Name: Toby Chappell

Homework #1
(Search Problem Set) Due: 2/25/20 (11:30am)

I. Let **A** be a mechanical assembly product made of *n* parts. In the following, we more specifically consider the 5-part assembly shown in Figure 1 (where each part has a distinct grey level and is identified by a digit between 1 and 5).

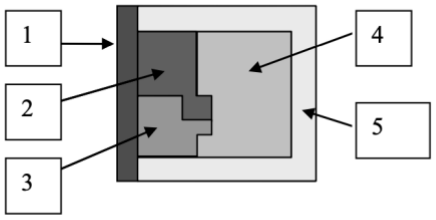


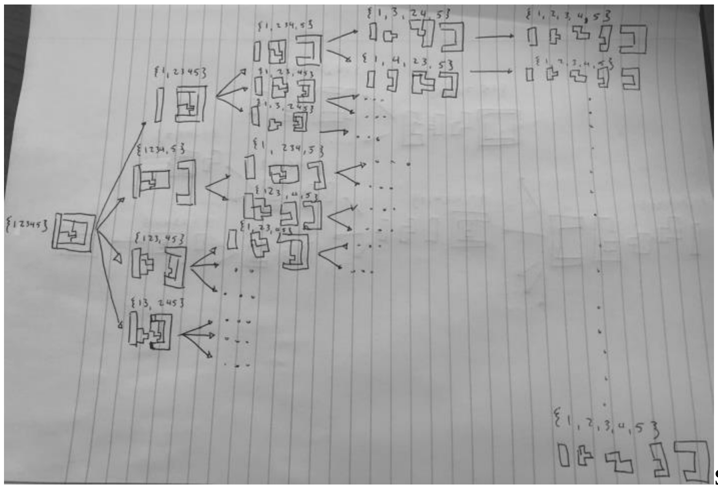
Figure 1: 5-part assembly product

We are interested in two tasks: first, disassembling **A**, next reassembling it.

A *subassembly* of **A** is any individual part of **A** or any collection of $1 < q \leq n$ parts in the same relative positions as in **A**. A legal *disassembly operation* separates a sub-assembly of **A** into two sub-assemblies by moving them far away from each other without collision. A legal *assembly operation* is a motion that brings two separated sub-assemblies of **A** into a sub-assembly of **A** without collision.

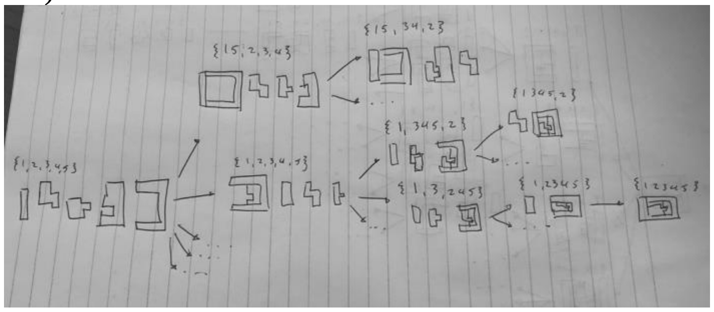
A *state* of the disassembly or assembly problem is a collection of sub-assemblies that contain *n* distinct parts in total. E.g., in the example of Figure 1, {123,45} describes a state made of two sub-assemblies, respectively made of parts 1, 2, and 3, and of parts 4 and 5. Similarly, {12,34,5} describes a state made of three sub-assemblies, respectively made of parts 1 and 2, of 3 and 4, and of 5. But {12,45} is not a state, since part 3 is not used in any of the sub-assemblies. {123,35,4} is not a state either, because part 3 is used twice

1. In the example of Figure 1, what are the initial and goal states of the disassembly problem? Of the reassembly problem?
    a. Disassembly:
        i. Initial state: {12345}
        ii. Goal state: {1,2,3,4,5}
    b. Reassembly:
        i. Initial state: {1,2,3,4,5}
        ii. Goal state: {12345}
2. Show the full search tree that can be generated to solve the disassembly problem. (Feel free to draw by hand and scan in.) Beside each node of the tree, indicate the corresponding state. Are all the leaves of this tree labeled by the goal state? Why? Would your answer be the same for any assembly product?

a. s

b. All of the leaves are labeled by the goal state since there is no legal disassembly that can cause the tree to get “stuck.” This is not true about the assembly product.

