Announcements

- Solution to Assignment 3 is posted
- Assignment 4 is available
- Midterm next Thursday.
  - 75 minutes anytime in 24 hour period.
  - Invidualized exams.
  - You may use programs and the Intenet, but you may not not consult or talk to anyone about the exam.
  - Be prepared for an oral exam after to explain how you got your answer.

Review: So far...

- Constraint satisfaction problems are defined in terms of variables, domains, constraints
- Constraint satisfactions problems can be solved with:
  - Search
  - Arc consistency with domain splitting
  - Local search (today!)
- Local search maintains a complete assignment of a value to each variable, and has a mix of improving and randomized steps.

Today: Learning Objectives

Today:

- Assignment 3 solution discussion
- Local Search

At the end of the class you should be able to:

- show how a CSP can be solved using local search
- compare stochastic algorithms
- explain how randomness works

Assignment 3 solution

Clicker Question

Which of the following is false:

A If there is a solution, arc consistency will not make any domains empty

B Arc consistency always halts for finite CSPs

C Arc consistency involves checking constraints multiple times

D Arc consistency always results in singleton domains if there is only one solution.

Clicker Question

What is not true of arc consistency with domain splitting:

A Arc consistency needs domain splitting to solve problems in general

B Together they can solve CSPs in polynomial time

C They typically result in a smaller search space than using search without arc consistency

D They always terminate with a solution if there is one for finite CSPs

Local Search

Local Search:

- Maintain a complete assignment of a value to each variable.
- Start with random assignment or a best guess.

Local Search

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.

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.

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.

Any Conflict

- 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.

Comparing Stochastic Algorithms

Which of the preceding algorithms work better?

Comparing Stochastic Algorithms

Which of the preceding algorithms work better?
How would we tell if one is better than the other?

Comparing Stochastic Algorithms

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?

Comparing Stochastic Algorithms

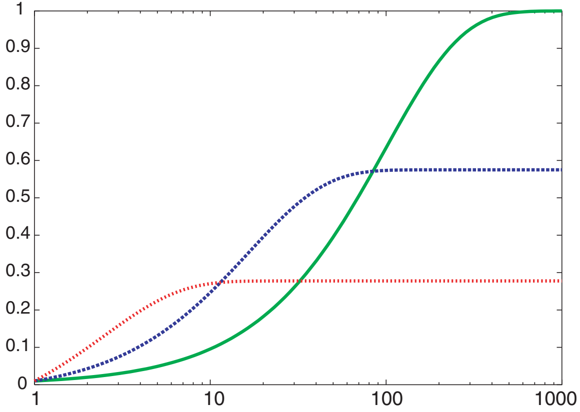
Which of the preceding algorithms work better?
How would we tell if one is better than the other?

- 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)

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.

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

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

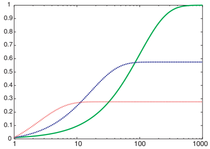
  This produces a cumulative distribution
- Do this this a few times to gauge the variability (take a statistics course!)

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 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



Randomized Algorithms

- 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

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.

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?

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.

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?

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.

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?

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.

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?

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.

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?

Steps

- One measure of an assignment is number of conflicts
- It is possible to weight some conflicts higher than others.
- Why would we?

Steps

- 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....

Steps

- 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,

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.

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}

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.

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.

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

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

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 $1-(1-p)^n$

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

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.

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.

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.

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.

Complex Domains

- When the domains are small or unordered, the neighbors of an assignment can correspond to choosing another value for one of the variables.

Complex 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.

Complex 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 changes each variable proportionally to the gradient of the heuristic function in that direction.
  The value of variable X_i goes from v_i to

Complex Domains

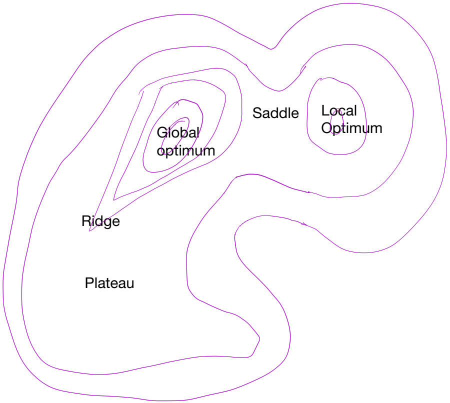
- When the domains are small or unordered, the neighbors of an assignment can correspond to choosing another value for one of the variables.
- When the domains are large and ordered, the neighbors of an assignment are the adjacent values for one of the variables.
- If the domains are continuous, Gradient descent changes each variable proportionally 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.

Complex 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 changes each variable proportionally 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 thousands or millions or billions of dimensions to minimize error on a dataset. (See CPSC 340).

