Chapter 3

Search

3.1 Basic Concepts

Definition 22 (Solution)
An assignment of values to variables which satisfies some stopping criterion (e.g., a goal predicate).

To be sure, *solution* is used in an ambiguous way. Depending upon context it may mean either of:

- **Optimal solution**: A solution which maximizes some evaluation criterion or function (as in the Df above).
- **Candidate solution**: An assignment of values to variables which may or may not be optimal or matching a goal; indeed, candidate solutions may even violate hard constraints, and thus fail to be proper solutions.

In other words, solution most generally means any point in the state space of the problem.

Definition 23 (State)
For a system identified by a set of random variables, a state is a joint assignment of values to those variables.

Or, an instance of some representation which naturally generalizes all possible (optimal) solutions.

Examples:

- Board games: the position on the board, plus whose turn it is.
- TSP complete solution space search: a permutation.
- TSP partial solution space search: a permutation over a subset of cities.
- SAT complete solution space search: a truth assignment to the Boolean variables.
- SAT partial solution space search: a truth assignment to a subset of the Boolean variables.
The *state space* of a problem means the space of all possible joint assignments of values to variables.

**Definition 24 (Search Space)**
*The space through which a search algorithm searches.*

The search space is *typically*, but not invariably, the same as the state space for a problem. That is, to find, say, an optimal point in solution space we may search through exactly the space of possible instantiations of the variables. It is, however, possible to do problem solving by searching some other space. For example, some of the candidate solution space may be ruled out as a possible location for the optimal solution, and so never searched. For another example, a search algorithm may repeatedly revisit a given state; in such a case, the search space may be infinite even when the state space is finite. (A simple-minded breadth-first search of the Missionaries and Cannibals problem may be like this, for example.) In general, these space are the same, or may be considered so for this subject.

### 3.2 NFL Theorem

**Definition 25 (Bias)**
*A search is **biased** if it favors some kinds of candidate solutions over others.*

**Examples:**

- *Simplicity bias* (Ockham’s Razor). Candidate solutions which are simpler are favored over those that are complex.
  
  - E.g., if simpler solutions are examined *first*, then they have the first chance to be found satisfactory.
  
  - If there are infinitely many representations, some simplicity bias is likely to be essential, as there will probably be no limit to the complexity of candidate solutions.

- *Heuristic biases.* For example, preferring solutions which have worked on similar problems (reasoning by analogy). Greedy search prefers to look at solutions which score highest on a heuristic evaluation function.

**Definition 26 (Unbiased search)**
*A search is **unbiased** if it does not favor any candidate solutions over any others.*

The simplest way to implement an unbiased search is to search exhaustively.

**Theorem 4 (No Free Lunch)** *When optimizing reward functions (minimizing cost functions), there’s no such thing as a free lunch. (Wolpert and Macready, 1997, IEEE Transactions on Evolutionary Computation)*

Another way of putting this is:

> Any bias introduced in search will favor some solutions and disfavor others.

If the solution to a problem you are working on is discriminated against, your biased search will be suboptimal; another search will be more efficient.
3.3 Some Types of Search

**Definition 27 (Local Search)** Local search begins from an arbitrary state in the search space and looks for an improvement in the neighborhood of that state, until no improvement can be found. (Iterative improvement)

A local search is simply a search in which at each step only a local subset of the search space is to be examined. The best option available in this subset then forms the basis for the next step, and the algorithm keeps going until it finds a good enough solution.

**Definition 28 (Best-First Search)** Best-first search attempts to improve upon the current state by minimizing an evaluation function.

There isn’t much difference in meaning between best-first and local search. They both generalize greedy search. Greedy search is that kind of best-first search which minimizes the heuristic evaluation function $h(x)$, where $x$ is a candidate solution.

Other best-first searches include:

- Tabu search, which is basically greedy, but may miss single step improvements because they are on the tabu list.
- Dijkstra’s algorithm minimizes $c(x)$. This is the sum of the cost to the current state and the cost from the current state to $x$.
- $A^*$ minimizes $f(x) = c(x) + h(x)$.

3.4 TSP Dynamic Programming

```plaintext
(1) INT TSP (L[,], P[,])
(2) INT g[,], n ← dim(L);
(3) FOR i = 1 TO n g[i, 0] ← L[i, 1];
(4) FOR k = 1 TO n − 2
   (5) [FOR ALL S ⊆ V \ {1} S.T. |S| = k
       (6) [FOR i ∉ S ∪ {1}
           (7) g[i, S] ← min_{j ∈ S} (L[i, j] + g[j, S \ {j}]));
           (8) P[i, S] ← THAT j;]]
       (9) g[1, V \ {1}] ← min_{j ∈ V \ {1}} (L[1, j] + g[j, V \ {1},j]);
   (10) P[1, V \ {1}] ← THAT j;
(11) RETURN g[1, V \ {1}];
```

Return value is the min length; returned $P$ reports the min path.

