Practical Issues with Simulated Annealing

Cost functions must be carefully developed, "fractal" & smooth

Example: if (critical_path > required) cost += 1000

\[ \text{vs. cost} += \max((\text{critical_path} - \text{required}), 0) \]

Cost functions must be FAST - called millions of times

Best: easy to calculate deltas, such as just updating length of changed nets
if updating nets connected to swapped nodes, beware of shared nets
Practical Issues with Simulated Annealing

Balancing multiple objectives complex, requires LOTS of testing
Area efficient vs. performance efficient implementations

\[ \text{Cost} = c_1 \times \text{wirelength} + c_2 \times \text{delay} + c_3 \times \text{power} + c_4 \times \text{cross}\text{talk} + \ldots \]

As few elements in the cost function as possible.

Cooling schedule also takes LOTS of testing. Some suggestions (VPR-ish):
1.) Start at a temperature 20*measured standard deviation of random swaps
2.) Make say 10xNumNodes moves per temperature
3.) Decrease temp based on acceptance rate \( \alpha \) of moves at temperature
   \[ \alpha > .96 \rightarrow .5x \]
   \[ .96 \geq \alpha > .80 \rightarrow .9x \]
   \[ .80 \geq \alpha > .15 \rightarrow .95x \]
   \[ .15 \geq \alpha \rightarrow .8x \]
4.) Stop when it no longer accepts any bad moves over a couple iterations.
Simulated Annealing Example

You are teaching a group project class for $N \geq 14$ students. You ask them to tell you people they would love to work with, and people they would hate. Have an annealer form teams of 4-5 students each.

Initial group formation:

Randomly form as many groups of 4 as we can.
Randomly assign leftovers to a group of 4 each.

Move function:

90% Randomly swap 1 member each from 2 random groups
5% Randomly dissolve a 4-group, add members to 4 other 4 groups
5% Randomly pick 4x 5 groups, take a random member from each

Cost Function:

For each pair of group members

- If hate, $\text{cost} += 1$
- If love, $\text{cost} -= 1$
Genetic Algorithm

"Breed" configurations, with survival of the fittest

Start with sample population

Evaluate with a cost function, eliminating less fit configurations

Generate new configurations from surviving configurations, and repeat
Genetic String

Express a placement as a string of characters, numbers, etc.

Node IDs in string, with position indicating location

\{ B F D G E H A I C \}
Genetic Algorithm Operators

Crossover
Take two strings, split at random point, and join left side of A with right of B

Parent 1: \{\text{BFDGE} \mid \text{HAIC}\} \times \{\text{GDBFE} \mid \text{CHIA}\} \rightarrow \{\text{BFDGE CHIA}\}

Note: each node should appear in the string EXACTLY once

For any repeats in right side, replace with a missing cell

Parent 1: \{\text{BFDGE} \mid \text{HAIC}\} \times \{\text{GCAFE} \mid \text{DHIB}\} \rightarrow \{\text{BFDGE} \leq \text{HIA}\}

\leq \text{A}

\leq \text{A}
Genetic Algorithm Operators (cont.)

Mutation
Randomly swap two cells (similar to simulated annealing move function)

Child:
\{EDIGHFBCA\} \rightarrow \{ECIGHFBDA\}

Inversion
Randomly select subsequence of string and reverse it

Child:
\{G|AFEDBHCI\} \rightarrow \{GHBDEFA\}
{CI}
Genetic Algorithm Algorithm

Create initial population of size generation_size;
for (generations = 0; generations < requested_generations; generations++) {
    for (l = 0; l < generation_size; l++) {
        choose parents randomly, weighted by population member's fitness;
        offspring[l] = crossover(parent1, parent2);
    }
    randomly apply mutations to offspring, with probability prob_mutation;
    randomly apply inversions to offspring, with probability prob_inversion;
    select new population from current population & offspring, based on fitness;
}
return most fit population member;