# Local Search

CPSC 322 – CSPs 5

Textbook §4.8

# Lecture Overview

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

# Stochastic Local Search for CSPs

- A 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
- Goal is to find an assignment with all constraints satisfied.
  - Scoring function: the number of unsatisfied constraints.
  - We want an assignment with minimum score.

# 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.

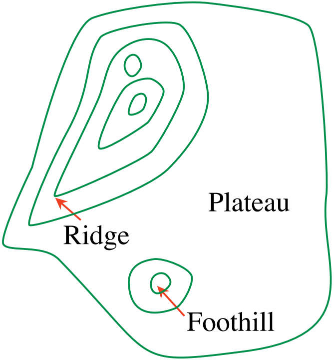
# Problems with Hill Climbing

**Foothills** local maxima that are not global maxima

**Plateaus** heuristic values are uninformative

**Ridge** foothill where a larger neighbour relation would help

**Ignorance of the peak** no way of detecting a global 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 will 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

# Lecture Overview

# Stochastic Local Search

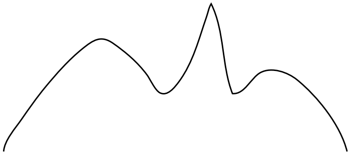
- We can bring these two ideas together to make a randomized version of hill climbing.
- As well as uphill steps we can allow for:
  - Random steps: move to a random neighbor.
  - Random restart: reassign random values to all variables.
- Which is more expensive computationally?

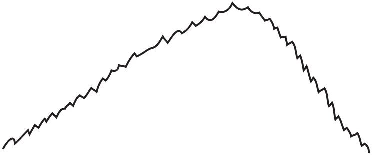
# Stochastic Local Search

- 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:

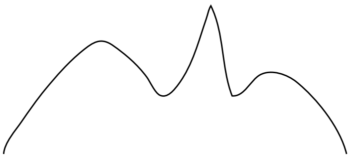


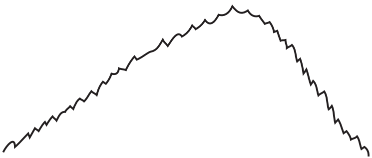


- Which of hill climbing with random walk and hill climbing with random restart would most easily find the maximum?

# 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

# 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

- 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