Chapters 4: Feature and Constraints

DIT410/TIN172 Artificial Intelligence

Peter Ljunglöf
modifed from slides by Poole & Mackworth
with some help from slides by Russel & Norvig

31 March, 2015

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)

2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)
2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

States and features

States can often be described in terms of features:

- States can be defined in terms of features: a state corresponds to an assignment of a value to each feature.
- Features can be defined in terms of states: a feature is a function of the states. The function returns the value of the feature on that state.
- Features are described by variables.
- Not all assignments of values to variables are possible.

Examples: 8-queens, crossword puzzle, course timetable.

More difficult: 8-puzzle, driving directions.

States and features

Just a few features can describe many states:

n	binary features can describe	2^n	states
10	binary features can describe	2^{10}	$= 1{,}024$
20	binary features can describe	2^{20}	$= 1{,}048{,}576$
30	binary features can describe	2^{30}	$= 1{,}073{,}741{,}824$
100	binary features can describe	2^{100}	$= 1{,}267{,}650{,}600{,}228{,}229,$ $401{,}496{,}703{,}205{,}376$

Constraint satisfaction problem

Standard search problem:

- the state is a "black box" – any old data structure that supports goal test, cost evaluation, successor

CSP is a more specific search problem:

- the state is defined by *variables* V_i, taking *values* from *domain* $\mathbf{D}_i$
- the goal test is a set of *constraints* specifying allowable combinations of values for subsets of variables

Since CSP is more specific, it allows useful algorithms with more power than standard search algorithms

Hard and soft constraints

Given a set of variables, assign a value to each variable that either

- satisfies some set of constraints:
  - satisfiability problems — "hard constraints"
- minimizes some cost function, where each assignment of values to variables has some cost:
  - optimization problems — "soft constraints"

Many problems are a mix of hard and soft constraints
(called constrained optimization problems)

Relationship to search

Differences to general search problems:

- The path to a goal isn't important, only the solution is.
- There are no predefined starting nodes.
- Often these problems are huge, with thousands of variables, so systematically searching the space is infeasible.
- For optimization problems, there are no well-defined goal nodes.

Posing a CSP

A CSP is characterized by

- A set of variables $V_1, V_2, \ldots, V_n$.
- Each variable V_i has an associated domain $\mathbf{D}_{V_i}$ of possible values.
- There are hard constraints on various subsets of the variables which specify legal combinations of values for these variables.
- A solution to the CSP is an assignment of a value to each variable that satisfies all the constraints.

Example: Scheduling activities

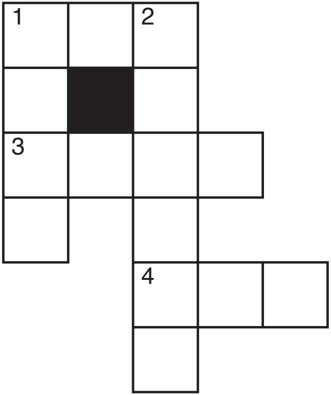
Variables: A, B, C, D, E that represent the starting times of various activities.

Domains: $\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \mathbf{D}_D = \mathbf{D}_E = \{1, 2, 3, 4\}$

Constraints:

$$(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\ (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\ (E < C) \wedge (E < D) \wedge (B \neq D)$$

Example: Crossword puzzle



Words:

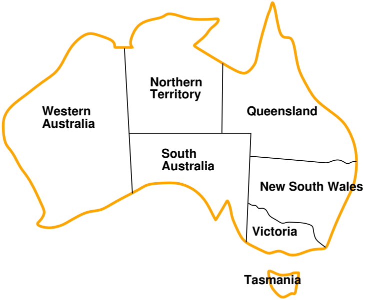
ant, big, bus, car, has
book, buys, hold,
lane, year
beast, ginger, search,
symbol, syntax

Dual representations

Many problems can be represented in different ways as a CSP, e.g., the crossword puzzle:

