# Local Search

- We have considered algorithms that systematically search the space.
- If the space is finite, they will either find a solution or report that no solution exists.
- Unfortunately, many search spaces are too big for systematic search and are possibly even infinite.
- *Local search* methods do not systematically search the whole search space but they are designed to find solutions quickly on average.
- They do not guarantee that a solution will be found even if one exists, and so they are not able to prove that no solution exists.

# Local Search

Local Search:

- Maintain an assignment of a value to each variable.
- At each step, select a "neighbor" of the current assignment (e.g., one that improves some heuristic value).
- Stop when a satisfying assignment is found, or return the best assignment found.

Requires:

- What is a neighbor?
- Which neighbor should be selected?

(Some methods maintain multiple assignments.)

# **Procedure** Local-Search($V$, $dom$, $C$)

**Inputs:** $V$: a set of variables,
$dom$: a function such that $dom(X)$ is the domain of variable $X$,
$C$: set of constraints to be satisfied.
**Local:** $A[V]$ an array of values indexed by $V$
**repeat**
(a try)
**for each** variable $X$ **do**
(random initialisation)
$A[X] ::=$ random value in $dom(X)$;
**while** (stop criterion not met & $A$ is not a satisfying assignment)
(local search or walk)
Select a variable $Y$ and a value $V \in dom(Y)$
$A[Y] ::= V$
**if** ($A$ is a satisfying assignment) **then**
**return** $A$
**until** termination

# Random Sampling and Random Walk

Random Sampling:

- The stop criterion is always true: the while loop is never executed.
- It keeps picking random assignments until it finds one that satisfies the constraints, and otherwise it does not halt.
- Random sampling is complete in the sense that, given enough time, it guarantees that a solution will be found if one exists, but there is no upper bound on the time it may take.

Random Walk:

- The while loop is only exited when it has found a satisfying assignment (i.e., the stopping criterion is always false and there are no random restarts).
- In the while loop it selects a variable and a value at random.
- Random walk is also complete in the same sense as random sampling.
- A random walk algorithm can either select a variable at random and then a value at random, or select a variable-value pair at random. The latter is more likely to select a variable when it has a larger domain.

# Selecting Neighbors in Local Search

- In *iterative best improvement*, the neighbor of the current selected node is one that optimizes some evaluation function.
- In *greedy descent*, a neighbor is chosen to minimize an evaluation function. This is also called *hill climbing* or greedy ascent when the aim is to maximize.
- If the domains are continuous, Gradient descent changes each variable proportional to the gradient of the heuristic function in that direction. The value of variable $X_i$ goes from $v_i$ to $v_i - \eta \frac{\partial h}{\partial X_i}$.
  Gradient ascent: go uphill; $v_i$ becomes $v_i + \eta \frac{\partial h}{\partial X_i}$.

# Local Search for CSPs

- Aim is to 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.
- Heuristic function to be minimized: the number of conflicts.

# Best improvement step

- Algorithms can differ in how much work they require to guarantee the best improvement step.
- At one extreme, an algorithm can guarantee to select a new assignment that gives the best improvement out of all of the neighbors.
- At the other extreme, an algorithm can select a new assignment at random and reject the assignment if it makes the situation worse.
- We will see typical algorithms that differ in how much computational effort they put in to ensure that they make the best improvement.
- Which of these methods works best is, typically, an empirical question.

