Local Search

- Remember local search works by picking an initial solution and making local moves that reduce the cost of the solution.
- The key problem in local search is to find good neighbours.
  - Which are easy to compute,
  - not to few and not to many
- A common technique is to partly solve the problem and when picking the initial solution and pick moves that preserve the partial solution. (Think of the partition problem).

Local and Global Minima

- The problem is that is you keep picking local moves that improve the solution you end up in local minima.
- We are looking for global minima, that is a solution which minimizes the cost.
- The problem is we sometimes get stuck in at a local minima. That is somewhere where there are no local improving moves that reduce the cost.

Local and Global Minima

- How do we avoid local minima?
- Start again with a different initial solution
- Allow increasing moves to be made to move us out of a local minima.
- There are many techniques but we will look at two:
  - TABU Search
  - Simulated Annealing.

Tabu Search

- Tabu search avoids cycles in the search space.
- Ideally the Tabu search algorithm would be as follows:

procedure Full—Tabu
Generate initial solution s^*
$T \leftarrow \emptyset$
while Some stopping condition not satisfied **do**
  $N \leftarrow LegalNeighbours(s^*)$
  $N' \leftarrow N \setminus T$ (remove from N any items found in the tabu list)
  $s^* \leftarrow selectBest(N')$.
  $T \leftarrow T \cup \{s^*\}$
end while

Tabu Search

- Obviously the set T can get large.
- The normal technique is to give each member an age and remove items that get too old. The age is often referred to as a "tabu tenure parameter"
- Not that tabu search does not avoid completely local minima, but it does remove cycles as we saw with the N-Queens example at the last lecture.
- With random restarts you should keep the old Tabu list. It has useful information.

Randomized Iterative Improvement

- Tabu search does not let you completely escape local minima
- A strategy is to sometimes allow increasing moves.
- **procedure Randomized–Iterative–Improvement** depends on a parameter *wp*
  Generate initial solution s^*
  while Some stopping condition not satisfied **do**
  $N \leftarrow Legaleighbours(s^*)$
  $u \leftarrow random([0, 1])$
  if $u \leq wp$ **then**
  $s^* \leftarrow PickRandom(N)$
  else
  $s^* \leftarrow selectBest(N)$
  end if
  end while

Randomized Iterative Improvement

- If *wp* is equal to 0 then we get normal local search.
- If *wp* is equal to 1 then we get a random walk.
  - Even with a random walk you'll find the solution eventually, it will just take a very long time.
- This can be combined with Tabu search.

Simulated Annealing

- Simulated Annealing takes its inspiration from physics. It uses a similar process to a liquid cooling down to make crystals.
- It uses the concept of temperature.
- The higher the temperature the higher the probability of accepting a cost increasing move.

Simulated Annealing

- Given a cost function f and a temperature T a current solution s^* and a candidate solution s the acceptance probability is as follows:

$$p_{accept}(T, s^*, s) = \begin{cases} 1 & \text{if } f(s) \leq f(s^*) \\ exp(\frac{f(s^*)-f(s)}{T}) & \text{otherwise} \end{cases}$$

- With a constant temperature T we can define the following algorithm:

Constant Temperature Annealing

Generate initial solution s^*

while Stopping condition not satisfied **do**

  $N \leftarrow LegalNeighbours(s^*)$

  $s \leftarrow PickRandom(N)$

  $u \leftarrow random([0, 1])$

  if $u \leq p_{accept}(T, s^*, s)$ **then**

    $s^* \leftarrow s$

  end if

end while

Simulated Annealing

- To produce perfect crystals you start with a high temperature and cool slowly.
- Simulated Annealing modifies the temperature each step.
- For example start with $T = 10$ and set $T \leftarrow 0.95 * T$ each step. This often works well.
- Determining the initial temperature is not so easy theoretically and depends on the problem.

Inspiration from Biology

- If we look around us at the world we see many organisms adapted to their environment.
- Biology tells us that genetics are responsible for this.
- Every cell has strands of DNA which code a number of genes.
- Each gene is responsible for a number of proteins which are the subroutines that build up a functioning cell.

Inspiration from Biology

- There are two mechanisms that produce variety:
  - Crossover, when a male and a female mate you get some genes from your mother and some from your father.
  - Random mutation, for various reasons bits of the DNA are altered randomly.

Inspiration from Biology

- Darwin coined the term “Natural Selection” other people often use the phrase “Survival of the fittest”
- If you are not well adapted to your environment then there is less chance of survival.
- So less chance of mating, so less chance that you pass your genes on.
- So overtime the bad genes disappear and the good genes dominate.

Inspiration from Biology

- Genetic algorithms try to mimic this.
- A genetic algorithm needs a few things:
  - A notion of a coding of a solution.
  - A crossover function. c. that takes two solutions and combines them to produce a new one.
  - A notion of mutation.

Genetic Algorithm Scheme

For minimizing f.

Generate initial population $S = \{s_1, \ldots, s_n\}$
while Stopping criteria not satisfied **do**
  Select two parent solutions s_i and s_j according to their fitness so, the lower $f(s)$ the higher the probability of selecting it.
  $s' \leftarrow c(s_i, s_j)$
  With a mutation probability mutate s'
  $S \leftarrow S \cup \{s'\}$
  Kill of old member of S with some scheme.
end while

Obviously there is a lot of leeway in the algorithm, lots of choices for you to change.
Finding a good representation and a good cross over function is crucial to make the algorithm work.

Genetic Algorithm Chromosome

- The solution is often represented as a string:
  $0 \quad 1 \quad 1 \quad \ldots \quad 1 \quad 1 \quad 0$
- A common crossover function picks a random point in the two stings and at that points switches from one string to another.
  $0 \quad 1 \quad 1 \quad \ldots \mid 1 \quad 1 \quad 0 \quad \ldots$
  $1 \quad 0 \quad 1 \quad \ldots \mid 0 \quad 0 \quad 1 \quad \ldots$
- To produce the offspring:
  $0 \quad 1 \quad 1 \quad \ldots \mid 0 \quad 0 \quad 1 \quad \ldots$

Example Encoding

- Remember the Knapsack problem. Given n items with weight w_i and value p_i you want to keep the total weight below some constant c while maximizing the value.
- The coding would be n 0/1 values, 1 represents in the sack and 0 represents out of the sack.