- First representation:
  - nodes represent word positions: 1-down. . . 6-across
  - domains are the words
  - constraints specify that the letters on the intersections must be the same
- Dual representation:
  - nodes represent the individual squares
  - domains are the letters
  - constraints specify that the words must fit

Example: Map colouring

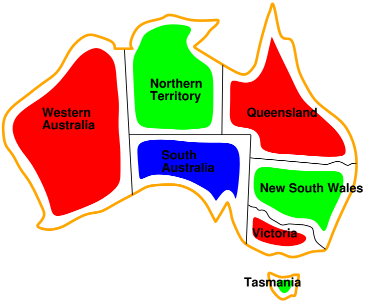


Variables: WA, NT, Q, NSW, V, SA, T

Domains: $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors,
e.g., $WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q, \ldots$

Example: Map colouring



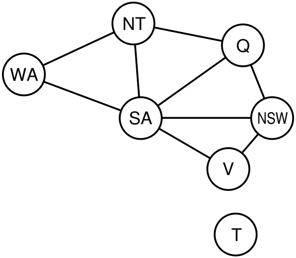
Solutions are assignments satisfying all constraints, e.g.,

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$$

Constraint graph

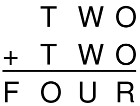
Binary CSP: each constraint relates at most two variables
(*note:* this does not say anything about the domains)

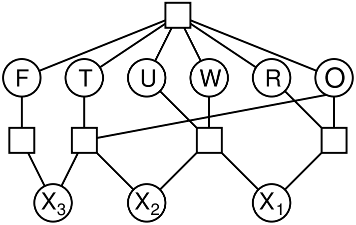
Constraint graph: nodes are variables, arcs show constraints



CSP algorithms can use the graph structure to speed up search,
e.g., Tasmania is an independent subproblem.

Example: Cryptarithmetic puzzle





Variables: $F, T, U, W, R, O, X_1, X_2, X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints: $\mathit{alldiff}(F, T, U, W, R, O)$

$O + O = R + 10 \cdot X_1$, etc.

Note: This is not a binary CSP.

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)

2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Generate-and-test algorithm

- Generate the assignment space $\mathbf{D} = \mathbf{D}_{V_1} \times \mathbf{D}_{V_2} \times \ldots \times \mathbf{D}_{V_n}$. Test each assignment with the constraints.
- Example:

$$
\begin{aligned}
\mathbf{D} &= \mathbf{D}_A \times \mathbf{D}_B \times \mathbf{D}_C \times \mathbf{D}_D \times \mathbf{D}_E \\
&= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \\
&\quad \times \{1,2,3,4\} \times \{1,2,3,4\} \\
&= \{\langle 1,1,1,1,1 \rangle, \langle 1,1,1,1,2 \rangle, \ldots, \langle 4,4,4,4,4 \rangle\}.
\end{aligned}
$$

- How many assignments need to be tested for n variables each with domain size d?

Backtracking algorithms

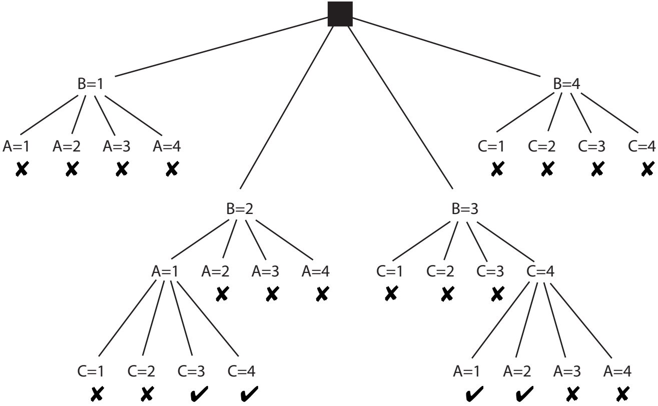
- Explore $\mathbf{D}$ by instantiating the variables one at a time
- Evaluate each constraint as soon as all its variables are bound
- Any partial assignment that doesn't satisfy the constraint can be pruned

Example Assignment $A = 1 \land B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

Simple backtracking example

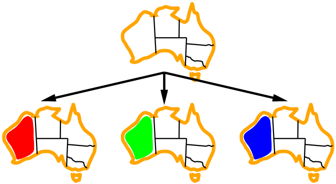
Variables: A, B, C. Domains: $\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \{1,2,3,4\}$.