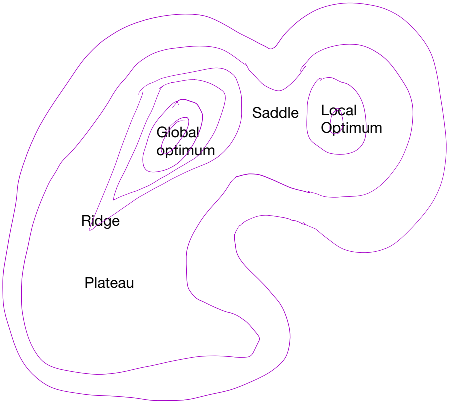
Problems with Greedy Descent

- a local optimum that is not a global optimum



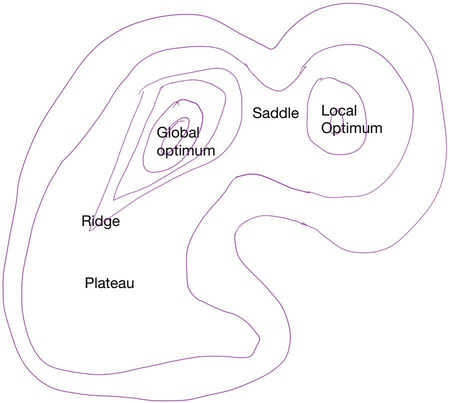
Problems with Greedy Descent

- a local optimum that is not a global optimum
- a plateau where the heuristic values are uninformative



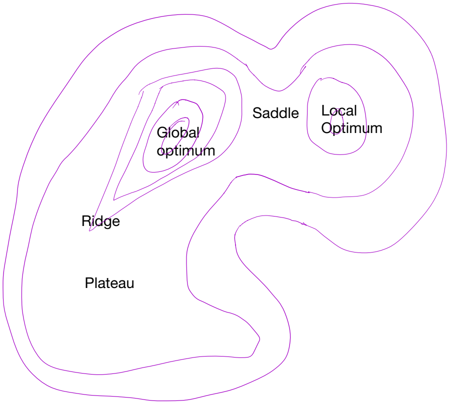
Problems with Greedy Descent

- 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



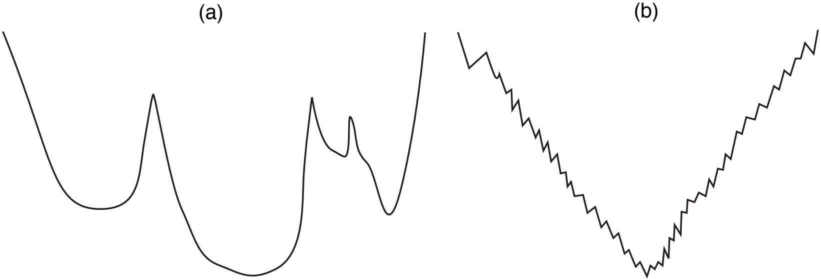
Problems with Greedy Descent

- 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

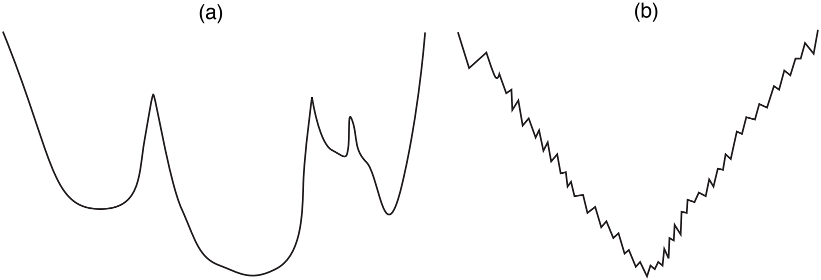
Two 1-dimensional search spaces; small step right or left:



- Which method would most easily find the global minimum?

1-Dimensional Ordered Examples

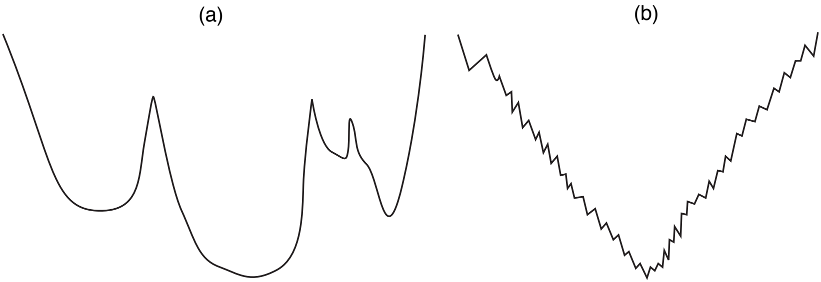
Two 1-dimensional search spaces; small step right or left:



- Which method would most easily find the global minimum?
- What happens in hundreds or thousands of dimensions?

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?

Clicker Question

Which of the following is true:

- A If an algorithm is above and to the left of another algorithm in a runtime distribution, it is always faster
- B A random walk cannot escape a local minima
- C The amount of time taken per step is about the same for all local search methods given modern data structures and the speed of computers
- D Carrying out arc consistency before doing a local search can reduce the search space