Non-deterministic algorithms & metaheuristics

Non-determinism

- Almost everything we've looked at has the same result and run characteristics (processor cycles, memory use) every time we run it.

- What can we say about those problems?
    - The solution space is finite, predictable, and within our run parameters (how much memory we have, how long we're willing to wait, etc.)

- Those are *deterministic* algorithms.
    - We may not know the result before we run, but it is pre-determined. [Islam: it is written]

Non-determinism (2)

- For instance, sorting a list of N objects. How many possible permutations of N objects are there?
  - There are *N!,* i.e. *factorial(N),* permutations (assuming there are no repetitions).
  - Before we start sorting, our list is in *one* of those *N!* permutations—we just don't know which one—and how to get to the one we want—until we run the algorithm.
- In other words, our "solution space" for sorting is of order *N!*
  - Or, to put it another way, we have N! bits of "entropy" when we start out with a sorting problem.
  - Do any of you know how to estimate *N!*?

Non-determinism (3)

- The standard method to estimate *n!* is Stirling's formula:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

- As you can see, this is *exponential.* Not nice.
- It's perhaps more helpful to look at the number of bits of entropy (*lg n!*):
  - *~ n(lg n - lg e) + 1/2 lg (2πn)*
- Example: n = 20, n! = 2,432,902,008,176,640,000 where we'd expect 61 bits of entropy (61.077 to be precise)
- Stirling gives: 20(4.322-1.443) + 3.487 = 61.071

Non-determinism (4)

- Anyway, where were we? If our sorting algorithm does *n lg n* swaps, and each swap reduces the search space by 1 bit, then we should reduce the entropy of our solution space to something like:
  - $\sim -n \lg e + 1/2 \lg (2\pi n)$
- Which, for n=20 comes to: -25.4 which means we have zero entropy: the problem is solved.

Non-determinism (5)

- Does this give us a clue how to sort better? What happens to our total entropy if we divide the list into two parts—obviously, the entropy is unchanged but each half individually only has this much entropy:
  - $\sim (n-1)(\lg n - 2 \lg e) + 1/2 \lg (2\pi(n-1))$
  - with a total of:
  - $\sim 2(n-1)(\lg n - 2 \lg e) + \lg (2\pi(n-1))$
- Which, for n=20 comes to: 43.6 bits
  - **We just "saved" 17.5 bits!!**

Non-determinism (6)

- Strictly speaking, non-deterministic algorithms are supposed to guarantee getting to the correct solution but may take one of several different paths to get there.
- What I really want to talk about are nature-inspired *metaheuristics:*
  - Particle Swarm Optimization
  - Evolutionary Algorithms
  - Simulated annealing
  - *etc.*

NP and NP-Complete

- A search problem is one where the time need to verify correctness is bounded by a polynomial on N, the size of the problem.

- NP problems are all the search problems that we aspire to solve (but don't necessarily have a solution).

- The P problems are a subset of NP and are the problems we know how to solve in polynomial time.

- NP-Complete problems are, essentially, the problems for which we have not found a solution in P. [Don't worry too much about the exact definition].

Metaheuristics

- What if our solution space is just <u>too</u> big or we just don’t know how to solve it deterministically? That’s to say, our problem is NP-Complete [more or less…]
- We cannot solve it by a deterministic algorithm therefore we can try a *metaheuristic* algorithm.
- Such algorithms:
  - Are not guaranteed to find the <u>best</u> solution;
  - Are non-deterministic because they are *stochastic* —they rely on random processes;
  - May not converge on a solution at all: e.g. Newton-Raphson method.

Evolutionary Algorithms

- Evolutionary Algorithms:
  - Model evolution, primarily based on survival-of-the-fittest
    - (although *Darwin*, my own evolutionary framework also has sexual selection)
  - Therefore, of paramount importance is the fitness function;
  - The essential steps are:
    - creates a colony of organisms, each based on a genotype;
    - evaluate the genotype by applying the fitness function;
    - survivors are allowed to reproduce; non-survivors are culled;
    - each new organism has a genotype based on its parent(s) but with mutation and/or "crossover" applied;
    - loop back to second step;
    - iterate until an organism with the required fitness value is found.

Evolutionary Algorithms (2)

- Example problem:
  - The Traveling Salesman problem:
  - There are some amazingly good OR-based solutions but it's also amenable to evolutionary computation.