

Local Search:

- Maintain a complete assignment of a value to each variable.
- Start with random assignment or a best guess.
- Repeat:
 - ▶ Select a variable to change
 - ▶ Select a new value for that variable
- Until a satisfying assignment is found

Local Search for CSPs

- Aim: find an assignment with zero unsatisfied constraints.
- Given an assignment of a value to each variable, a **conflict** is an unsatisfied constraint.
- The goal is an assignment with zero conflicts.
- Function to be minimized: the number of conflicts.

Iterative Best Improvement (2 stage) “greedy descent”

- Start with random assignment (for each variable, select a value for that variable at random)
- Repeat:
 - ▶ Select a variable that participates in the most conflicts
 - ▶ Select a different value for that variable
- Until a satisfying assignment is found

All selections are random and uniform.

- Start with random assignment (for each variable, select a value for that variable at random)
- Repeat:
 - ▶ Select a variable at random that participates in any conflict
 - ▶ Select a different value for that variable
- Until a satisfying assignment is found

All selections are random and uniform.

Which of the preceding algorithms work better?

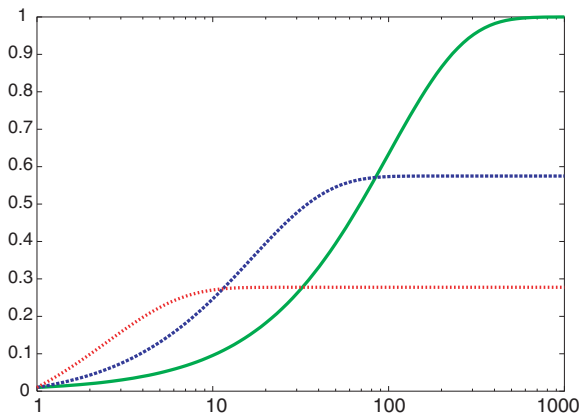
How would we tell if one is better than the other?

- How can you compare three algorithms when
 - ▶ 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 make much sense.

Runtime Distribution

x-axis runtime (or number of steps)

y-axis the proportion (or number) of runs that are solved within that runtime



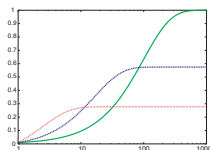
Runtime Distribution

- Run the same algorithm on the same instance for a number of trials (e.g., 100 or 1000)
- Sort the trials according to the run time.
- Plot:

x-axis run time of the trial
y-axis index of the trial

This produces a cumulative distribution

- Do this a few times to gauge the variability (take a statistics course!)
- Sometimes use number of steps instead of run time (because computers measure small run times inaccurately) . . . not good measure to compare algorithms if steps take different times



- A probabilistic mix of *greedy* and *any-conflict* — e.g., 70% of time pick best variable, otherwise pick any variable in a conflict – works better than either alone.

Stochastic local search is a mix of:

- **Greedy descent:** pick the best variable and/or value
- **Random walk:** picking variables and values at random
- **Random restart:** reassigning values to all variables

Some of these might be more complex than the others.
A probabilistic mix might work better.

Greedy Descent Variants

To select a variable to change and a new value for it:

- **Most Improving Step:** Find a variable-value pair that minimizes the number of conflicts.
What data structures are required?
- **Two Stage Choice:** Select a variable that participates in the most conflicts.
Select a value that minimizes the number of conflicts.
What data structures are required?
- **Any Conflict:** Select a variable that appears in any conflict.
Select a value at random.
What data structures are required?
- Select a variable at random.
Select a value that minimizes the number of conflicts.
What data structures are required?
- Select a variable and value at random; accept this change if it doesn't increase the number of conflicts.
What data structures are required?

- One measure of an assignment is number of conflicts
- It is possible to weight some conflicts higher than others.
- Why would we?
Because some are easier to solve than other. E.g., in scheduling exams....
- If A is a total assignment, define $h(A)$ to be a measure of the difficulty of solving problem from A .
- $h(A) = 0$ then A a solution; lower h is better

Variant: Simulated Annealing

- Pick a variable at random and a new value at random.
- If it isn't worse, accept it.
- If it is worse, accept it probabilistically depending on a temperature parameter, T :
 - ▶ With current assignment A and proposed assignment A' accept A' with probability $e^{(h(A)-h(A'))/T}$

Note: $h(A) - h(A')$ is negative if A' is worse.

- Probability of accepting a change:

Temperature	1-worse	2-worse	3-worse
10	0.91	0.81	0.74
1	0.37	0.14	0.05
0.25	0.02	0.0003	0.000006
0.1	0.00005	2×10^{-9}	9×10^{-14}

- Temperature can be reduced.

Random Restart

- A random restart involves reassigning all variables to values at random.
- allows for exploration of a different part of the search space.
- Each run is independent of the others, so probabilities can be derived analytically.

Suppose each run has a probability of p of finding a solution.
We do n runs or until a solution is found.

The probability of n runs failing to find a solution is $(1 - p)^n$

The probability of finding a solution in n -runs is $1 - (1 - p)^n$

n	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.8$
5	0.410	0.832	0.969	0.9997
10	0.65	0.971	0.9990	0.9999998
20	0.878	0.9992	0.9999991	0.99999999999
50	0.995	0.99999998	0.9999999999999991	1.0

- To prevent cycling we can maintain a **tabu list** of the k last assignments.
- Don't allow an assignment that is already on the tabu list.
- If $k = 1$, we don't allow an assignment of to the same value to the variable chosen.
- We can implement it more efficiently than as a list of complete assignments.
- It can be expensive if k is large.

