Local Search

CS 486/686: Introduction to Artificial Intelligence
Overview

• Uninformed Search
  - Very general: assumes no knowledge about the problem
  - BFS, DFS, IDS

• Informed Search
  - Heuristics
  - A* search and variations

• Search and Optimization
  - What are the problem features?
  - Iterative improvement: hill climbing, simulated annealing
  - Genetic algorithms
Introduction

• Both uninformed and informed search systematically explore the search space
  - Keep 1 or more paths in memory
  - Solution is a path to the goal

• For many problems, the path is unimportant
Examples

AV ~B V C
~A V C V D
B V D V ~E
~C V ~D V ~E
...

Agents = dispatch centers
Informal Characterization

• Combinatorial structure being optimized
• Constraints have to be satisfied
• There is a cost function
  - We want to find a good solution
• Search all possible states is infeasible
  - Often easy to find some solution to the problem
  - Often provably hard (NP-complete) to find the best solution
Typical Example: TSP

- Goal is to minimize the length of the route
- **Constructive method**: Start from scratch and build up a solution
- **Iterative improvement method**: Start with solution (may be suboptimal or broken) and improve it
Constructive Methods

• For the optimal solution we can use A*
• But...
• We do not need to know how we got the solution
  - We just want the solution
Iterative Improvement Methods

- Idea: Imagine all possible solutions laid out on a landscape
  - Goal: find the highest (or lowest) point
Iterative Improvement Methods

- Start at some random point
- Generate all possible points to move to
- If the set is not empty, choose a point and move to it
- If you are stuck (set is empty), then restart
Iterative Improvement Methods

• What does it mean to “generate points to move to”
  - Generating the moveset

• Depends on the application

TSP

2-swap
Hill Climbing (Gradient Descent)

- Main idea
  - Always take a step in the direction that improves the current solution value the most
- Variation of best-first search
- Very popular for learning algorithms

“...like trying to find the top of Mt Everest in a thick fog while suffering from amnesia”, Russell and Norvig
Hill Climbing

1. Start with some initial configuration $S$
2. Let $V$ be the value of $S$
3. Let $S_i$, $i=1,...,n$ be neighbouring configs, $V_i$ are corresponding values
4. Let $V_{\text{max}}=\max_i V_i$ be value of best config and $S_{\text{max}}$ is the corresponding config
   - If $V_{\text{max}}<V$ return $S$ (local optimum)
   - Let $S \leftarrow S_{\text{max}}$ and $V \leftarrow V_{\text{max}}$. Go to 3
Judging Hill Climbing

• Good news
  - Easy to program
  - Requires no memory of where we have been
  - Important to have a “good” set of moves
    - Not too many, not too few
Judging Hill Climbing

- Bad news
  - It can get stuck
  - Local maxima/minima
  - Plateaus
Improving Hill Climbing

- Plateaus
  - Allow for sideways moves
    - But be careful since might move sideways forever

- Local Maxima
  - Random restarts: *If at first you do not succeed, try, try again!*
Randomized Hill Climbing

- Like hill climbing except
  - You choose a random state from the move set
  - Move to it if it is better than current state
  - Continue until you are bored
More Randomization

• Hill climbing is incomplete
  - can get stuck at local optima

• A random walk is complete
  - but very inefficient

• New Idea:
  - Allow the algorithm to make some “bad” moves in order to escape local optima
**Example: GSAT**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AV~BVC</td>
<td>1</td>
<td>~AVCVD</td>
<td>1</td>
<td>BVDV~E</td>
</tr>
<tr>
<td><del>CV</del>DV~E</td>
<td>1</td>
<td><del>AV</del>CVE</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Configuration $A=1$, $B=0$, $C=1$, $D=0$, $E=1$

Goal is to maximize the number of satisfied clauses: $\text{Eval(config)} = \# \text{ satisfied clauses}$

**GSAT Move Set**: Flip any 1 variable

---

**WALKSAT (Randomized GSAT)**

Pick a random unsatisfied clause;

