Announcements

- Solution to Assignment 3 is posted
- Assignment 4 is available. Use AISpace 1 or AIPython; AISpace 2 is a graphical tracer for AIPython and is not necessary.
- 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
- 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: 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 helps
- know a bit about population methods

Local Search

Local Search:

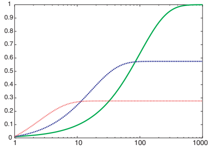
- Maintain a complete assignment of a value to each variable.
- Start with random assignment (or a good guess)
- Repeat:
  - Select a variable to change
  - Select a new value for that variable
- Until a satisfying assignment is found

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

Greedy Descent Variants

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

- 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:

- 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:

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- 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:

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- 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:

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that appears in any conflict.
  Select a value that minimizes the number of conflicts.

Greedy Descent Variants

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

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that appears in any conflict.
  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:

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that appears in any conflict.
  Select a value that minimizes the number of conflicts.
  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:

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that appears in any conflict.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable at random.
  Select a value that minimizes the number of conflicts.
  What needs to be done at every step?

Greedy Descent Variants

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

- Find a variable-value pair that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable that appears in any conflict.
  Select a value that minimizes the number of conflicts.
  What data structures are required?
- Select a variable at random.
  Select a value that minimizes the number of conflicts.
  What needs to be done at every step?
- Select a variable and value at random; accept this change if it doesn't increase the number of conflicts.

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

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
- Temperature can be reduced.

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
- 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	2×10^{-9}	9×10^{-14}

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

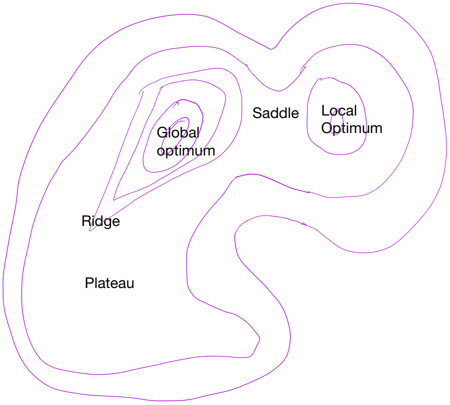
- When the domains are small or unordered, the neighbors of an assignment can correspond to choosing another value for one of the variables.
- When the domains are large and ordered, the neighbors of an assignment are the adjacent values for one of the variables.
- If the domains are continuous, Gradient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

