SEARCH

BLIND SEARCH: *Systematic Search*

Depth 1st: Continue along current path looking for goal

Breadth 1st: Expand all nodes at current level before progressing to next level

Depth–limited Search: Depth-1st + depth-limit

Iterative Deepening Search: limit=0,limit=1, ...

USING COST: $g(n)$=*cost from start to n*

Uniform Cost Search: Select best $cost(n)$

Branch-and-bound: Select best $g(n)$.

USING HEURISTIC: $h(n)$=*Estimate cost to a goal*

Greedy Search: Select best $h(n)$

A*: Select best $f(n) = g(n) + h(n)$

IDA*: Depth-1st + f-cost limit.

Hill-climbing: Depth-1st exploring best $h(n)$ first

Simulated Annealing: Hill climbing + random walk

Beam Search: Breadth-1st pruning worse $h(n)'s$

BRANCH-AND-BOUND

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else
   - remove head of queue,
   - expand it,
   - place in queue, and
   - **sort entire queue** with **least cost-so-far** nodes in front
5. Recurse to 2

BRANCH-AND-BOUND SUMMARY

When to use

- Desire best solution
- Keep track of cost so far

When to avoid

- May be overly conservative

Potential improvement

- Dynamic Programming

BRANCH-AND-BOUND + DYNAMIC PROG.

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else
   - remove head of queue,
   - expand it,
   - place in queue,
   - ⋆ remove redundant paths:
     Paths that reach the same node as other paths but are more expensive, and
   - **sort entire queue** with **least cost-so-far** nodes in front
5. Recurse to 2

A*

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place in queue, and **sort entire queue** with **least cost-so-far** + **estimated-cost-remaining** nodes in front
5. If multiple paths reach a common goal, keep only lowest cost-so-far path
6. Recurse to 2

$$f(\text{node}) = g(\text{node}) + h(\text{node}),$$

where
$f(\text{node})$ = estimated total cost
$g(\text{node})$ = cost-so-far to node
$h(\text{node})$ = estimated-cost-remaining (*heuristic*).

Properties of h:

- Lower bound ($\leq$ actual cost)

- Nonnegative
- Closer h to true cost, more heuristic information
- What if $h(\text{node}) = 0$ for all nodes?

A^* SUMMARY

When to use

- Desire best solution
- Keep track of cost so far
- Heuristic information available

When to avoid

- No good heuristics

HILL CLIMBING SEARCH

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place **sorted** children in front of queue
5. Recurse to 2

HILL CLIMBING SUMMARY

When to use

- Depth limited or known beforehand
- All solutions at same depth
- Desire good solution
- Reliable estimate of remaining distance to goal
- Fast if good estimate

When to avoid

- Large or infinite subtrees
- No good estimate
- Difficult terrain

BEAM SEARCH

1. Put start state onto queue
2. If queue is empty then fail
3. If head of queue is goal then succeed
4. Else remove head of queue, expand it, place children at end of queue
5. If finishing a level, keep only w best nodes in queue
6. Recurse to 2

BEAM SEARCH SUMMARY

When to use

- Large or infinite search tree
- Solution depth unknown
- Prefer shallow solution
- Possibly fast
- No more than wb nodes stored

When to avoid

- Can't tell which solutions to prune
- Prefer conservative