Constraints: $(A < B) \wedge (B < C)$.

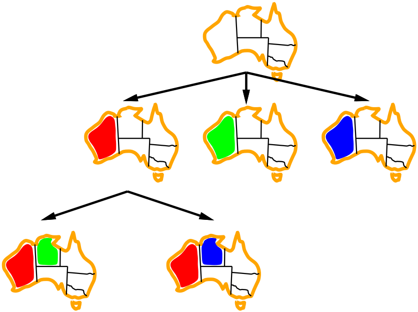


Example: Australia map colours

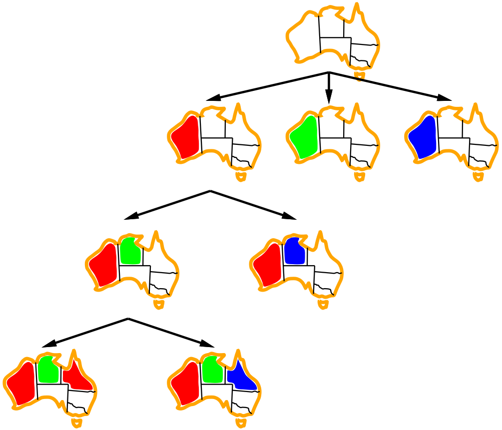
Example: Australia map colours



Example: Australia map colours



Example: Australia map colours



CSP as graph searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node N is the assignment $[X_1 = v_1, \ldots, X_k = v_k]$.
  - Select a variable Y that isn't assigned in N.
  - For each value $y_i \in dom(Y)$, $[X_1 = v_1, \ldots, X_k = v_k, Y = y_i]$ is a neighbour if it is consistent with the constraints.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)
2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Consistency algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.

Example: Is the scheduling example domain consistent?

Consistency algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.

Example: Is the scheduling example domain consistent?

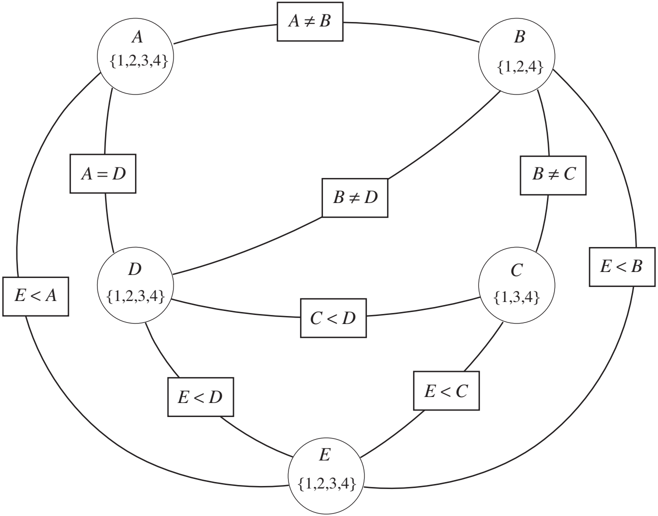
- $\mathbf{D}_B = \{1, 2, 3, 4\}$ is *not* domain consistent, since $B = 3$ violates the constraint $B \neq 3$.
- $\mathbf{D}_C = \{1, 2, 3, 4\}$ is *not* domain consistent, since $C = 2$ violates the constraint $C \neq 2$.