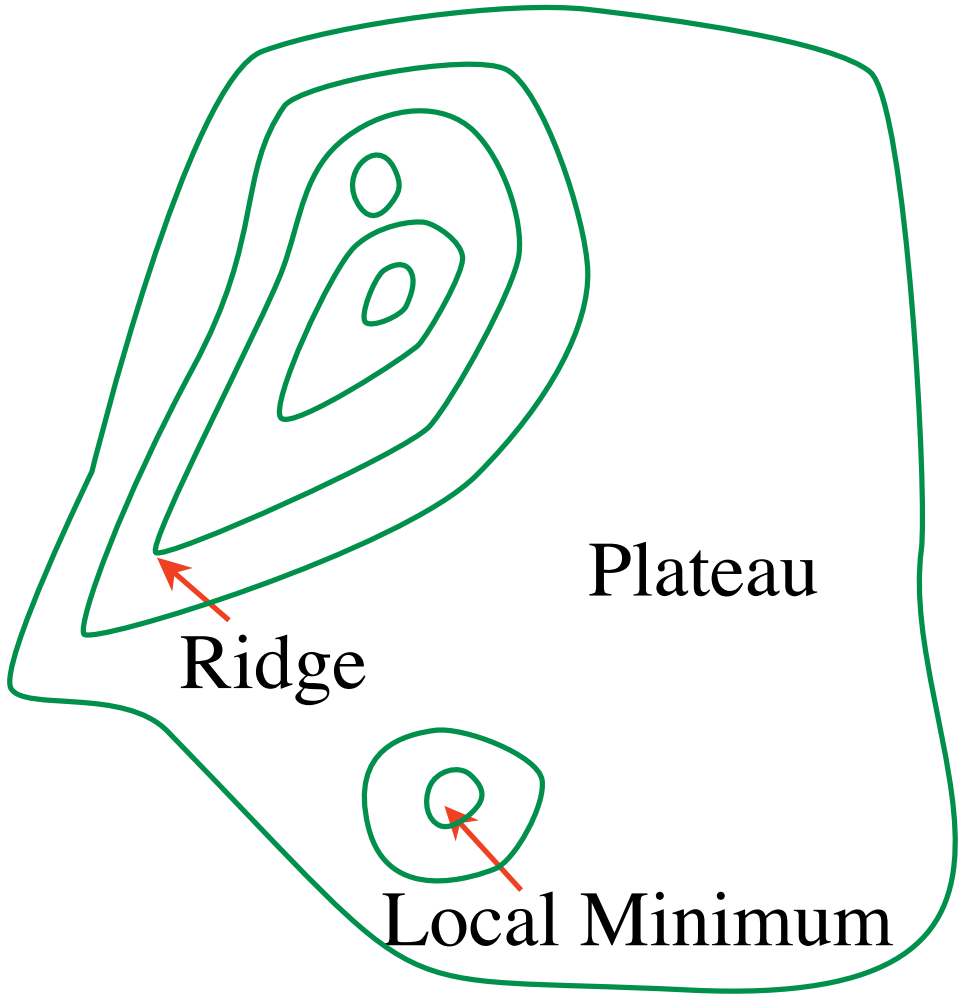
# Greedy Descent Variants

- Find the variable-value pair that minimizes the number of conflicts at every step ("1-stage choice" or "Most Improving Step").
- Select a variable that participates in the most number of conflicts. Select a value that minimizes the number of conflicts ("2-Stage choice")
- Select a variable that appears in any conflict. Select a value that minimizes the number of conflicts ("Any Conflict choice").
- Select a variable at random. Select a value that minimizes the number of conflicts.
- Select a variable and value at random; accept this change if it doesn't increase the number of conflicts.

Although some theoretical results exist, deciding which method works better in practice is an empirical question.

# Problems with Greedy Descent

- a local minimum that is not a global minimum
- a plateau where the heuristic values are uninformative
- a ridge is a local minimum where *n*-step look-ahead might help



# Randomized Algorithms

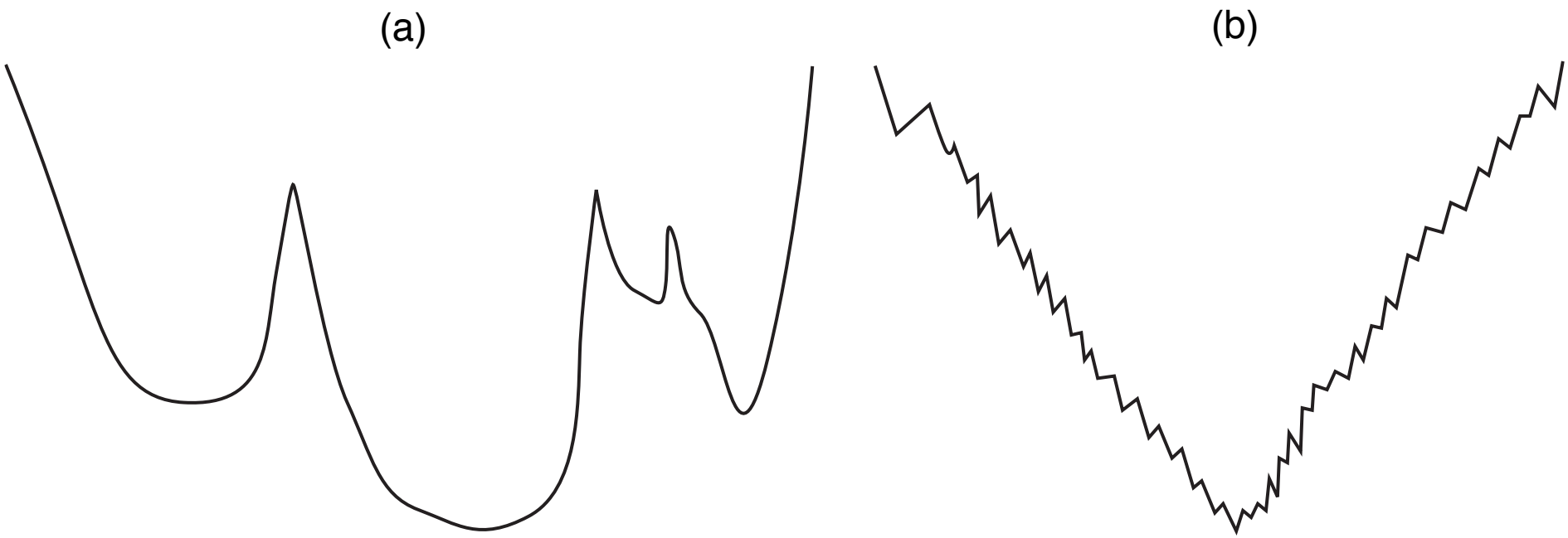
- Consider two methods to find a minimum value:
  - Greedy descent, starting from some position, keep moving down & report minimum value found
  - Pick values at random & report minimum value found
- Which do you expect to work better to find a global minimum?
- Can a mix work better?

