Constraint Satisfaction Problems

Greedy Local Search

Bernhard Nebel, **Julien Hué**, and **Stefan Wölfl**

Albert-Ludwigs-Universität Freiburg

June 19, 2007

Greedy Local Search

Constraint solving techniques so far discussed:

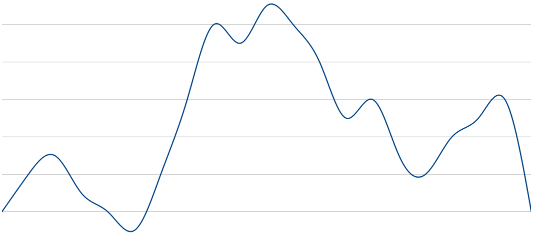
- Inference
- Search
- Combinations of inference and search
  $\rightsquigarrow$ improve overall performance; nevertheless worst-time complexity is high

$\Rightarrow$ approximate solutions, for example, by greedy local search methods

$\Rightarrow$ in particular of interest, when we look at optimization problems (e.g. traveling salesman problem, minimize violations of so-called soft constraints)

Example

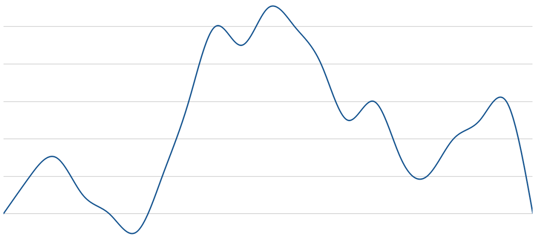
Principle of Stochastic Local Search



Etymology: Greek stokhastikos, from stokhasts, diviner, from stokhazesthai, to guess at
Stochastic (Wiktionary): Relating to stochastics.

Example

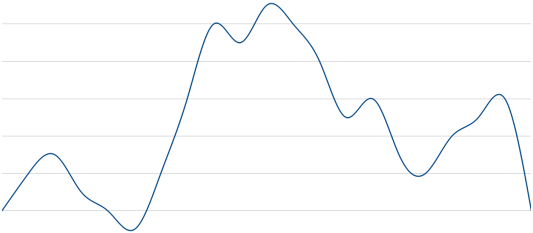
Principle of Stochastic Local Search



Etymology: Greek stokhastikos, from stokhasts, diviner, from stokhazesthai, to guess at
Stochastic (Wiktionary): Relating to stochastics.
Stochastics (Wiktionary): The branch of statistics that deals with stochastic systems

Example

Principle of Stochastic Local Search



Etymology: Greek stokhastikos, from stokhasts, diviner, from stokhazesthai, to guess at
Stochastic (FreeDictionary): Involving or containing a random variable or variables

Greedy algorithm

A first method: greedy heuristics

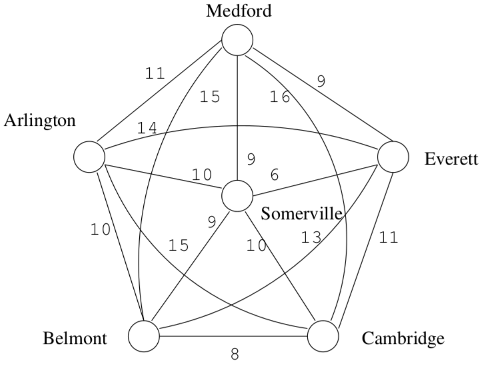
- Building step by step a solution $(v_1 \mapsto x_1, \ldots, v_n \mapsto x_n)$
- Generic Algorithm:
  - $s \leftarrow \emptyset$
  - While s is not a total assignment
  - Pick a variable v_i and a value $x_i \in D_i$.
  - $v_i \leftarrow x_i$
  - EndWhile

  Is actually backtracking without BT.

Sometimes some pretreatment are realized before the greedy part.

Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

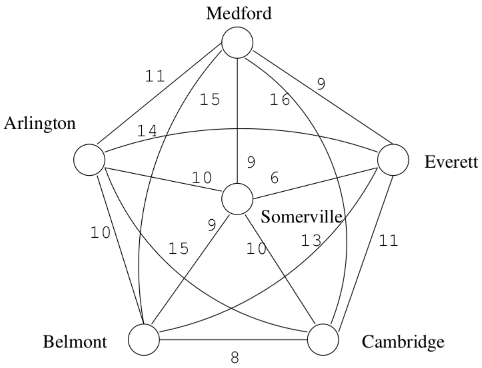


Objective: Find a path going through all the nodes with minimal cost.

Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

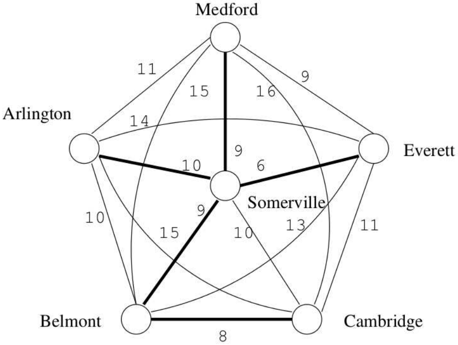
Find the minimum spanning tree T.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

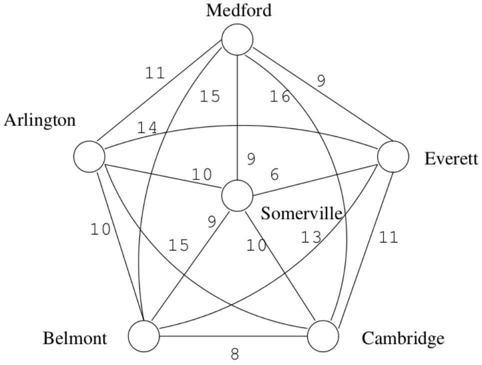
Find the minimum spanning tree T.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

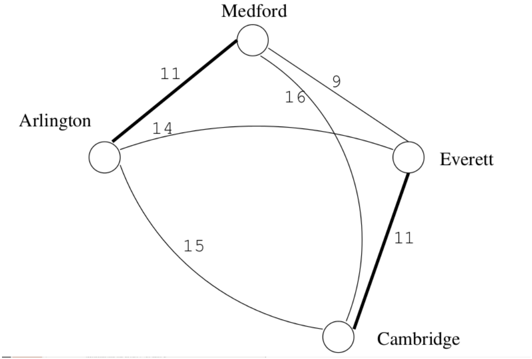
Find a perfect matching G^* for the graph restricted to the vertices with an odd degree.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

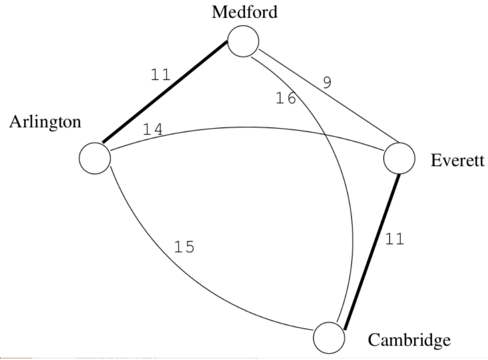
Find a perfect matching G^* for the graph restricted to the vertices with an odd degree.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

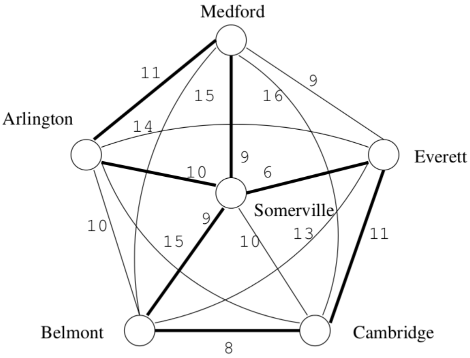
Merge G^* and T.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