Constraint network

A *constraint network* is a graph, which has:

- an oval-shaped node for each variable,
- a rectangular node for each constraint,
- a domain of values associated with each variable node, and
- an arc from variable X to each constraint that involves X.

Example: Constraint network



Domain consistency vs. arc consistency

- Domain consistency only considers *unary constraints*
  - these are usually not shown in a constraint network
  - because domain consistency is so very easy to check and maintain
- Arc consistency considers *binary* (and more) constraints
  - i.e., the nodes and arcs in the constraint network

Arc consistency

- An arc $\langle X, r(X, Y_1 \dots Y_n)\rangle$ is arc consistent if:
  - for each value $x \in dom(X)$, there is some assignment $y_1 \dots y_n \in dom(Y_1 \dots Y_n)$ such that $r(x, y_1 \dots y_n)$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What if arc $\langle X, r(X, Y_1 \dots Y_n)\rangle$ is *not* arc consistent?
  - all values of X in $dom(X)$ for which there is no corresponding assignment in $dom(Y_1 \dots Y_n)$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, Y_1 \dots Y_n)\rangle$ consistent.

Arc consistency algorithm

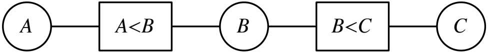
- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?
  - An arc $\langle X, r(X, Y_1 \ldots Y_n)\rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$
  - Each domain has a single value $\Longrightarrow$
  - Some domains have more than one value $\Longrightarrow$

Arc consistency algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?
  - An arc $\langle X, r(X, Y_1 \ldots Y_n)\rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$ no solution
  - Each domain has a single value $\Longrightarrow$ unique solution
  - Some domains have more than one value $\Longrightarrow$ there may or may not be a solution

Quiz: Arc consistency

The variables and constraints are in the constraint graph:



Assume the initial domains are $\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \{1, 2, 3, 4\}$.

How will the domains look like after making the graph arc consistent?

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)
2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Finding solutions when AC finishes

What if some domains have more than one element after AC?

- We can always resort to searching
- Split one of the domains, then recursively solve each half
  - i.e., perform AC on the resulting graph, then split a domain, perform AC, split a domain, perform AC, split, etc.
- It is often best to split a domain in half
  - i.e., if $\mathbf{D}_X = \{1, \ldots, 1000\}$, we can split into $\{1, \ldots 500\}$ and $\{501, \ldots, 1000\}$
- Do we need to restart from scratch?
  - no, only some arcs risk losing their arc consistency after the split

Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)
2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Variable elimination

Complementary simplification methods:

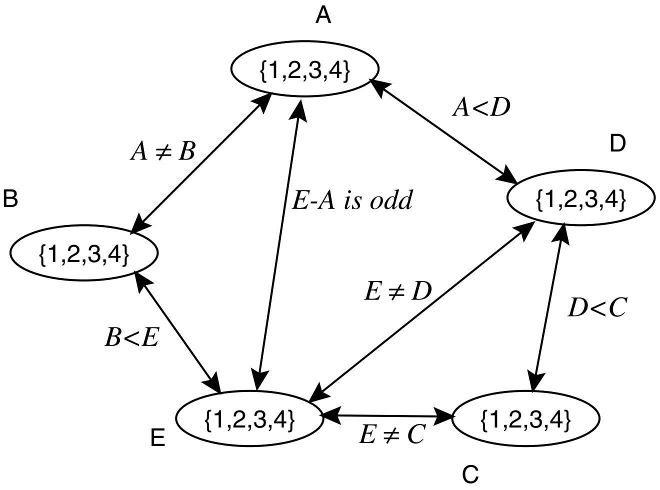
- Arc consistency (AC) simplifies the network by removing values of variables.
- Variable elimination (VE) simplifies the network by removing variables.