Consider flipping each variable in the clause

- If any improve $\text{Eval}$, then accept the best
- If none improve $\text{Eval}$, then with prob $p$ pick the move that is least bad; prob $(1-p)$ pick a random one
Simulated Annealing

1. S is initial config and V = Eval(S)
2. Let i be a random move from the moveset and let $S_i$ be the next config, $V_i = \text{Eval}(S_i)$
3. If $V < V_i$, then $S = S_i$ and $V = V_i$
4. Else with probability $p$, $S = S_i$ and $V = V_i$
5. Go to 2 until you are bored
What About p?

• How should we choose the probability of making a “bad” move?
  - $p=0.1$ (or some fixed value)?
  - Decrease $p$ with time?
  - Decrease $p$ with time and as $V-V_i$ increases?
  - ...
Selecting Moves in Simulated Annealing

- If new value $V_i$ is better than old value $V$ then definitely move to new solution
- If new value $V_i$ is worse than old value $V$ then move to new solution with probability

$$e^{- (V - V_i)/T}$$

**Boltzmann Distribution**: $T > 0$ is a parameter called temperature. It starts high and decreases over time towards 0. If $T$ is close to 0 then the prob. of making a bad move is almost 0.
Properties to Simulated Annealing

- When $T$ is high:
  - **Exploratory phase**: even bad moves have a chance of being picked (random walk)

- When $T$ is low:
  - **Exploitation phase**: “bad” moves have low probability of being chosen (randomized hill climbing)

- If $T$ is decreased slowly enough then simulated annealing is guaranteed to reach optimal solution
Genetic Algorithms

- Populations are encoded into a representation which allows certain operations to occur
  - Usually a bitstring
  - Representation is key - needs to be thought out carefully
- An encoded candidate solution is an individual
- Each individual has a fitness
  - Numerical value associated with its quality of solution
- A population is a set of individuals
- Populations change over generations by applying operators to them
  - Operations: selection, mutation, crossover
Typical Genetic Algorithm

- Initialize: Population $P \leftarrow N$ random individuals
- Evaluate: For each $x$ in $P$, compute $\text{fitness}(x)$
- Loop
  - For $i=1$ to $N$
    - Select 2 parents each with probability proportional to fitness scores
    - Crossover the 2 parents to produce a new bitstring (child)
    - With some small probability mutate child
    - Add child to population
  - Until some child is fit enough or you get bored
- Return best child in the population according to fitness function
Selection

- Fitness proportionate selection: $P(i) = \frac{\text{fitness}(i)}{\sum_j \text{fitness}(j)}$
  - Can lead to overcrowding

- Tournament selection
  - Pick i, j at random with uniform probability
  - With probability p select fitter one

- Rank selection
  - Sort all by fitness
  - Probability of selection is proportional to rank

- Softmax (Boltzmann) selection: $P(i) = \frac{e^{\text{fitness}(i)/T}}{\sum_j e^{\text{fitness}(j)/T}}$
Crossover

• Combine parts of individuals to create new ones
• For each pair, choose a random crossover point
  – Cut the individuals there and swap the pieces

\[
\begin{array}{c|c}
101 & 0101 \\
\hline
011 & 1110 \\
\end{array}
\]

Cross over

\[
\begin{array}{c|c}
011 & 0101 \\
\hline
101 & 1110 \\
\end{array}
\]

Implementation: use a crossover mask m

Given two parents a and b the offspring are

\((a \wedge m) \vee (b \wedge \neg m)\) and \((a \wedge \neg m) \vee (b \wedge m)\)
Mutation

• Mutation generates new features that are not present in original population.

• Typically means flipping a bit in the string. 

  100111 mutates to 100101

• Can allow mutation in all individuals or just in new offspring.
Example
Summary

• Useful for optimization problems
• Often the second-best way to solve a problem
  - If you can, use A* or linear programming or ...
• Need to think about how to escape from local optima
  - Random restarts
  - Allowing for bad moves
  - ...