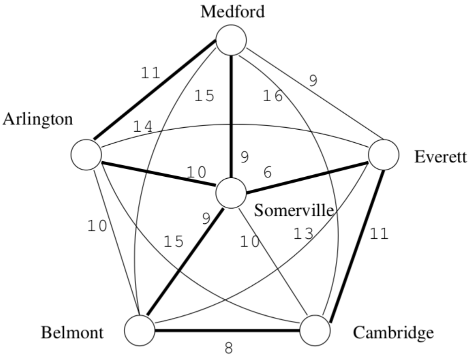
Merge G^* and T.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

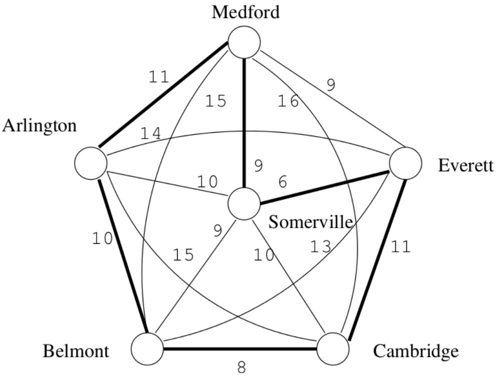
Create an Eulerian tour using the triangle inequality.



Example: Christofides Algorithm

Sometimes run a greedy algorithm after a pretreatment.
Example: the Christofides Algorithm for the TSP

Create an Eulerian tour using the triangle inequality.



The solution is always at most 3/2 of the optimal solution.

Stochastic Greedy Local Search

Stochastic Greedy Local Search (SLS)

Features:

- greedy, hill-climbing traversal of the search space
- in particular, no guarantee to find a solution even if there is one
- search space: states correspond to complete assignment of values to all variables of the constraint network, which are not necessarily solutions of the network
- no systematic search

The SLS-Algorithm

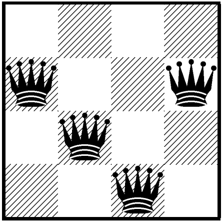
SLS $(\mathcal{C}, \textbf{max\_tries}, \textbf{cost})$:

Input: a constraint network $\mathcal{C}$, a number of tries max_tries, a cost function cost

Output: A solution of $\mathcal{C}$ or "false"

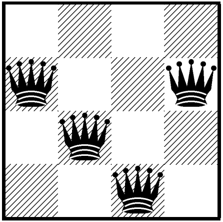
repeat max_tries times
  instantiate a complete random assignment $\overline{a} = (a_1, \ldots, a_n)$
  repeat
    if $\overline{a}$ is consistent **then return** $\overline{a}$
    else let Y be the set of assignments that differ from $\overline{a}$ in exactly one variable-value pair (i.e., change one v_i value a_i to a new value a_i')
      $\overline{a} \leftarrow$ choose an $\overline{a}\,'$ from Y with maximal cost improvement
    endif
  until current assignment cannot be improved
endrepeat
return "false"