# Randomized Greedy Descent

- As well as downward steps we can allow for:
  - Random steps: move to a random neighbor.
  - Random restart: reassign random values to all variables.
- For problems involving a large number of variables, a random restart can be quite expensive.

# 1-Dimensional Ordered Examples

Two 1-dimensional search spaces; 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?

# Stochastic Local Search

Stochastic local search is a mix of:

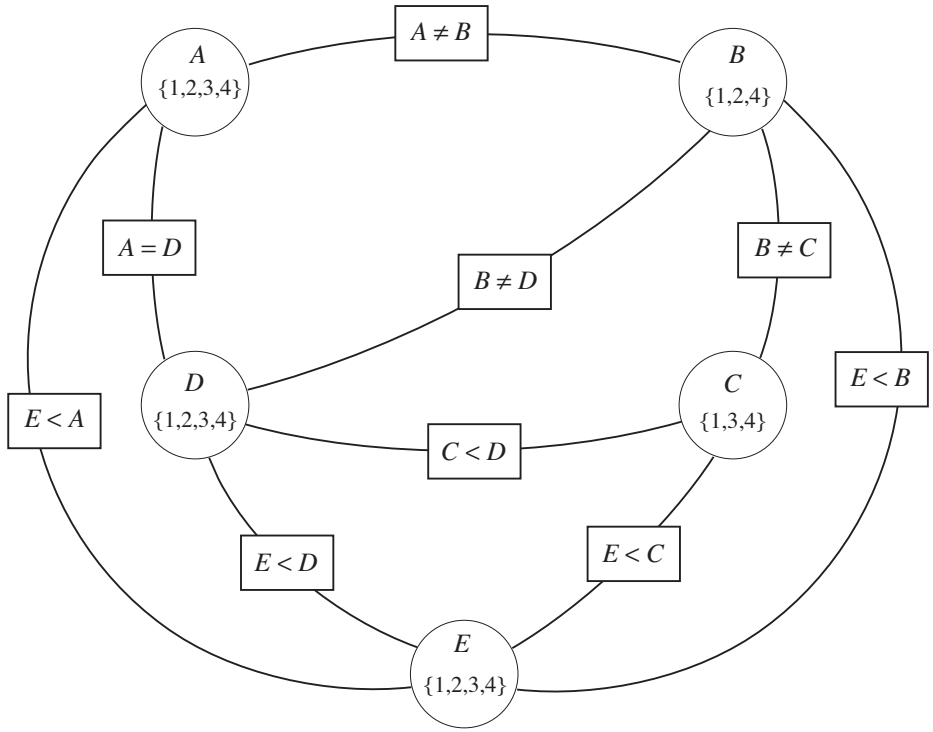
- Greedy descent: move to a lowest neighbor
- Random walk: taking some random steps
- Random restart: reassigning values to all variables

# Example Local Search in delivery scheduling

- Suppose gradient descent starts with the assignment A=2, B=2, C=3, D=2, E=1.
- This assignment has an evaluation of 3, because it does not satisfy A≠B, B≠D, and C<D.
- Its neighbor with the minimal evaluation has B=4 with an evaluation of 1 because only C<D is unsatisfied.
- This is now a local minimum.
- A random walk can then change D to 4, which has an evaluation of 2.
- It can change A to 4, with an evaluation of 2, and then change B to 2 with an evaluation of zero, and a solution is found.
- Trace of the assignments through the walk:

| $A$ | $B$ | $C$ | $D$ | $E$ | evaluation |
|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | 1 | 3 |
| 2 | 4 | 3 | 2 | 1 | 1 |
| 2 | 4 | 3 | 4 | 1 | 2 |
| 4 | 4 | 3 | 4 | 1 | 2 |
| 4 | 2 | 3 | 4 | 1 | 0 |

- Different initializations, or different choices when multiple assignments have the same evaluation, give different results.



