Lecture 4: Local/Stochastic Search
Local Search

Given:
• A set of states (configurations) $S = \{X_1, \ldots, X_M\}$
• A function that evaluates each state: $Eval(X)$

Find global maximum: $X^*$ such that $Eval(X^*)$ is greater than all $Eval(X_i)$ for all values of $X_i$
In This Lecture

- Either set of configurations too large to be enumerated explicitly
- Or computation of $Eval(.)$ may be expensive
- Therefore we cannot find the maximum of $Eval(.)$ by simply trying out all states
- Solutions with similar values of $Eval(.)$ are considered equivalent for the problem at hand
- We do not care how we get to $X^*$ (the path), we care only about the description of the configuration $X^*$

Up until now we cared about the path. Now we don't. We're in charge, so we can do that.
Too many configurations to look at them all
Once we find the best configuration, we don’t really care about the path to the solution.

• VLSI layout:
  – $X =$ placement of components + routing of interconnections
  – $Eval =$ Distance between components + %unused + routing length + ?
Real-World Examples

- Scheduling: Given $m$ machines, $n$ jobs
- $X =$ assignment of jobs to machines
- $Eval =$ completion time of the $n$ jobs (minimize)
Example: TSP (Traveling Salesperson)

\[ X_1 = \{1 \ 2 \ 5 \ 3 \ 6 \ 7 \ 4\} \]

- Configuration \( X \) = Path through all the nodes
- \( Eval = \) Length of path
- Size of search space = \((N-1)!/2\)

TSP: You, the salesperson, wants to travel through every city once in the cheapest way possible.

\((N-1)!/2\) is correct because you don’t care about the starting city (hence \((N-1)!\) instead of \(N!\)), and you don’t care about the direction (hence the /2)
Example: SAT (SATisfiability)

\[
\begin{align*}
A \lor \neg B \lor C \\
\neg A \lor C \lor D \\
B \lor D \lor \neg E \\
\neg C \lor \neg D \lor \neg E \\
\neg A \lor \neg C \lor E
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>5</td>
</tr>
<tr>
<td>$X_2$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>4</td>
</tr>
</tbody>
</table>

Configuration: Assignment of true/false to each variable
Eval: Number of clauses satisfied
Real world examples of SAT problems?

Model checking
Mine sweeper
Sudoku
College class scheduling
Example: N-Queens

N^N configurations (One queen per column)

Eval function is number of ways a queen can attack another queen

Find a configuration in which no queen can attack any other queen

What’s Eval() here?

Eval(X) = 5
Eval(X) = 2
Eval(X) = 0
Local Search

1. $X_0 \leftarrow$ Initial state
2. Repeat until we are “satisfied” with the current configuration:
3. Evaluate some of the neighbors in $Neighbors(X_i)$
4. Select one of the neighbors $X_{i+1}$
5. Move to $X_{i+1}$
Search

The definition of the neighborhoods is not obvious or unique in general. The performance of the search algorithm depends critically on the definition of the neighborhood which is not straightforward in general.

1. Evaluate some of the neighborhoods $Neighbors(X_i)$
2. Are we “satisfied” with the current configuration?
3. Evaluate some of the neighborhoods $Neighbors(X_i)$
4. Select one of the neighbors $X_{i+1}$
5. Move to $X_{i+1}$

Ingredient 1. Selection strategy: How to decide which neighbor to accept

Ingredient 2. Stopping condition

Lots of questions to answer
Start at red rectangular thing
Look at your left neighbor X-1, and your right neighbor X+1 and go whichever way is better.
You get stuck in a local optimum, though. Phooey
**Most Basic Algorithm: Hill-Climbing (Greedy Local Search)**

- $X \leftarrow$ Initial configuration
- Iterate:
  1. $E \leftarrow \text{Eval}(X)$
  2. $\mathcal{N} \leftarrow \text{Neighbors}(X)$
  3. For each $X_i$ in $\mathcal{N}$
     
     $E_i \leftarrow \text{Eval}(X_i)$
  4. If all $E_i$'s are lower than $E$
     
     Return $X$
  
  Else
  
   $i^* = \text{argmax}_i \{E_i\}$  
   
   $X \leftarrow X_{i^*}$  
   
   $E \leftarrow E_{i^*}$

Called hill climbing because you always try to go up hill.

Neighbors could be more than just one left and one right. You could look much further.