Example

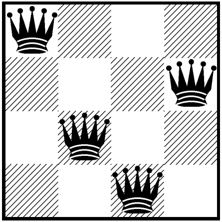


$c(a) = 4$

Example

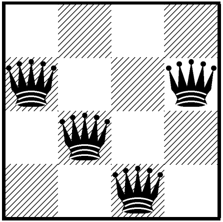


$c(a) = 4$

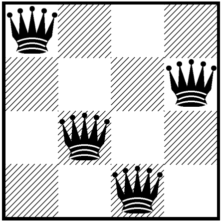


$c(a) = 1$

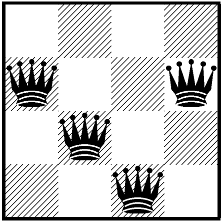
Example



$c(a) = 4$

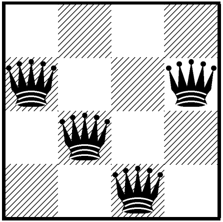


$c(a) = 1$

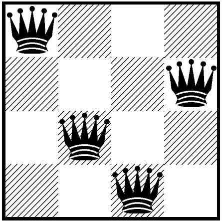


$c(a) = 4$

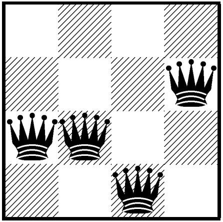
Example



$c(a) = 4$

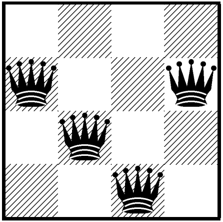


$c(a) = 1$

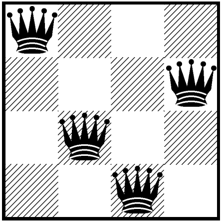


$c(a) = 2$

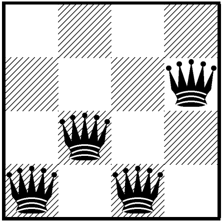
Example



$c(a) = 4$

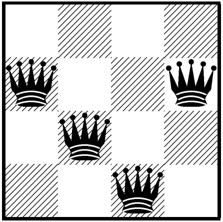


$c(a) = 1$

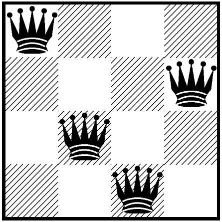


$c(a) = 3$

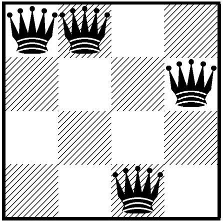
Example



$c(a) = 4$

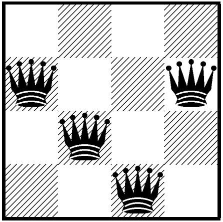


$c(a) = 1$

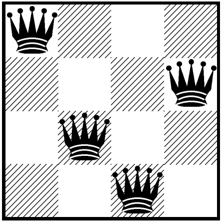


$c(a) = 1$

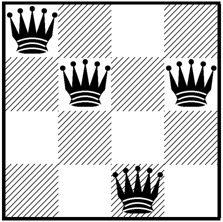
Example



$c(a) = 4$

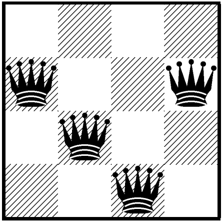


$c(a) = 1$

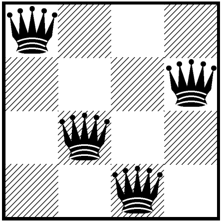


$c(a) = 2$

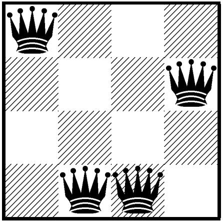
Example



$c(a) = 4$

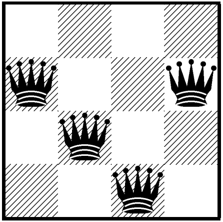


$c(a) = 1$

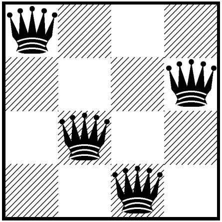


$c(a) = 2$

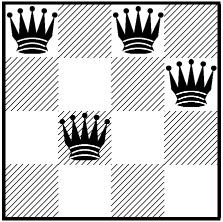
Example



$c(a) = 4$

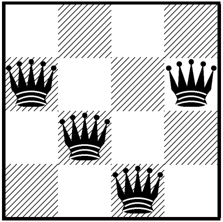


