Lecture 1: Discrete search

1 Decision-making problems

Sometimes we know how to state a problem, but don't have an immediate way to compute the answer. This class is about computational approaches to expressing decision-making problems and finding optimal or good or satisfactory solutions.

Formulating problems will be at least as important to us as solving them.

We will consider *optimization problems*, which are specified with two components

- **solution space** S: the set of possible answers
- **evaluation criterion** f: a function from elements of the solution space to real numbers

The goal is to find an optimal solution, which is any $s \in S$ satisfying

$$\forall s' \in S . f(s) \geqslant f(s')$$

A useful subclass are *satisfaction problems*, in which f has the range $\{0, 1\}$, and we are satisfied with any solution s such that $f(s) = 1$.

If we don't know anything more about the problem, we're in big trouble. There's nothing better to do than try all the s's (if they are enumerable) or randomly guess (if not).

Throughout the course, we'll explore different kinds of assumptions we can make about S and f, and how we can take algorithmic advantage of those assumptions to solve the problem more effectively.

Here are some of the properties that will affect our approach:

- Whether S is *discrete* or *continuous*;
- Whether S is has an *atomic*, *factored*, or *structured* representation;
- If S continuous, whether f is *linear* or *convex* or more complex;
- If S is continuous, whether its boundaries are *linear* or *convex* or more complex;
- Whether S can be interpreted as a set of *paths* through a state space X, governed by choices of actions from an action space A (sometimes called a *sequential decision problem*);
- If S is a set of paths, whether $f(s) = f([x_0, a_0, \ldots, x_{n-1}, a_{n-1}, x_n])$ is *additive*; that is, that there is some c such that $f([x_0, a_0, \ldots, x_{n-1}, a_{n-1}, x_n]) = \sum_{i=0}^{n-1} c(x_i, a_i)$;
- If S is a set of paths, whether the state x_{i+1} depends *deterministically*, *nondeterministically*, or *probabilistically* on x_i and a_i;

- If S is a set of paths, whether the state x_i is is known (if it is not known, the problem is *partially observable*, and we might know a set containing x_i or a probability distribution over X);
- If S is a set of paths, whether the dynamics governing the dependence of x_{i+1} on x_i and a_i is known (if not, then it is called *reinforcement learning* problem).

There are many other possible interesting properties, but we will concentrate on these in this course.

2 Discrete space, atomic representation

If we don't assume anything, there's not much we can do.

Assume *neighbor function*, so that $N(s) \subset S$. Implicitly, the idea is that there is some kind of smoothness; that neighbors of s will have similar f values.

2.1 Hill climbing

Algorithm:

```
hillClimb(S, f, N):
    s = randomDraw(S)
    loop:
        bestNeighbor = argmax(N(s), f)
        if f(s) >= f(bestNeighbor):
            return s
        s = bestNeighbor
```

Improved by random restarts

When there are too many neighbors, we can select one at random and move there if it's an improvement:

```
hillClimb(S, f, randomNeighbor):
    s = randomDraw(S)
    loop until tired:
        n = randomNeighbor(s)
        if f(n) > f(s):
            s = n
    return s
```

2.2 Simulated annealing

Great paper: "Optimization by Simulated Annealing," S. Kirkpatrick, C.D. Gelatt, Jr., and M. P. Vecchi, *Science*, volume 220, number 4598, 1983.

Like hill-climbing, but get out of local optima by sometimes making steps that don't improve the objective. Gradually 'anneal' the system by making it less and less likely to take non-improving moves. Derived from a statistical mechanics simulation algorithm due to Metropolis.

Usually described as *minimizing* energy. So we'll try to minimize f.

Pseudocode taken from Wikipedia (9/9/2010):

```
sa(S, f, randomNeighbor, temp, kMax):
    s = randomDraw(S)
    e = f(s)
    k = 0
    while k < kMax:
        sNew = randomNeighbor(s)
        eNew = f(sNew)
        if p(e, eNew, temp(k / kMax)) > random():
            (s, e) = (snew, enew)
        k += 1
```

Probability of accepting a proposed move:

$$p(e, e', T) = \begin{cases} 1 & \text{if } e' < e \\ e^{(e-e')/T} & \text{otherwise} \end{cases}$$

If e' is much higher than e, then the move is much less likely to be accepted (remember that we're trying to minimize e here); if T is high, then the move is more likely to be accepted.

This version takes a fixed budget of iterations `kMax` and selects a temperature as a function of the fraction of those iterations that have already occurred.

Kirkpatrick et al. fix a temperature and run at that temperature until it has had some number (e.g. 10) successful moves, and then decrease it.

SA is particularly appropriate for domains where:

- There is no perfect solution, but it's possible to find solutions much better than randomly generated ones;
- many good near-optimal solutions, so stochastic search ought to find one; and
- no one of the near-optimal solutions is significantly better, so it's not worth spending a lot of time looking for the optimum.

Even simpler: *threshold acceptance* (Dueck and Scheuer 1990). Accept all improving moves; accept other moves if $e - e' < D$. Decrease D over time.

Example: Partitioning a circuit into two chips

You have N circuits, to be partitioned into two chips. Each pair of circuits i, j has a_{ij} wires between them. You want to:

- Minimize the number of wires running between the chips
- Roughly balance the number of circuits on each chip

Solution space: $\langle \mu_1, \ldots, \mu_N \rangle$.

$$\mu_i = \begin{cases} 1 & \text{circuit } i \text{ is on chip 1} \\ -1 & \text{otherwise} \end{cases}$$

Number of wires between chips:

$$N_c(\mu) = \sum_{i<j} \frac{a_{ij}}{4}(\mu_i - \mu_j)^2$$

Imbalance:

$$B(\mu) = \left(\sum_i \mu_i\right)^2$$

Objective (to be minimized). Trade off criteria with λ.

$$f(\mu) = N_c(\mu) + \lambda B(\mu)$$

Random $N_c \approx 6000$; Hill-climbing $N_c \approx 1400$; annealed $N_c \approx 600$.

3 Example domains

- Best kind of car to buy
- Best town in the USA to live in
- Best path to drive to the Empire State building
- Traveling salesdroid problem
- Bin packing
- Route for vacuuming robot
- Route for surveillance plane
- Space telescope viewing schedule
- Object recognition (based on images or point clouds)