Local Search

CPSC 322 – CSPs 5

Textbook §4.8

Lecture Overview

1. Recap
2. Randomized Algorithms
3. Comparing SLS Algorithms
4. SLS Variants

Local Search

Definition

The problem of solving a CSP phrased as local search problem is given by:

- CSP. In other words, a set of variables, domains for these variables, and constraints on their joint values. A node in the search space will be a complete assignment to *all* of the variables.
- Neighbour relation. assignments that differ in the value assigned to one variable, or in the value assigned to the variable that participates in the largest number of conflicts
- Scoring function. Number of unsatisfied constraints.

Hill Climbing

Hill climbing means selecting the neighbour which best improves the scoring function.

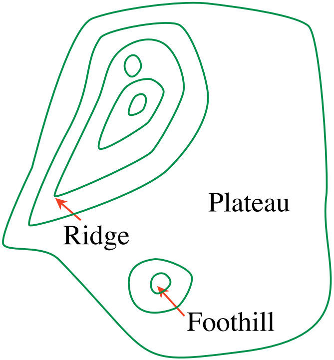
- For example, if the goal is to find the highest point on a surface, the scoring function might be the height at the current point.

Problems with Hill Climbing

Foothills local maxima that are not global maxima

Plateaus heuristic values are uninformative

Ridge foothill where a larger neighbour relation would help



Lecture Overview

1. Recap
2. Randomized Algorithms
3. Comparing SLS Algorithms
4. SLS Variants

Randomized Algorithms

- Consider two methods to find a maximum value:
  - Hill climbing, starting from some position, keep moving uphill & report maximum value found
  - Pick values at random & report maximum value found
- Which do you expect to work better to find a maximum?

Randomized Algorithms

- Consider two methods to find a maximum value:
  - Hill climbing, starting from some position, keep moving uphill & report maximum value found
  - Pick values at random & report maximum value found
- Which do you expect to work better to find a maximum?
  - hill climbing is good for finding local maxima
  - selecting random nodes is good for finding new parts of the search space
- A mix of the two techniques can work even better

Stochastic Local Search

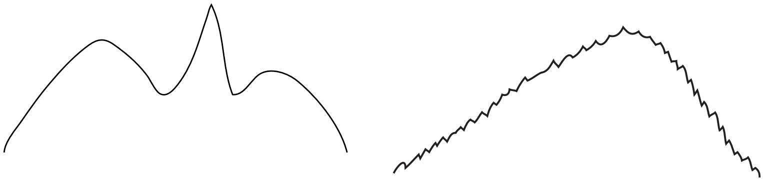
- We can bring these two ideas together to make a randomized version of hill climbing.
- As well as uphill steps we can allow for:
  - Random steps: move to a random neighbor.
  - Random restart: reassign random values to all variables.
- Which is more expensive computationally?

Stochastic Local Search

- We can bring these two ideas together to make a randomized version of hill climbing.
- As well as uphill steps we can allow for:
  - Random steps: move to a random neighbor.
  - Random restart: reassign random values to all variables.
- Which is more expensive computationally?
  - usually, random restart (consider that there could be an extremely large number of neighbors)
  - however, if the neighbour relation is computationally expensive, random restart could be cheaper

1-Dimensional Ordered Examples

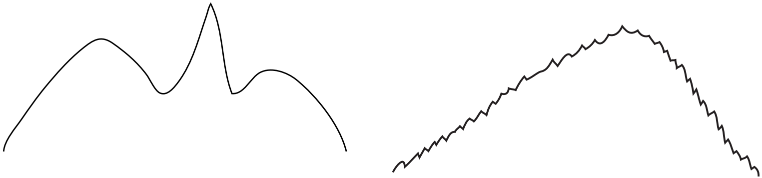
Two 1-dimensional search spaces; step right or left:



- Which of hill climbing with random walk and hill climbing with random restart would most easily find the maximum?

1-Dimensional Ordered Examples

Two 1-dimensional search spaces; step right or left:



- Which of hill climbing with random walk and hill climbing with random restart would most easily find the maximum?
  - left: random restart; right: random walk
- As indicated before, stochastic local search often involves both kinds of randomization

Random Walk

Some examples of ways to add randomness to local search for a CSP:

- When choosing the best variable-value pair, randomly sometimes choose a random variable-value pair.
- When selecting a variable followed by a value:
  - Sometimes choose the variable which participates in the largest number of conflicts.
  - Sometimes choose, at random, any variable that participates in some conflict.
  - Sometimes choose a random variable.
  - Sometimes choose the best value for the chosen variable.
  - Sometimes choose a random value for the chosen variable.