Variable elimination algorithm

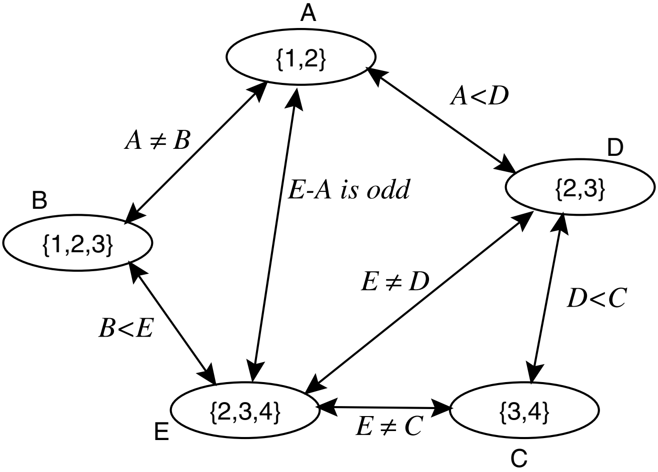
Variable elimination algorithm:

- Select a variable X to eliminate.
  - Remove X by constructing a new constraint on all variables that occur in some X constraint.
  - This new constraint replaces all constraints that involve X, forming a reduced network that does not involve X.
- The variables are eliminated according to some *elimination ordering:*
  - Different elimination orderings can result in different intermediate constraints.

Example network



Example: Arc-consistent network



Example: Eliminating variable C

Constraints: $E \neq C$ and $D < C$.

Domains: $\mathbf{D}_C = \{3, 4\}$, $\mathbf{D}_D = \{2, 3\}$, $\mathbf{D}_E = \{2, 3, 4\}$.

Example: Eliminating variable C

Constraints: $E \neq C$ and $D < C$.

Domains: $\mathbf{D}_C = \{3, 4\}$, $\mathbf{D}_D = \{2, 3\}$, $\mathbf{D}_E = \{2, 3, 4\}$.

$r_1 : E \neq C$	C	E
	3	2
	3	4
	4	2
	4	3

Example: Eliminating variable C

Constraints: $E \neq C$ and $D < C$.

Domains: $\mathbf{D}_C = \{3, 4\}$, $\mathbf{D}_D = \{2, 3\}$, $\mathbf{D}_E = \{2, 3, 4\}$.

$r_1 : E \neq C$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : D < C$	C	D
	3	2
	4	2
	4	3

Example: Eliminating variable C

Constraints: $E \neq C$ and $D < C$.

Domains: $\mathbf{D}_C = \{3, 4\}$, $\mathbf{D}_D = \{2, 3\}$, $\mathbf{D}_E = \{2, 3, 4\}$.

$r_1 : E \neq C$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : D < C$	C	D
	3	2
	4	2
	4	3

$r_3 : r_1 \bowtie r_2$	C	D	E
(join r_1, r_2)	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

Example: Eliminating variable C

Constraints: $E \neq C$ and $D < C$.

Domains: $\mathbf{D}_C = \{3, 4\}$, $\mathbf{D}_D = \{2, 3\}$, $\mathbf{D}_E = \{2, 3, 4\}$.

$r_1 : E \neq C$	C	E
	3	2
	3	4
	4	2
	4	3

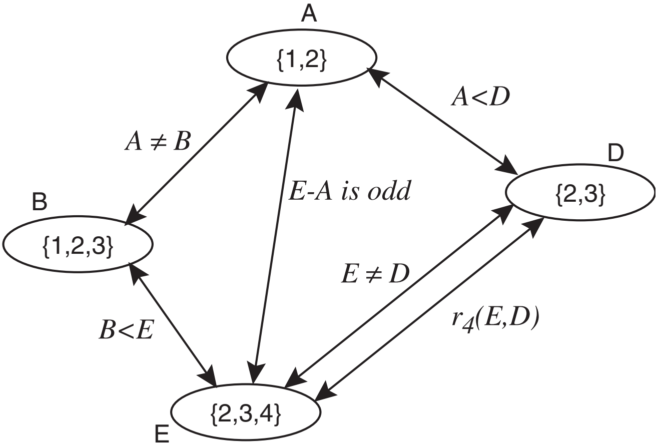
$r_2 : D < C$	C	D
	3	2
	4	2
	4	3

$r_3 : r_1 \bowtie r_2$	C	D	E
(join r_1, r_2)	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