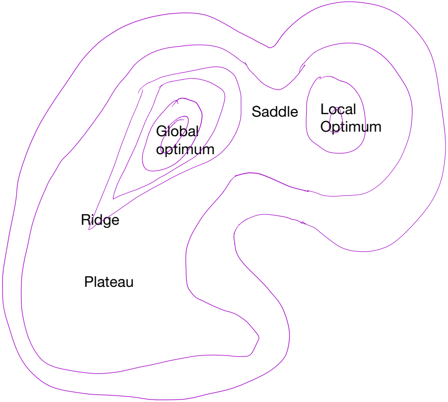
Problems with Greedy Descent

- a local optimum that is not a global optimum



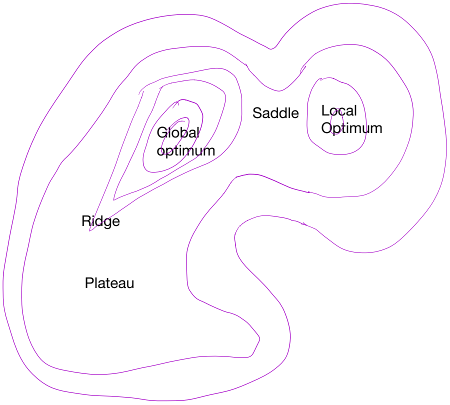
Problems with Greedy Descent

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



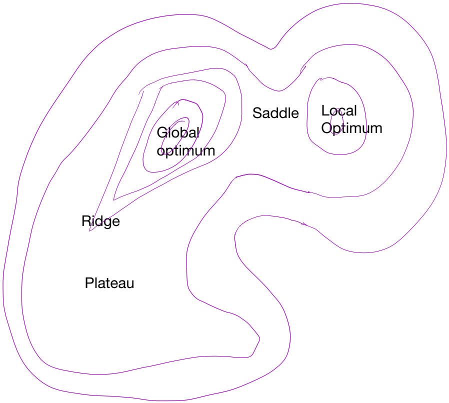
Problems with Greedy Descent

- 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



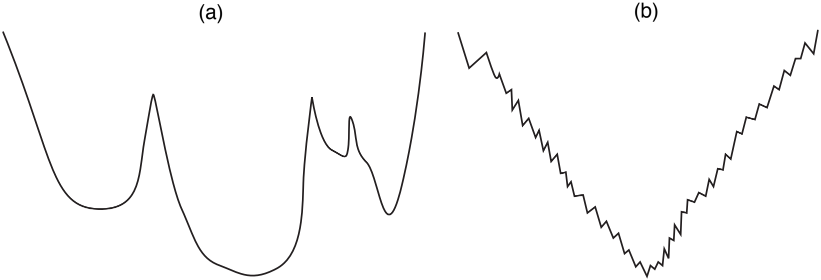
Problems with Greedy Descent

- 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

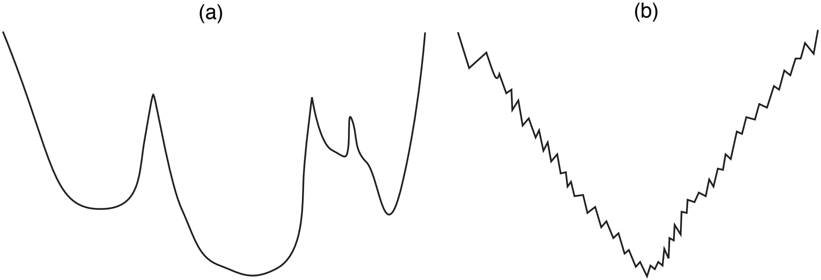
Two 1-dimensional search spaces; small step right or left:



- Which method would most easily find the global minimum?

1-Dimensional Ordered Examples

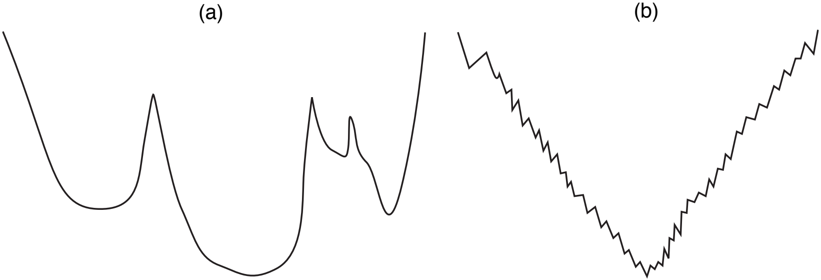
Two 1-dimensional search spaces; small step right or left:



- 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?

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.

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.

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.

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.

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.
- The value of k lets us limit space and parallelism.

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.
- 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.

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.

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.

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 crossover: form two offspring each taking different parts of their parents.

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 crossover: 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.

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.

Clicker Question

Which of the following is false:

- A Population based methods carry out multiple local searches at once
- B The time taken by population-based methods is number of individuals (local searches) multiplied by the *minimum* time one the local searches finds a solution
- C Crossover with selecting fittest individuals allows genetic algorithms to combine good parts of potential solutions
- D It is more likely that a population-based method will find a solution than a local search with no restart
- E Population-based methods are guaranteed to find a solution if there is one, even without randomness

Optimization

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)

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.

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)

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

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

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

Propositional Satisfiability Problems

A Propositional Satisfiability (SAT) Problem is an instance of a CSP with

- Boolean variables: a variable with domain $\{true, false\}$.

Propositional Satisfiability Problems

A Propositional Satisfiability (SAT) Problem is an instance of a CSP with

- Boolean variables: a variable with domain $\{true, false\}$.
  We write $X = true$ as the atom x, and $X = false$ as the $\neg x$, "not x".
  A literal is an atom or the negation of an atom.

Propositional Satisfiability Problems

A Propositional Satisfiability (SAT) Problem is an instance of a CSP with

- Boolean variables: a variable with domain $\{true, false\}$.
  We write $X = true$ as the atom x, and $X = false$ as the $\neg x$, "not x".
  A literal is an atom or the negation of an atom.
- Clausal constraints: a clause is an expression of the form $l_1 \vee l_2 \vee \cdots \vee l_k$, where each l_i is a literal, and $\vee$ means "or".
  The clause is true if at least one of the l_i is true.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.
  - for $i < j$, y_i and y_j cannot both be true, so $\neg y_i \vee \neg y_j$.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.
  - for $i < j$, y_i and y_j cannot both be true, so $\neg y_i \vee \neg y_j$.
  - one of the y_i must be true, so: $y_1 \vee \cdots \vee y_k$.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.
  - for $i < j$, y_i and y_j cannot both be true, so $\neg y_i \vee \neg y_j$.
  - one of the y_i must be true, so: $y_1 \vee \cdots \vee y_k$.
- A clause $\neg x_i \vee \neg y_j \vee \neg z_k$ is equivalent to $\neg(x_i \wedge y_j \wedge z_k)$.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.
  - for $i < j$, y_i and y_j cannot both be true, so $\neg y_i \vee \neg y_j$.
  - one of the y_i must be true, so: $y_1 \vee \cdots \vee y_k$.
- A clause $\neg x_i \vee \neg y_j \vee \neg z_k$ is equivalent to $\neg(x_i \wedge y_j \wedge z_k)$.
  Therefore each false assignment of values can be represented as a clause. So clausal form can represent any finite constraints.