Recall that dumb brute force (exhaustive search) for TSP has time complexity $(n−1)!/2$, which is superexponential. Have we done better? Analysing the inner loop we see the statements are executed this many times:
(5) $C_{k}^{n-1}$  
(6) $n - 1 - k$  
(7) $k$

So, the time complexity is:

$$T(n) = \sum_{k=1}^{n-2} k(n - 1 - k)C_{k}^{n-1}$$

Since

$$(n - 1 - k)C_{k}^{n-1} = \frac{(n - 1)(n - 2)!}{(n - 2 - k)!k!} = (n - 1)C_{k}^{n-2}$$

$$T(n) = (n - 1) \sum_{k=1}^{n-2} kC_{k}^{n-2}$$

And since

$$\sum_{k=0}^{n} C_{k}^{n} = 2^n$$  (Equal to the no of subsets of a set of size n)

So,

$$n2^{n-1} = n \sum_{k=0}^{n-1} C_{k}^{n-1}$$

$$= \sum_{k=1}^{n} nC_{k-1}^{n-1}$$  (rewrite indices)

$$= \sum_{k=1}^{n} kC_{k}^{n}$$  (Purdom & Brown, 3.55)

Hence,

$$T(n) = (n - 1)(n - 2)2^{n-3}$$

So, finally (using $O$ properties from §2.2.1),

$$T(n) \in \Theta(2^{n-3})$$

### 3.5 Exercises

#### 3.5.1 Vocabulary

Before you begin these exercises, you should make sure you understand the meaning of the words and phrases in this list. Write down brief definitions for each one, and discuss them in your tute. You may need to use a dictionary, the textbook, or search the web to find a definition – if you do search the web, make sure you look at more than one page.

- breadth-first search
3.5.2 Search problems

(i) Define the state spaces for:
   
   (a) Scheduling lecture theatres
   (b) The Euler cycle problem
       
       An Euler cycle is a circuit through a graph that includes every edge exactly once.
   (c) The maximal clique problem
       
       An clique is a fully connected subgraph. The maximal clique problem is to find the largest such subgraph in a graph.
   (d) Process scheduling for a multitasking operating system

(ii) Define local neighborhoods for the same.

(iii) Define perturbation functions for those neighborhoods.

(iv) What are the main differences between local search and exhaustive search? When is a local search more appropriate?

(v) Define heuristic evaluation functions for these problems:
(a) Getting dressed in the morning. I have a wardrobe full of clothes, and they don’t all match. For example, my red skirt looks very good with my red blouse and fairly good with my black tunic, but looks awful with my green T-shirt. Some of my clothes suit formal occasions, some are casual, and some can be worn to either. Some of my clothes are better for warm weather and some are better for cool weather. I usually start by grabbing a random (clean) garment and then rummaging around for something else to wear with it. (Is this a complete-solution approach or a partial-solution approach?)

(b) Turn-based strategy games such as chess, checkers or go.

(c) Pathfinding in computer games. Computer-controlled players need to be able to chase human-controlled players around areas with various different kinds of terrain obstacles.

(d) Computerized real-time strategy games such as the Warcraft and Command and Conquer series. In these games, the computer’s AI must control units for building, exploration and resource gathering as well as combat with the player. (Can you think of some factors that limit the quality of the computer’s strategy?)

(e) Exploring the solar system. Last year, the Huygens probe was dropped on Titan, one of Saturn’s moons. The few images that the probe transmitted back to Earth showed a surprisingly Earth-like landscape, with evidence of mud, rivers and lakes. It is thought that Titan might be the only place in the solar system other than Earth where it rains, although the rain on Titan would be methane. Many planetary scientists would love to send a robotic rover mission to Titan to discover more about its geology, just as the Pathfinder, Spirit and Opportunity rovers have explored Mars. Titan is so far away from Earth that even at its closest approach, radio waves take 66 minutes to travel one way. That would mean that Earth-based operators would have to wait over two hours to see results from their commands – too long to control a rover in real time. Therefore the rover would have to have a sophisticated artificial intelligence system that would allow it to plan its immediate actions and respond to local conditions, such as rain and mud, while allowing its operators to specify long-term goals. Develop a local search model for this problem that allows the rover to choose between different actions based on the long-term goals specified by the operators and the local conditions revealed by the probe’s sensors. Note: There are many ways to design such a control program, and local search might not be the best way. Can you think of other approaches that might work well?

3.5.3 Local Search

(i) Recall graph algorithms from second-year Algorithms and Data Structures. Finding a minimum spanning tree can be thought of as a search problem.

(a) Devise a model for this problem. What is the search space of your model?

(b) Here is high-level pseudocode for Prim’s Algorithm:

```plaintext
create a tree containing a single vertex
create a set containing all the edges in the graph
loop until every edge in the set connects two vertices in the tree
  remove from the set the smallest edge between
```
a vertex in the tree and a vertex not in the tree
add that edge to the tree

Is Prim's Algorithm a greedy algorithm?

(c) Here is high-level pseudocode for Kruskal's Algorithm:

create a spanning forest $F$
create a set $S$ containing all the edges in the graph
while $S$ is nonempty
    remove the smallest edge from $S$
    if that edge connects two different trees
        add it to $F$
    else discard that edge

Is Kruskal's Algorithm a greedy algorithm?

(ii) Explain why Dijkstra's algorithm (or $A^*$) is not greedy. What ingredient in the evaluation function is incompatible with (pure) greed?

(iii) Some search problems are harder than others. What are some factors that affect the difficulty of a search problem? See if you can come up with some examples of problems that are affected by these factors.