$r_4 : \pi_{\{D,E\}} r_3$	D	E
(project r_3	2	2
onto D, E)	2	3
	2	4
	3	2
	3	3

$\hookrightarrow$ new constraint

Resulting network after eliminating C



Outline

1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)

2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

Local search for CSPs

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 / Greedy descent algorithm:

- Repeat until a satisfying assignment is found:
  - Select a variable to change
  - Select a new value for that variable
- Heuristic function to be minimized: the number of conflicts.

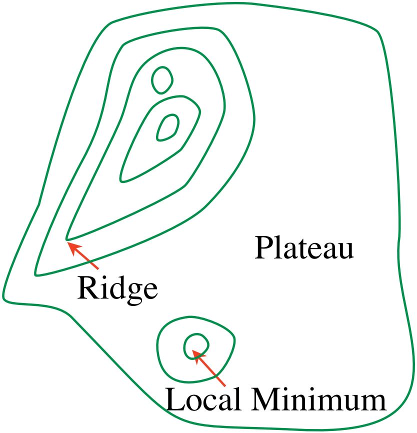
Variants of greedy descent

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.

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



Outline

Randomized algorithms

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

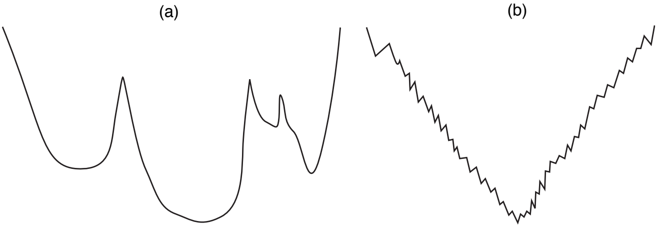
Randomized greedy descent

As well as downward steps we can allow for:

- Random steps: move to a random neighbor.
- Random restart: reassign random values to all variables.

1-dimensional illustrative example

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



- Which method would most easily find the global minimum?
  - random steps or random restarts?
- What happens in hundreds or thousands of dimensions?
  - e.g., different dimensions 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

Outline

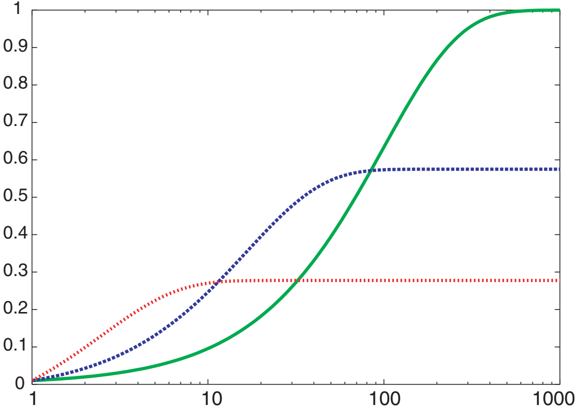
1. *Features and constraints*
   - States, features and constraints (4.1–4.2)
   - Soving CSPs using search (4.3–4.4)
   - Consistency algorithms (4.5)
   - Domain splitting (4.6)
   - Variable elimination (4.7)
2. *Local search (4.8–4.9)*
   - Iterative best improvement (4.8.1)
   - Randomized algorithms (4.8.2)
   - Evaluating randomized algorithms (4.8.3)
   - Population-based methods (4.9)

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



Outline

Beam search

Idea: maintain a population of k assignments in parallel, instead of one:

- At every stage, 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.