Stochastic Local Search Variants; Planning Intro

CPSC 322 – CSPs 6

Textbook §4.8; §11.1

Lecture Overview

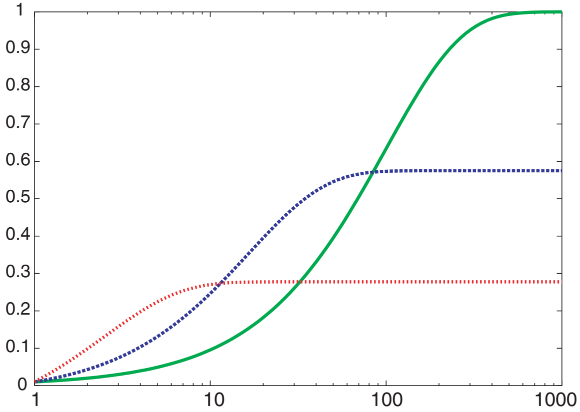
1. Recap
2. SLS Variants

Stochastic Local Search

- Idea: combine hill climbing (advantage: finds local maximum) with randomization (advantage: doesn't get stuck).
- As well as uphill steps we can allow a small probability of:
  - Random steps: move to a random neighbor.
  - Random restart: reassign random values to all variables.

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



Lecture Overview

1. Recap
2. SLS Variants

Variant: 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

Variant: 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 heuristic 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.