Ordered and Continuous Domains

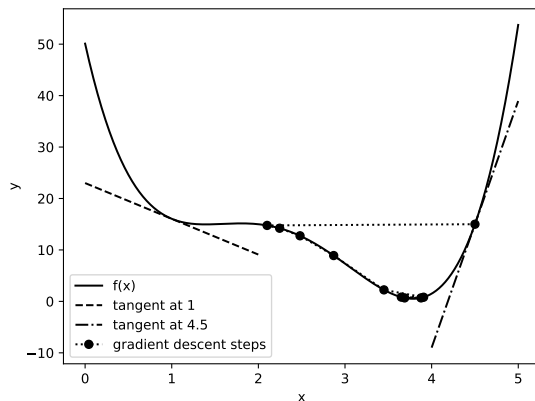
- When the domains are small or unordered, the neighbors of an assignment can correspond to choosing another value for one of the variables.
- When the domains are large and ordered, the neighbors of an assignment are the adjacent values for one of the variables.
- If the domains are continuous, **Gradient descent** moves each each variable downhill; proportional to the gradient of the heuristic function in that direction.

The value of variable X_i goes from v_i to $v_i - \eta \frac{\partial h}{\partial X_i}$.

η is the step size.

- Neural networks do gradient descent with many parameters (variables) to minimize an error on a dataset. Some large language models have over 10^{12} parameters.

Gradient Descent



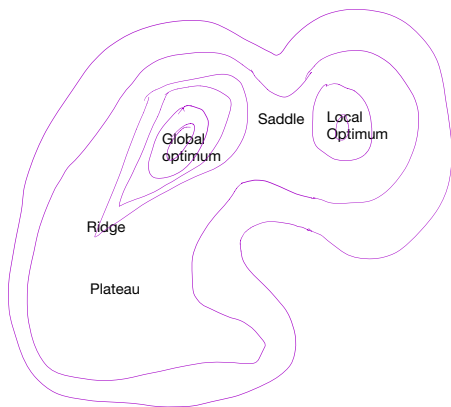
$$y = 2 * (x - 1.3) * (x - 1.5) * (x - 2) * (x - 4.5) + 15$$

Step size is 0.05 and gradient descent starts at $x = 4.5$.

What if it starts at $x = 5.5$? (Hint: the derivative is more than 4 times larger). What if it starts at $x = 1.5$?

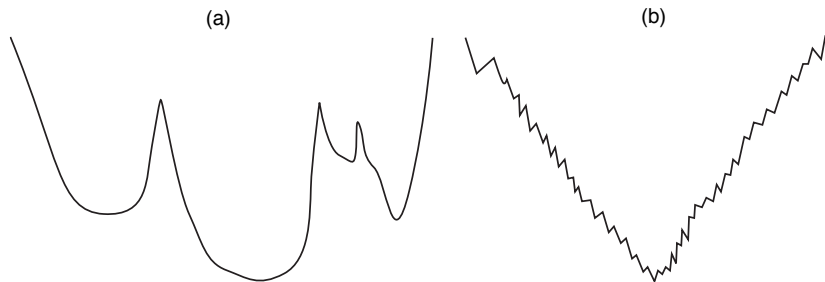
Problems with Greedy Descent

- a local optimum that is not a global optimum
- a plateau where the heuristic values are uninformative
- a ridge is a local minimum where n -step look-ahead might help
- a saddle is a flat area where steps need to change direction



1-Dimensional Ordered Examples

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



- Which method would most easily find the global minimum?
- What happens in hundreds or thousands of dimensions?
- What if different parts of the search space have different structure?

A total assignment is called an **individual**.

- **Idea:** maintain a population of k individuals instead of one.
- At every stage, update each individual in the population.
- Whenever an individual is a solution, it can be reported.
- Like k restarts, but uses k times the *minimum* number of steps.

- Like parallel search, with k individuals, but choose the k best out of all of the neighbors.
- When $k = 1$, it is greedy descent.
- The value of k lets us limit space and parallelism.
- Problem: lack of diversity of individuals.

Stochastic Beam Search

- Like beam search, but it probabilistically chooses the k individuals at the next generation.
- The probability that a neighbor is chosen is proportional to its heuristic value.
- This maintains diversity amongst the individuals.
- The heuristic value reflects the fitness of the individual.
- Like asexual reproduction: each individual mutates and the fittest ones survive.

- Like stochastic beam search, but pairs of individuals are combined to create the offspring.
- For each generation:
 - ▶ Randomly choose pairs of individuals where the fittest individuals are more likely to be chosen.
 - ▶ For each pair, perform a crossover: form two offspring each taking different parts of their parents.
 - ▶ Mutate some values.
- Stop when a solution is found.

- Given two individuals:

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

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

- Select i at random.
- Form two offspring:

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

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

- The effectiveness depends on the ordering of the variables.
- Many variations are possible.

An **optimization problem** is given

- a set of variables, each with an associated domain
- an **objective function** that maps total assignments to real numbers, and
- an **optimality criterion**, which is typically to find a total assignment that minimizes (or maximizes) the objective function.

Constraint optimization problem

- In a constraint optimization problem the objective function is factored into a sum of soft constraints
- A **soft constraint** is a function from scope of constraint into non-negative reals (the cost)
- The aim is to find a total assignment that minimizes the sum of the values of the soft constraints.
- Can use systematic search (e.g., A^* or branch-and-bound search)
- Arc consistency can be used to prune dominated values
- Can use local search
- Problem: we can't tell if a value is a global minimum unless we do systematic search