1 A* Completeness

We previously discussed that A* is optimal, but is it complete?

Recall, an algorithm is **complete** if it guarentees to find a solution when one exists.

In a graph where there are infinitely many nodes, even if the path to the goal is finite, A* may never terminate. See example in slides for such a graph. Such an example generalizes to all algorithms that guarantee optimality. None are able to guarantee the completeness for such a graph.

2 Heuristic Consistency

A heuristic is consistent if the following hold: if one step takes us from n to n', then $h(n) \leq h(n') + cost(n, n')$ where $cost(n, n')$ is the cost to get from n to n'. Similar to the triangle inequality. Can also be written as

$$g(n) + h(n) \leq g(n') + h(n')$$

All consistent heuristics are admissible, but not the other way around. When a consistent heuristic is used in A* search, A* will be optimally efficient. **Optimal efficiency** means that any other optimal algorithm must expand at least the nodes that A* expands. Meaning no other optimal algorithm can be defined which expands fewer nodes. Please see slides for proof.

3 Iterative Deepening A*

Also called **limited-cost depth first A***. You define some cutoff cost c, then only expand to costs below that. If no solution is found, increase cost and repeat. During each iteration, use DFS. It is not guarenteed to find the optimal solution. Only a cost below c, for the c you defined. This can take a lot of time (iterations), but uses less memory.

Addresses memory concerns in a similar way that iterative deepening DFS substitutes BFS.

4 Picking a Heuristic

A heuristic h is said to dominate a heuristic h' is $h'(n) \geq h(n)$ for all n. Generally, the heuristic that dominates will find a solution faster.

An good heuristic is easy to compute, and not much cheaper than solving the original problem.

5 Introduction to N Queens

N Queens is the problem of putting N queens on an N x N chess board such that no queen attacks any other queen on the board.

5.1 Search Formulation

5.1.1 Successors

We can define a successor to be all valid ways of placing a queen in the next column (column 1 is the first column, then 2, then 3, and so on until N).

5.1.2 Goal State

A goal state is a state where all N queens have been placed.

5.2 Search Algorithm

Well, the depth of the solution is the same for all solutions. Because of this, there is no reason to use BFS. DFS is much better. The time complexity is the same as BFS and the space is much lower.

6 Constraint Satisfaction Problem (CSP)

A CSP is defined by:

1. A set of variables, $x_1, x_2, x_3, ...x_n$
2. A domain D for each variable x_i
3. Constraints $c_1, c_2, ...c_m$

A constraint is specified by:

1. A subset of variables
2. All the allowable joint assignments to those variables

Goal: find a complete, consistent assignment

6.1 Graph Coloring

Given a graph G, and a number of colors n, try to find a matching from colors to vertices such that each vertex can be assigned a color, and no two adjacent vertices have the same color.

6.2 Meeting Assignments

We are given a list of meetings, and a list of days. The constraints are of the form (meeting1, meeting2), and are interpreted to mean that meeting 1 and meeting2 cannot occur on the same day. How can this be reduced to graph coloring? Let the days represent the colors, make each meeting a node in the graph, and draw an edge between any meeting1 and meeting2 that appear in the constraints (this enforces that they cannot be matched to the same color, or in this case, occur on the same day).

6.3 Cryptarithmetic Puzzles

See examples from slides to remember that a cryptarithmetic puzzle is. Introduces the concept of auxiliary variables. Auxiliary variables are variables introduced into the problem, despite not being originally defined, to help with defining constraints.

7 How to Solve CSPs

7.1 State Representation

State: define which variables are assigned, and which are unassigned, and for those assigned, what they are assigned to. This can be used to constitute a state.

7.2 Assigning Variables

CSPs satisfy **commutativity**, meaning the order in which variables are assigned does not matter. They can be applied in any order.

7.3 Tree Size

If you have d variables which need to be assigned, and b possible values that each variable can be, then the number of nodes in the search tree is proportional to b^d.

7.4 Searching

Can recursively call search, each time assigning a new variable. If success, return successful assignment. If fail, set next variable to a different value at the previous level. Once all assignments have been tried, return global failure. See slides for more in depth pseudo code.

7.5 Optimization

The general idea is to proactively determine which assignments are possible/impossible under our current assignment at each step, and is there a more efficient way to search through them?