$c(a) = 1$

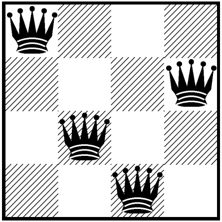


$c(a) = 2$

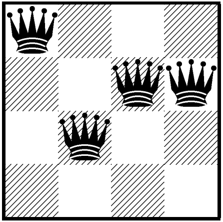
Example



$c(a) = 4$

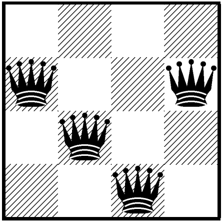


$c(a) = 1$

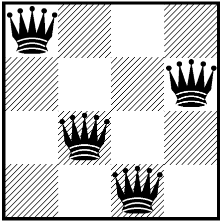


$c(a) = 2$

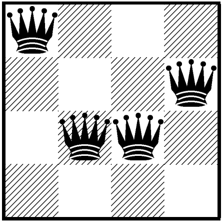
Example



$c(a) = 4$

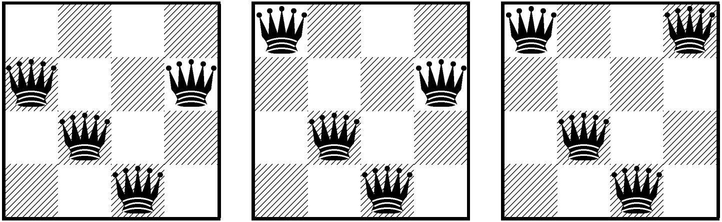


$c(a) = 1$



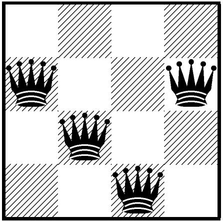
$c(a) = 3$

Example

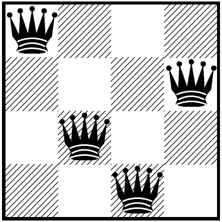


$c(a) = 4$ $c(a) = 1$ $c(a) = 3$

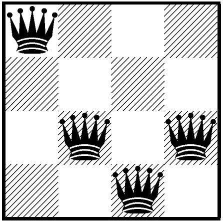
Example



$c(a) = 4$

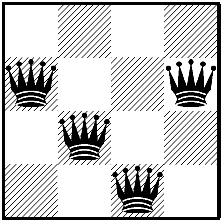


$c(a) = 1$

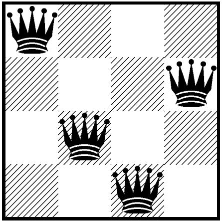


$c(a) = 3$

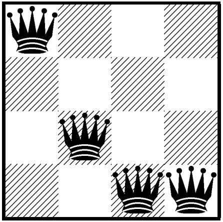
Example



$c(a) = 4$

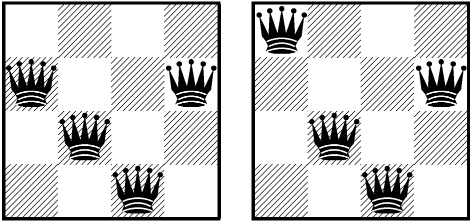


$c(a) = 1$



$c(a) = 3$

Example



$c(a) = 4$ $\qquad$ $c(a) = 1$

. . . is a local minimum, from which we cannot escape in SLS

Improvements

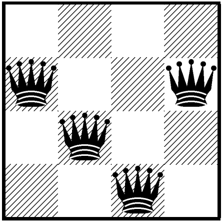
In principal, there are two ways for improving the basic SLS-algorithm:

- different strategies for escaping local minima
- other policies for performing local changes

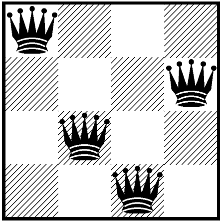
Heuristics for Escaping Local Minima

- Plateau Search: allow for continuing search by sideways moves that do not improve the assignment