3. Show in a similar way a portion of the search tree for the reassembly problem? Are all the leaves of this tree labeled by the goal state? (Note that you don’t have to show the entire tree.)

a. 

b. Not all the leaves are labeled by the goal state meaning there are cases where the tree gets “stuck” (ie. {15,234}).

4. Given the above definition of the legal disassembly and assembly operations, is it always the case that the reassembly search tree is at least as large as the disassembly search tree? Why? Is this an indication that disassembling a product is easier than to reassembling it?

a. The reassembly search tree is at least larger since the assembly the reverse every legal disassembly is a legal assembly. In addition, there are legal assemblies that are not legal disassemblies making the assembly tree larger. This means it is harder to disassemble a product than to reassemble it.

II. Let D be the domain of all humans, dead and alive. For each element of D, we know his/her parents and his/her children. A human **a** claims to be a descendant of **b**.

1. Formulate the problem of verifying **a**’s claim as a search problem: What is the state space, the initial state, the successor function, and the goal state? Is there a single possible formulation?

a. State Space: Ancestors of **a**

b. Initial State: human **a**

c. Successor Function: Check a generation above current human

d. Goal State: human **b**

   e. There are multiple formulations since you can start at **b** and prove **a** is a descendant as well

2. On the average, which should be the easier way to verify **a**’s claim: by showing that **b** is one of **a**’s ancestors, or by showing that **a** is one of **b**’s descendants? Why?
   a. Showing that **a** is one of **b**’s descendants since there are fewer descendants of **b** than ancestors of **a**
3. Which search technique – breadth-first, depth-first, bi-directional, iterative deepening – would you use? Why?
   a. Breadth-first search so the search would check each generation before checking the next generation
4. Assume that the dates of birth of **a** and **b** are given (but not those of other humans). How would you use this information to help the search?
   a. It would be possible to approximate the generational gap and add it to the path cost at each step and once the gap is greater than the difference between the dates of birth, you would begin searching along a different branch

III. Express the following problem as a search state problem:

“Three jugs are filled with water. Neither has any measuring marker on it. Each can be fully or partially emptied in a drain or another jug. The jugs respectively measure 12, 8, and 3 gallons. One needs to measure out exactly one gallon.”

In particular, give a description of the state space (no need to list all possible states), the initial state, the goal test, and the successor function. For the successor function it is not necessary that you write every possible “state→state” combination, but you should make it clear how one would derive the successor states of any given state. Note that I do not ask you to describe a solution of the problem!

- State Space: Every possible combination of jugs filled with differing amounts of water
- Initial State: Each jug is filled with water
- Successor Function: Empty jug into drain or pour a jug into another jug
- Goal State: 1 jug measuring 1 gallon

IV. Construct a search tree of uniform branching factor b and uniform depth d for which it is *possible* that depth-first search uses more memory than breadth-first search. (The depth of a node in a search tree is the length of the path from the root of the tree to N. In particular, the depth of the root is 0.)

Memory of Search Algorithms:

- DFS = O(d)
- BFS = O(b^s) where s is the depth of the shallowest goal node

Therefore, DFS should use more memory than BFS when b^s < d, or, a tree with small branching factor and a shallow goal node should require more memory for DFS in comparison with BFS.

Is there any tree and distribution of goal nodes for which depth-first search *always* requires more memory than breadth-first? Explain briefly. To answer this question, recall that depth-first search assumes no intrinsic ordering of the successors of a node; it may pick any ordering.

There is no case in which DFS always requires more memory than BFS since DFS can go straight to the shallowest goal node (random ordering).