Problem Solving

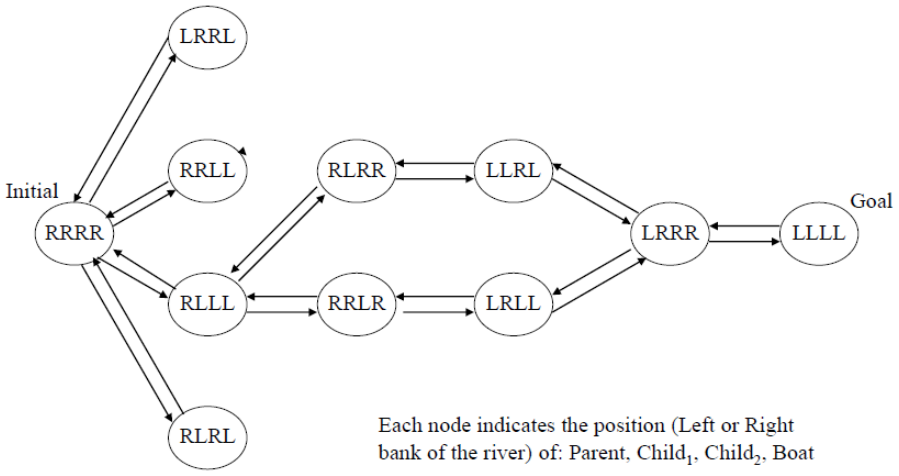
Search Methodology

1. Formulate problem solving as search on a graph
2. Given a problem:

nodes	Specify representation of states, especially initial and goal states
arcs	Specify actions (successor function, neighborhood function) that take us from one state to another
search	Find a path from an initial state to a goal state

3. Depending on the problem: assign a cost to each arcs or nodes

Search Graph for River Crossing Puzzle



Repeated States

Repeated states are one of the main concern in searching. There are two effective ways to deal with it:

1. Don't return to a visited state (use hash table)
2. Don't create graphs with cycles

Informed (Heuristic) Search

Assign a cost to each action/arc and define a heuristic function $h(n)$:

$$h(n) = \text{estimated cost of the cheapest path from the state at node n to a goal state}$$

A* Search

Define $f^*(n) = g^*(n) + h^*(n)$ to be the cost of an optimal path going through node n, where $g^*(n)$ is the cost of an optimal path from initial node to node n, and $h^*(n)$ is the cost of an optimal path from node n to a goal node. $f^*(n)$ is hard to know, so we use the estimate $f(n)$.

Always proceed with the node with lowest $f(n)$.

Admissible Heuristics

Def: A heuristic function $\mathrm{h(n)}$ is admissible if $\mathrm{h(n)} \leq \mathrm{h}^*(n)$ for all n.

Thm: If $\mathrm{h(n)}$ is admissible, the A^* algorithm is guaranteed to find an optimal path to a goal node.

Consistent (Monotone) Heuristics

Def: A heuristic function $\mathrm{h(n)}$ is consistent if $\mathrm{h(n)} \leq \mathrm{cost(n, n')} + \mathrm{h}^*(n)$ for all n.

Thm: If $\mathrm{h(n)}$ is admissible, the A^* algorithm is guaranteed to find an optimal path to a goal node.

Dominating Heuristics

Def: Given admissible heuristics $h_1(n)$ and $h_2(n)$, $h_2(n)$ dominates $h_1(n)$ if, for all nodes n,

$h_2(n) \geq \mathrm{h}_1(\mathrm{n})$, and there exists a node n' such that $\mathrm{h}_2(\mathrm{n}') > \mathrm{h}_1(\mathrm{n}')$.

Thm: A^* with $\mathrm{h}_1(\mathrm{n})$ expands at least as many nodes as A^* with $\mathrm{h}_2(\mathrm{n})$, if $\mathrm{h}_2(\mathrm{n})$ dominates $\mathrm{h}_1(\mathrm{n})$.

Complexity for A^*

- A^* is complete, optimal, and optimally efficient (no algorithm can do better)
- Time complexity is exponential:

  $$O\big((b^{\varepsilon})^d\big) = O\big(b^{\varepsilon d}\big)$$

  where ε is the maximum relative error $\big(\mathrm{h}^*(\mathrm{n}) - \mathrm{h(n)}\big)/\mathrm{h}^*(\mathrm{n})$
  b is the branching factor
  d is the depth of the goal node
- Amount of memory needed is a problem

Iterative deepening A^*

Perform iterative DFS and cutoff a branch when its total cost $\mathrm{f(n)}$ exceeds a given threshold. This threshold increases for each iteration, and the new threshold is the minimum cost of all values that exceeded the current threshold.

Since it is a DFS algorithm, its memory usage is lower than A^*. However it might end up exploring the same nodes many times.

Constraint satisfaction problems

A CSP is defined by:

- A set of variables $\{\mathrm{x}_1, \dots, \mathrm{x}_\mathrm{n}\}$
- A set of values for each variable $\mathrm{dom}(\mathrm{x}_1), \dots, \mathrm{dom}(\mathrm{x}_\mathrm{n})$
- A set of constraints $\{\mathrm{C}_1, \dots, \mathrm{C}_\mathrm{m}\}$

A solution to a CSP is a complete assignment to all the variables that satisfies the constraints.

TODO: add more