state be generated, and that inverse operators exist.

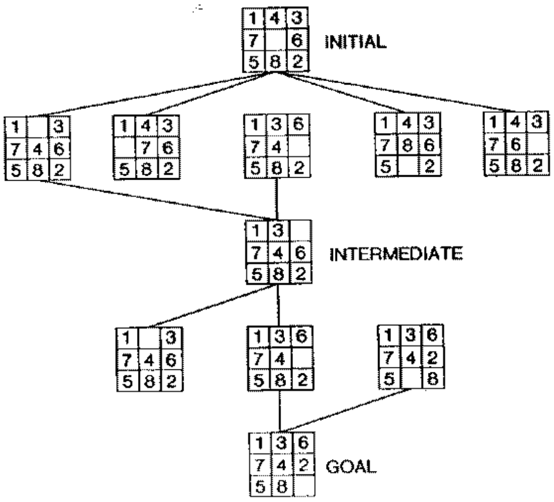


B-34

Work forward from the initial state and backwards from the goal state until the two searches meet in the middle.



Generate several states that *must* be on the solution path, and try to connect them via forward and backward searches.



EXAMPLES OF EVALUATION FUNCTIONS

- For chess, the difference between the sums of the piece values for the two players: If black has a king, and a pawn (1), while white has a king, a pawn (1) and a bishop (3), then the value for white is 3.

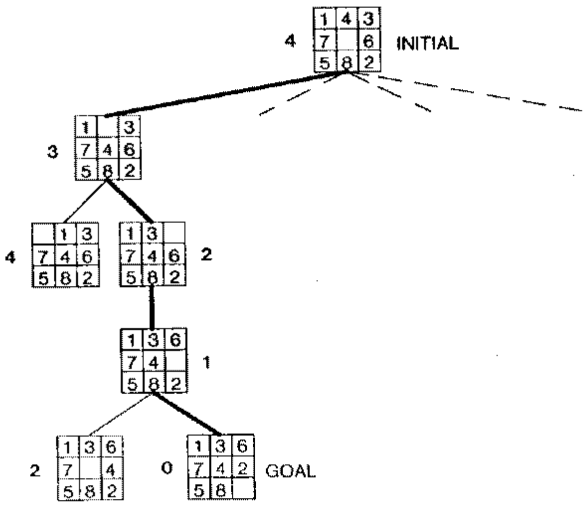
- For R1, the cost of the partial configuration can be used to yield a minimum cost result.

- For the eight puzzle, count the number of tiles that are out of place or their distance from the correct location.

B-55

Repeat until a state is found that is better than the current state:

- Generate and evaluate a successor state, stopping when a goal state is reached.



Heuristics in AI Search

Definition

A **Heuristic** *is an operationally-effective nugget of information on how to direct search in a problem space. Heuristics are only approximately correct.*

Examples

- "If-then" rules for state-transition selection.
- Evaluation functions on states for mono- and bi-polar search.
- Macro-operator formation [discussed later].
- Problem decomposition [e.g. hypothesizing islands on the search path].
- Estimation of distance between S_{curr} and S_G.

Heuristics in AI Search

Definition

A **Heuristic** *is an operationally-effective nugget of information on how to direct search in a problem space. Heuristics are only approximately correct.*

Examples

- "If-then" rules for state-transition selection.
- Macro-operator formation [discussed later].
- Problem decomposition [e.g. hypothesizing islands on the search path].
- Estimation of distance between S_{curr} and S_G. (e.g. manhattan, eucledian, topological distance)
- Value function on each Succ(S_{curr})
- cost(path(S_0,S_{curr})) + E[cost(path(S_{curr},S_G))]
- Utility: value(S) - cost(S)

Heuristics Search

Greedy Search Methods

- Hill-Climbing (incl. steepest-ascent)
- Gradient Search (heuristic DFS)
- Delta-Min Search

Best-First Search

```
BESTFS(S_curr, S_goal, S-LIST)
IF S_curr = S_goal, SUCCESS
For s_i in SUCC(S_curr)
  Insert-sort(<s_i, f(s_i)>, S-LIST)
IF S-LIST = Null, FAILURE
  ELSE RETURN BESTFS(FIRST(S-LIST), S_goal, S-LIST)
```

- Full-breadth search
- "Ragged" fringe expansion

Beam Search

- ***Best-few*** BFS
- Beam-width parameter
- Uniform fringe expansion

A* Search

Cost Function Definitions

Let $g(S_{curr})$ =
actual cost to reach S_{curr} from S_0

Let $h(S_{curr})$ =
estimated cost from S_{curr} to S_G

Let $f(S_{curr}) = g(S_{curr}) + h(S_{curr})$

Admissibility Definition

An heuristic is **admissible** *with respect to a search method if it guarantees finding the optimal solution first, even when its value is only an estimate.*

The A* Criterion

If $h(S_{curr})$ always equals or underestimates the remaining cost, then $f(S_{curr})$ is admissible with respect to Best-First Search

A* Search

A* Search = BESTFS with admissible $f = g + h$ under the admissibility constraints above

B* Search

Ideas

- Admissible heuristics for mono- and bi-polar search
- "Eliminates" horizon problem in game-trees [more later]

Definitions

Let Best(S) = Always optimistic eval fn.

Let Worst(S) = Always pessimistic eval fn.

Hence: Worst(S) < True-eval(S) < Best(S)

Basic B* Search

```
B*(S) is defined as:

If there is an S_i in SUCC(S_curr)
    s.t. Forall other S_j in SUCC(S_curr)
         W(S_i) > B(S_j)
   Then select S_i
   Else ProveBest(SUCC(S_curr)
     or DisproveRest(SUCC(S_curr)
```

Difficulties in B*

- Guaranteeing eternal pessimism in W(S) (eternal optimism is somewhat easier)
- Switching among ProveBest and DisproveRest
- Usually W(S) << True-eval(S) << B(S) (not possible to achieve $W(S_i) > B(S_j)$)

CONSTRAINTS IN THE R1 TASK

- Board A is of width 6, so it can only fit into the two slots of width 6: {2, 3}.
- Board B is of width 4, so it can fit in any of the slots: {1, 2, 3, 4}.
- Board C is of width 4, so it can fit in any of the slots (it is difficult to directly apply the constraint that it must be before board A until board A has been configured): {1, 2, 3, 4}.
- Board D must go in slot 2: {2}.

B-45

EXAMPLE OF CONSTRAINT
PROPAGATION: R1