# Tabu lists

- 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.

# Random Walk

Variants of random walk:

- When choosing the best variable-value pair, randomly sometimes choose a random variable-value pair.
- When selecting a variable then a value:
  - Sometimes choose any variable that participates in the most conflicts.
  - Sometimes choose any variable that participates in any conflict (a red node).
  - Sometimes choose any variable.
- Sometimes choose the best value and sometimes choose a random value.

# Comparing Stochastic Algorithms

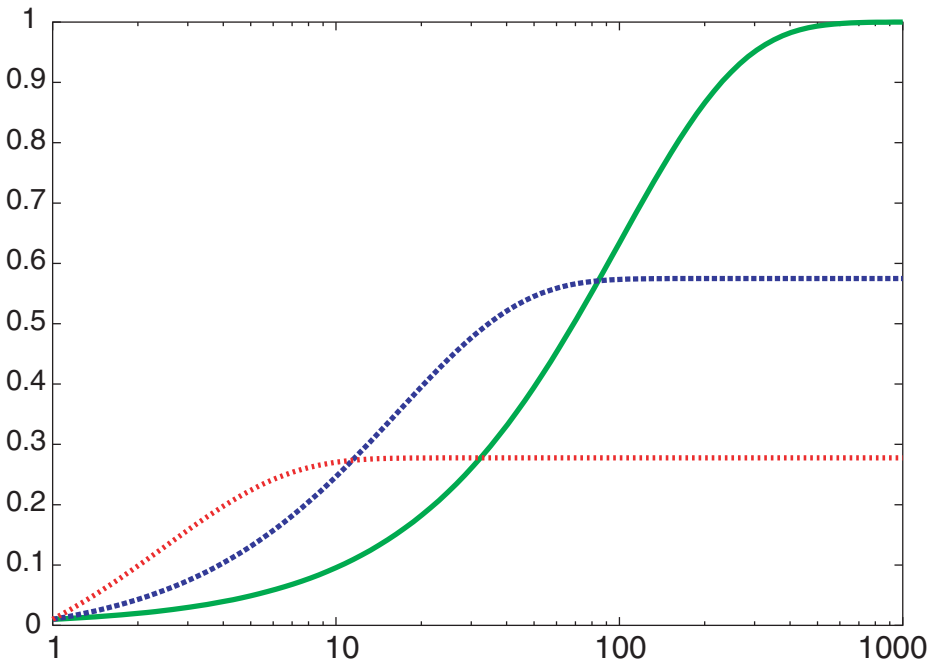
- How can you compare three algorithms when
  - one solves the problem 30% of the time very quickly but doesn't halt for the other 70% of the cases
  - one solves 60% of the cases reasonably quickly but doesn't solve the rest
  - one solves the problem in 100% of the cases, but slowly?

# Comparing Stochastic Algorithms

- 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

Plots runtime (or number of steps) and the proportion of the runs that are solved within that runtime. "Scheduling Problem 1" of `AISpace.org`




Red distribution is for the two-stage greedy descent. Blue distribution is for the one-stage greedy descent. Green distributionis a greedy descent with random walk, where first a random node that participates in a conflict is chosen, then the best value for this variable is chosen with a 50% chance and a random value is chosen otherwise.

Red algorithm solved the problem 40% of the time in 10 or fewer steps. Blue algorithm solved the problem in about 50% of the runs in 10 or fewer steps. Green algorithm found a solution in 10 or fewer steps in about 12% of the runs. Red algorithm found a solution in about 58% of the runs. Blue algorithm could find a solution about 80% of the time. Green algorithm always found a solution.

This only compares the number of steps; the time taken would be a better evaluation.

# Variant: Simulated Annealing

- 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 assignment $n$ and proposed assignment $n'$ we move to $n'$ with probability $e^{(h(n')-h(n))/T}$
- Temperature can be reduced.

# Variant: Simulated Annealing

- 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 assignment $n$ and proposed assignment $n'$ we move to $n'$ with probability $e^{(h(n')-h(n))/T}$
- Temperature can be reduced.

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 | 0.000005 |
| 0.1 | 0.00005 | 0 | 0 |

# Parallel Search

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.

# Beam Search

- Like parallel search, with $k$ individuals, but choose the $k$ best out of all of the neighbors.
- When $k = 1$, it is greedy descent.
- 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 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.

# Genetic Algorithms

- 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 cross-over: form two offspring each taking different parts of their parents:
  - Mutate some values.
- Stop when a solution is found.

# Crossover

- Given two individuals:

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

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

- Select $i$ at random.
- Form two offspring:

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

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

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