Local Search (Greedy Descent):

- Maintain an assignment of a value to each variable.
- Repeat:
  - Select a variable to change
  - Select a new value for that variable
- Until a satisfying assignment is found

1:	<pre>procedure Local-Search(V, dom, C)</pre>			
2:	Inputs			
3:	V: a set of variables			
4:	<i>dom</i> : a function such that $dom(X)$ is the domain of variable X			
5:	C: set of constraints to be satisfied			
6:	Output			
7:	complete assignment that satisfies the constraints			
8:	Local			
9:	A[V] an array of values indexed by V			
10:	repeat			
11:	for each variable X do			
1 <b>2</b> :	$A[X] \leftarrow$ a random value in $dom(X)$ ;			
13:	while stopping criterion not met & A is not a satisfying assignment			
	do			
14:	Select a variable Y and a value $V \in dom(Y)$			
15:	$\operatorname{Set} A[Y] \leftarrow V$			
16:	if A is a satisfying assignment then			
17:	return A			
18:	until termination			
	Figure 4.6: Local search for finding a solution to a CSP			

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

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

- Find a variable-value pair that minimizes the number of conflicts
- Select a variable that participates in the most conflicts. Select a value that minimizes the number of conflicts.
- Select a variable that appears in any conflict. Select a value that minimizes the number of conflicts.
- 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.

- 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 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  $v_i - \eta \frac{\partial h}{\partial X_i}$ .  $\eta$  is the step size.

# 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



- 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?

As well as downward 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?

# 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 is a mix of:

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

Variants of random walk:

- When choosing the best variable-value pair, \_\_\_\_\_\_ 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

• Sometimes choose any variable.

• Sometimes choose the best value and sometimes choose a random value.

< 🗆 I

#### • 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?

### • 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 (or number) of the runs that are solved within that runtime.



< 🗆 )



Figure 4.9: Run-time distributions. These are empirical run-time distributions of 1,000 runs, with each run having a limit of 1,000 steps. On the x-axis is the number of steps (using a logarithmic scale), and on the y-axis is the number of successes out of 1,000 runs. This is for the sample CSP, "Scheduling Problem 1," of Alspace. org. Distributions 1 and 2 are two separate runs for the two-stage greedy descent. Distributions 1 is for the ene-stage greedy descent. Distributions 1 is chosen extra and mode that participates in a conflict (a red node in Alspace.org) is chosen, then the best value for this variable is chosen with a 50% chance and a random value is chosen otherwise.

## 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<sup>(h(n')-h(n))/T</sup>
- Temperature can be reduced.

"annealing schedule" or "cooling schedule"

< 🗆 )

## 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<sup>(h(n')-h(n))/T</sup>
- 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	0.000006
0.1	0.00005	$2 imes 10^{-9}$	$9 imes 10^{-14}$

- 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.

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.
- When  $k = \infty$ , it is breadth-first search.
- The value of k lets us limit space and parallelism.

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

Siven 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:  $X_1 = a_1, \dots, X_i = a_i, X_{i+1} = b_{i+1}, \dots, X_m = b_m$  $X_1 = b_1, \ldots, X_i = b_i, X_{i+1} = a_{i+1}, \ldots, X_m = a_m$  The effectiveness depends on the ordering of the variables. Many variations are possible.

< 🗆 I