Representing finite CSPs as SAT problems

It is possible to convert any finite CSP into a propositional satisfiable problem:

- A variable Y with domain $\{v_1, \ldots, v_k\}$ can be converted into k Boolean variables $\{Y_1, \ldots, Y_k\}$, where Y_i is true when Y has value v_i and is false otherwise.
  Each Y_i is called an indicator variable.
  - for $i < j$, y_i and y_j cannot both be true, so $\neg y_i \vee \neg y_j$.
  - one of the y_i must be true, so: $y_1 \vee \cdots \vee y_k$.
- A clause $\neg x_i \vee \neg y_j \vee \neg z_k$ is equivalent to $\neg(x_i \wedge y_j \wedge z_k)$.
  Therefore each false assignment of values can be represented as a clause. So clausal form can represent any finite constraints.
  Often we can much more concise.

Consistency Algorithms with SAT

Arc consistency can be made much more efficient in SAT problems than for general CSPs.

- Because domains are binary, pruning a domain is equivalent to assigning a value to the variable.

Consistency Algorithms with SAT

Arc consistency can be made much more efficient in SAT problems than for general CSPs.

- Because domains are binary, pruning a domain is equivalent to assigning a value to the variable.
- If X is assigned *true*, all of the clauses with x can be removed, as they are all satisfied.
- If X is assigned *true*, all of the clauses with $\neg x$, of the form $\neg x \vee w$ can be simplified to w.

Consistency Algorithms with SAT

Arc consistency can be made much more efficient in SAT problems than for general CSPs.

- Because domains are binary, pruning a domain is equivalent to assigning a value to the variable.
- If X is assigned *true*, all of the clauses with x can be removed, as they are all satisfied.
- If X is assigned *true*, all of the clauses with $\neg x$, of the form $\neg x \vee w$ can be simplified to w.
- If we get to a clause with one element, we can assign the corresponding Boolean variable.
- If all of the literals in a clause are removed, there is no solution.

Consistency Algorithms with SAT

Arc consistency can be made much more efficient in SAT problems than for general CSPs.

- Because domains are binary, pruning a domain is equivalent to assigning a value to the variable.
- If X is assigned *true*, all of the clauses with x can be removed, as they are all satisfied.
- If X is assigned *true*, all of the clauses with $\neg x$, of the form $\neg x \vee w$ can be simplified to w.
- If we get to a clause with one element, we can assign the corresponding Boolean variable.
- If all of the literals in a clause are removed, there is no solution.

— uniformity of the constraints means efficient data structures.

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable
- Changing any value in an unsatisfied clause makes the clause satisfied.

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable
- Changing any value in an unsatisfied clause makes the clause satisfied.
- If a variable X is changed to be true
  - all of the clauses containing x become satisfied

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable
- Changing any value in an unsatisfied clause makes the clause satisfied.
- If a variable X is changed to be true
  - all of the clauses containing x become satisfied
  - only those clauses with $\neg x$ can become unsatisfied.

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable
- Changing any value in an unsatisfied clause makes the clause satisfied.
- If a variable X is changed to be true
  - all of the clauses containing x become satisfied
  - only those clauses with $\neg x$ can become unsatisfied.

  This allows for efficient indexing of clauses.

Local search with SAT

Local search can be much more efficient for SAT problems:

- A complete assignment can be represented as a bit-vector
- There is only one alternative value for a variable
- Changing any value in an unsatisfied clause makes the clause satisfied.
- If a variable X is changed to be true
  - all of the clauses containing x become satisfied
  - only those clauses with $\neg x$ can become unsatisfied.

  This allows for efficient indexing of clauses.
- The search space is expanded. Before a solution has been found, more than one of the indicator variables for a variable Y could be true, or all of the indicator variables could be false.