Lecture Overview

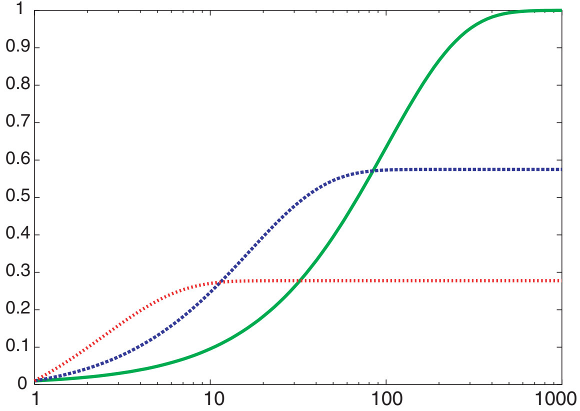
1. Recap
2. Randomized Algorithms
3. Comparing SLS Algorithms
4. SLS Variants

Comparing Stochastic Algorithms

- How can you compare three algorithms when (e.g.,)
  - one solves the problem 30% of the time very quickly but doesn't halt for the other 70% of the cases
  - one solves 60% of the cases reasonably quickly but doesn't solve the rest
  - one solves the problem in 100% of the cases, but slowly?
- Summary statistics, such as mean run time, median run time, and mode run time don't tell the whole story
  - mean: what should you do if an algorithm *never* finished on some runs (infinite? stopping time?)
  - median: an algorithm that finishes 51% of the time is preferred to one that finishes 49% of the time, regardless of how fast it is

Runtime Distribution

- A RTD plots runtime (or number of steps) and the proportion (or number) of the runs that are solved within that runtime.
  - note the use of a log scale on the x axis



Lecture Overview

Greedy Descent with Min-Conflict Heuristic

This is one of the best techniques for solving CSP problems:

- At random, select one of the variables v that participates in a violated constraint
- Set v to one of the values that minimizes the number of unsatisfied constraints
- This can be implemented efficiently:
  - Data structure 1 stores currently violated constraints
  - Data structure 2 stores variables that are involved in violated constraints
  - Selecting the variable to change is a random draw from data structure 2
  - For each of v's values i, count the number of constraints that would be violated if v took the value i
  - When the new value is set:
    - add all variables that participate in newly-violated constraints
    - check all variables that participate in newly-satisfied constraints to see if they participate in any other violated constraints

Simulated Annealing

- Annealing: a metallurgical process where metals are hardened by being slowly cooled.
- Analogy: start with a high "temperature": a high tendency to take random steps
- Over time, cool down: more likely to follow the gradient
- Here's how it works:
  - Pick a variable at random and a new value at random.
  - If it is an improvement, adopt it.
  - If it isn't an improvement, adopt it probabilistically depending on a temperature parameter, T.
    - With current node n and proposed node n' we move to n' with probability $e^{(h(n')-h(n))/T}$
  - Temperature reduces over time, according to an annealing schedule

Tabu lists

- SLS algorithms can get stuck in plateaus (why?)

Tabu lists

- SLS algorithms can get stuck in plateaus (why?)
- To prevent cycling we can maintain a tabu list of the k last nodes visited.
- Don't visit a node that is already on the tabu list.
- If $k = 1$, we don't allow the search to visit the same assignment twice in a row.
- This method can be expensive if k is large.

Parallel Search

- Idea: maintain k nodes instead of one.
- At every stage, update each node.
- Whenever one node is a solution, report it.
- Like k restarts, but uses k times the minimum number of steps.
- There's not really any reason to use this method (why not?), but it provides a framework for talking about what follows...

Beam Search

- Like parallel search, with k nodes, but you choose the k best out of all of the neighbors.
- When $k = 1$, it is hill climbing.
- When $k = \infty$, it is breadth-first search.
- The value of k lets us limit space and parallelism.

Stochastic Beam Search

- Like beam search, but you probabilistically choose the k nodes at the next generation.
- The probability that a neighbor is chosen is proportional to the value of the scoring function.
    - This maintains diversity amongst the nodes.
    - The scoring function value reflects the fitness of the node.
    - Biological metaphor: like asexual reproduction, as each node gives its mutations and the fittest ones survive.

Genetic Algorithms

- Like stochastic beam search, but pairs of nodes are combined to create the offspring:
- For each generation:
  - Randomly choose pairs of nodes, with the best-scoring nodes being more likely to be chosen.
  - For each pair, perform a cross-over: form two offspring each taking different parts of their parents
  - Mutate some values
- Report best node found.

Crossover

- Given two nodes:

$$X_1 = a_1, X_2 = a_2, \ldots, X_m = a_m$$

$$X_1 = b_1, X_2 = b_2, \ldots, X_m = b_m$$

- Select i at random.
- Form two offspring:

$$X_1 = a_1, \ldots, X_i = a_i, X_{i+1} = b_{i+1}, \ldots, X_m = b_m$$

$$X_1 = b_1, \ldots, X_i = b_i, X_{i+1} = a_{i+1}, \ldots, X_m = a_m$$

- Note that this depends on an ordering of the variables.
- Many variations are possible.