Problem, however, is getting stuck in local optimum
For SAT, we could say that neighbors are configurations with one variable changed. For TSP, we could say that neighbors are configurations with two edges swapped.
O(N^2) neighbors, because there are (n choose 2) possible pairs of edges to swap
n choose 3 is $O(N^3)$
Issues

Trade-off on size of neighborhood:

• Larger neighborhood = better chance of finding a good maximum but may require evaluating an enormous number of moves

• Smaller neighborhood = smaller number of evaluations but may get stuck in poor local maxima
Multiple “Poor” Local Maxima
No one likes plateaus
To get from $X_{\text{start}}$ to $X^*$ you need to first go down...so $X_{\text{start}}$ is a local optimum.
Memory used: very little...constant amount...pretty much the size of the neighborhood

Remarks

• How much memory is used?
• All we can hope is to find the local maximum “closest” to the initial configuration. Can we do better than that?
Stochastic Search: Randomized Hill-Climbing

• \( X \leftarrow \text{Initial configuration} \)
• Iterate:
  1. \( E \leftarrow \text{Eval}(X) \)
  2. \( X' \leftarrow \text{one configuration randomly selected in Neighbors}(X) \)
  3. \( E' \leftarrow \text{Eval}(X') \)
  4. If \( E' > E \)
     \( X \leftarrow X' \)
     \( E \leftarrow E' \)

Critical change: We no longer select the best move in the entire neighborhood.
One of the best algorithms for SAT
In this algorithm, you don't always go uphill. When you randomly pick a variable, it could cause you to go downhill!
X_i is a variable in the unsatisfied clause

WALKSAT
Iterate until all clauses are satisfied or max iterations:

1. Select an unsatisfied clause
2. With probability p:
   Select a variable x_i at random
3. With probability 1-p:
   Select the variable x_i such that changing x_i will unsatisfy the least number of clauses (Max of Eval(X))
4. Change the assignment of the selected variable x_i
Simulated Annealing

1. \( E \leftarrow \text{Eval}(X) \)
2. \( X' \leftarrow \text{one configuration randomly selected in Neighbors (X)} \)
3. \( E' \leftarrow \text{Eval}(X') \)
4. If \( E' \geq E \)
   \( X \leftarrow X' \)
   \( E \leftarrow E' \)

Else accept the move to \( X' \) with some probability \( p \):
   \( X \leftarrow X' \)
   \( E \leftarrow E' \)

Critical change: We no longer move always uphill. Next question: How to choose \( p \)?


The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.
How to set $p$? Intuition

$E = E(X)$

$E' = E(X')$

$E - E'$ is large: It is more likely that we are moving toward a (promising) sharp maximum so we don’t want to move downhill too much.

$E = E(X)$

$E' = E(X')$

$E - E'$ is small: It is likely that we are moving toward a shallow maximum that is likely to be a (uninteresting) local maximum, so we like to move downhill to explore other parts of the landscape.
Choosing $p$: Simulated Annealing

- If $E' \geq E$ accept the move
- Else accept the move with probability:
  \[ p = e^{-\frac{(E - E')}{T}} \]
- Start with high temperature $T$ and decrease $T$ gradually as iterations increase ("cooling schedule")

This temperature stuff is they metaphor-y part.
Simulated Annealing

\[ X \leftarrow \text{Initial configuration} \]
\[ T \leftarrow \text{Initial high temperature} \]

Iterate:

1. Do \( K \) times:
   \[ E \leftarrow \text{Eval}(X) \]
   \[ X' \leftarrow \text{one configuration randomly selected in Neighbors}(X) \]
   \[ E' \leftarrow \text{Eval}(X') \]
   If \( E' \geq E \)
      \[ X \leftarrow X'; E \leftarrow E'; \]
   Else accept the move with prob \( p = e ^{-\frac{(E - E')}{T}} \):
      \[ X \leftarrow X'; E \leftarrow E'; \]
2. \( T \leftarrow \alpha T \) \( \infty \)

alpha < 1
As \( t \) goes down, probability \( p \) goes down. And as \( p \) goes down you take fewer chances.
Stopping condition depends on the problem.
Simulated Annealing

\( X \leftarrow \text{Initial configuration} \)

\( T \leftarrow \text{Initial high temperature} \)

Iterate:

1. **Do \( K \) times:**
   - \( E \leftarrow \text{Eval}(X) \)
   - \( X' \leftarrow \text{one configuration randomly selected in Neighbors}(X) \)
   - \( E' \leftarrow \text{Eval}(X') \)
   - If \( E' < E \):
     - \( X \leftarrow X' \)
     - \( E \leftarrow E' \)
   - Else accept the move with prob \( p = e^{-(E - E')/T} \):
     - \( X \leftarrow X' \)
     - \( E \leftarrow E' \)

2. \( T \leftarrow \alpha T \)

Iterate a number of times keeping the temperature fixed

Progressively decrease the temperature using an exponential cooling schedule:

\( T(n) = \alpha^n T \) with \( \alpha < 1 \)

\( T = 0 \rightarrow \text{Greedy hill climbing} \)

\( T = \infty \rightarrow \text{Random walk} \)
Basic Example

Starting point: We move most of the time uphill

Iteration 150: Random downhill moves allow us to escape the local extremum

$T = 15.6975$

$T = 12.677$
Basic Example

Iteration 180: Random downhill moves have pushed us past the local extremum

Iteration 800: As $T$ decreases, fewer downhill moves are allowed and we stay at the maximum
Basic Example

Note that larger deviations from uphill search are allowed at high temperature.
Where does this come from?

- If the temperature of a solid is $T$, the probability of moving between two states of energy is:
  $$e^{-\Delta\text{Energy}/kT}$$

- If the temperature $T$ of a solid is decreased slowly, it will reach an equilibrium at which the probability of the solid being in a particular state is:

- Probability (State) proportional to $e^{-\text{Energy(State)}/kT}$

- Boltzmann distribution $\rightarrow$ States of low energy relative to $T$ are more likely

- Analogy:
  - State of solid $\leftrightarrow$ Configurations $X$
  - Energy $\leftrightarrow$ Evaluation function Eval(.)
A TSP Example

N = 13 nodes (in a circle)

Repeat K = 100N times

Optimal configuration has E = 25

Starting configuration has E = 55
Energy of the configurations as the algorithm runs
Start bottom right, end top left.
Another Example

$N = 13$ nodes

Repeat $K = 100N$ times
"Belongs to \( S^* \)" just means an optimal solution

So...if you run an infinite number of iterations you will eventually find the optimal solution. Note...the limit as \( T \) goes to 0 does NOT mean \( T = 0 \).

Simulated annealing is a useful algorithm that is actually used in real life, unlike most things Luis talks about.
Simulated Annealing

\[ X \leftarrow \text{Initial configuration} \]
\[ T \leftarrow \text{Initial high temperature} \]

Iterate:

1. Do \( K \) times:
   \[ E \leftarrow \text{Eval}(X) \]
   \[ X' \leftarrow \text{one configuration randomly selected in Neighbors}(X) \]
   \[ E' \leftarrow \text{Eval}(X') \]
   If \( E' \geq E \)
      \[ X \leftarrow X'; E \leftarrow E'; \]
   Else accept the move with prob \( p = e^{-\frac{(E - E')}{T}} \):
      \[ X \leftarrow X'; E \leftarrow E'; \]
2. \( T \leftarrow \alpha T \)

Many parameters need to be tweaked!!
Genetic/Evolutionary Algorithms
Genetic Algorithms

Configurations = Individuals in a population
Eval = measure of fitness
Least fit individuals DIE without reproducing
Most fit individuals reproduce more often
Each generation should be better than the past!
Genes are contiguous groups of 1’s and 0’s

| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

**GA: Implementation**

- Configurations represented by strings:
  \[ \chi = [1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1] \]

- Analogy:
  - The string is the chromosome representing the individual
  - String made up of genes
  - Configuration of genes are passed on to offsprings
  - Configurations of genes that contribute to high fitness tend to survive in the population

- Start with a random population of \( P \) configurations and apply two operations
  - *Reproduction*: Choose 2 “parents” and produce 2 “offsprings”
  - *Mutation*: Choose a random entry in one (randomly selected) configuration and change it
Genetic Algorithms: Reproduction

Parents:

1 0 0 1 1 0 0 1
1 0 1 1 0 0 0 1
Genetic Algorithms: Reproduction

Parents:

1 0 0 1 1 0 0 1
1 0 1 1 0 0 0 1

Select random crossover point:

1 0 0 1 1 0 0 1
1 0 1 1 0 0 0 1
Genetic Algorithms: Reproduction

Parents:

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1
\end{array}
\quad
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0
\end{array}
\]

Select random crossover point:

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0
\end{array}
\quad
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1
\end{array}
\]

Offsprings:

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1
\end{array}
\quad
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0
\end{array}
\]

An offspring receives part of the genes from each of the parents.
**Genetic Algorithms: Mutation**

- Implements random deviations from inherited traits
- Corresponds loosely to “random walk”: Introduce random moves to avoid local extrema
Basic GA Outline

- Create initial population $X = \{X_1, \ldots, X_p\}$
- Iterate:
  1. Select $K$ random pairs of parents $(X, X')$
  2. For each pair of parents $(X, X')$:
     1.1 Generate offsprings $(Y_1, Y_2)$ using crossover operation
     1.2 For each offspring $Y_i$:
         Replace randomly selected element of the population by $Y_i$
         With probability $\mu$:
         Apply a random mutation to $Y_i$
- Return the best individual in the population
Basic GA Outline

- Create initial population \(X = \{X_p\}\)
- Iterate:
  1. Select \(K\) random pairs of parents \((X, X')\)
  2. For each pair of parents \((X, X')\):
     1.1 Generate offspring \(Y_i\)
     Variation: Generate only one offspring
     - For operation on each offspring \(Y_i\)
     - Replace randomly selected individuals in the population by \(Y_i\)
     - With probability \(\mu\):
       - Apply a random mutation to \(Y_i\)
   - Possible strategy:
     - Select the best \(rP\) individuals \((r < 1)\) for reproduction and discard the rest
     - Implements selection of the fittest

- Return the best individual in the population
Genetic Algorithms: Selection

- Discard the least-fit individuals through threshold on Eval or fixed percentage of population
- Select best-fit (larger Eval) parents in priority
- Example: Random selection of individual based on the probability distribution

\[
Pr(\text{individual } X \text{ selected}) = \frac{Eval(X)}{\sum_{Y \in \text{population}} Eval(Y)}
\]

- Corresponds loosely to the greedy part of hill-climbing (we try to move uphill)
Basic GA Outline

• Create initial population $X = \{X_1, \ldots, X_p\}$

• Iterate:
  1. Select $K$ random parents
  2. For each pair of parents $(X, X')$:
     1.1 Generate offsprings $(Y_1, Y_2)$ using crossover operation
  3. Offspring $Y_i$:
     - Randomly selected element of population by $Y_i$
     - With probability $\mu$:
       Apply a random mutation to $Y_i$

• Return the best individual in the population
How to encode configurations as strings of 1’s and 0’s?
You want things that should stick together to be next to each other in the string.
$N = 13$

$P = 100$ elements in population

$\mu = 4\%$ mutation rate

$r = 50\%$ reproduction rate
Best (lowest cost) element in population

Initial population

Best K elements in population candidate for reproduction
Another TSP Example

Converges and remains stable after generation 23

0.4% difference:
GA = 11.801
SA = 11.751

But: Number of operations (number of cost evaluations) much smaller (approx. 2500)
Population at generation 40
People were extremely excited 20 years ago...not so much now...GA don't work very well. But they are cool.

Encoding a problem for a GA is very difficult to do well.

Here's a quote from Russell + Norvig:
"[It] is not clear whether the appeal of genetic algorithms arises from their performance or from their aesthetically pleasing origins in the theory of evolution."
GA Discussion

- Why does it work at all?
- Limited theoretical results (informally!):
  - Suppose that there exists a partial assignment of genes \( s \) such that:
    \[
    \text{Average of } \text{Eval}(X) \geq \text{Average of } \text{Eval}(Y) \\
    \text{if } X \text{ contains } s \\
    \text{and } Y \in \text{Population}
    \]
  - Then the number of individuals containing \( s \) will increase in the next generation
- Key consequence: The design of the representation (the chromosomes) is critical to the performance the GA. It is probably more important than the choice of parameters of selection strategy, etc.
Summary

- Hill Climbing
- Stochastic Search
- Simulated Annealing
- Genetic Algorithms

- Class of algorithms applicable to many practical problems
- Not useful if more direct search methods can be used
- The algorithms are general black-boxes. What makes them work is the correct engineering of the problem representation
  - State representation
  - Neighborhoods
  - Evaluation function
  - Additional knowledge and heuristics
(Some) References

- Russell & Norvig, Chap. 4
- Numerical Recipes (http://www.nr.com/).