Example: Plateau search



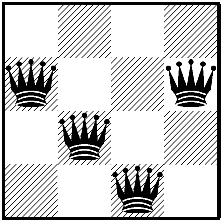
$c(a) = 4$



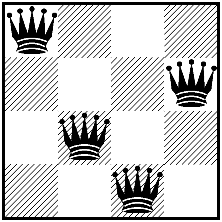
$c(a) = 1$

...is a local minimum, from which we cannot escape in SLS

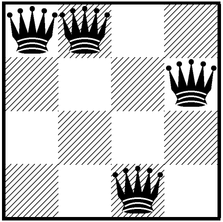
Example: Plateau search



$c(a) = 4$

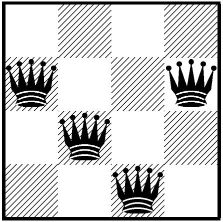


$c(a) = 1$

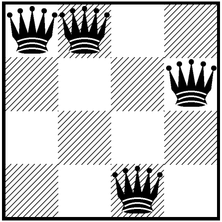


$c(a) = 1$

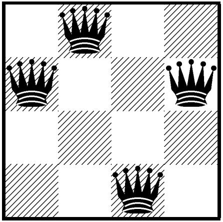
Example: Plateau search



$c(a) = 4$

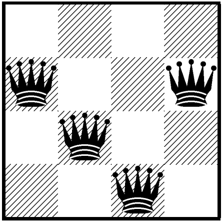


$c(a) = 1$

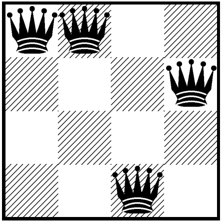


$c(a) = 1$

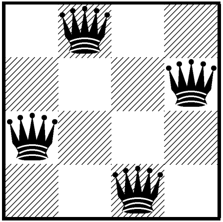
Example: Plateau search



$c(a) = 4$



$c(a) = 1$



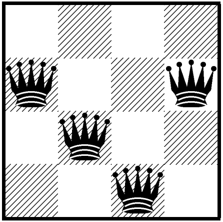
$c(a) = 0$

Heuristics for Escaping Local Minima

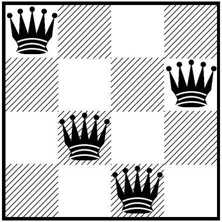
- Constraint weighting/ breakout method: as a cost measure use a weighted sum of violated constraints; initial weights are changed when no improving move is available. *Idea:* if no change reduces the cost of the assignment, increase the weight of those constraints that are violated by the current assignment.

Example: Plateau search

$$w(1,2) = 1 \; w(1,3) = 1 \; w(1,4) = 1$$
$$w(2,3) = 1 \; w(2,4) = 1 \; w(3,4) = 1$$



$c(a) = 4$

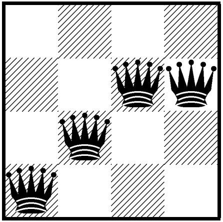


$c(a) = 1$

. . . is a local minimum, from which we cannot escape in SLS

Example: Plateau search

$w(1,2) = 1$ $w(1,3) = 1$ $w(1,4) = 1$
$w(2,3) = 2$ $w(2,4) = 1$ $w(3,4) = 1$



$c(a) = 5$

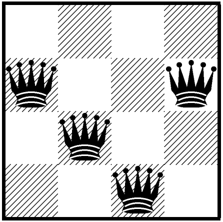
...Now the constraint between 2 and 3 is considered more important

Heuristics for Escaping Local Minima

- Tabu search: prevent cycling over assignments of the same cost. For this, maintain a list of "forbidden" assignments, called tabu list (usually a list of the last n variable-value assignments). The list is updated whenever the assignment changes. Then changes to variable assignments are only allowed w.r.t. to variable-value pairs not in the tabu list.

Example: Plateau search

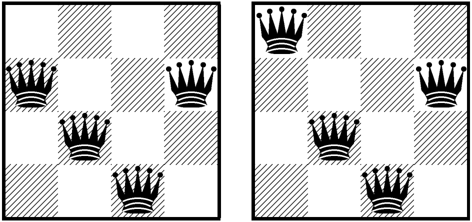
Tabu list: $\{$ (3213) $\}$



$c(a) = 4$

Example: Plateau search

Tabu list: { (3213) (4213) }

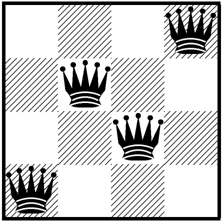


$c(a) = 4$ $c(a) = 1$

...local optimum

Example: Plateau search

Tabu list: { (3213) (4213) (1324) }

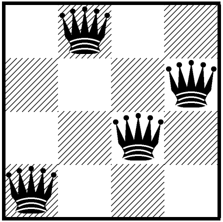


$c(a) = 2$

. . . restart but no possible improvement

Example: Plateau search

Tabu list: $\{$ (3213) (4213) (1324) (1423) $\}$



$c(a) = 1$

. . . restart and so on

Random Walk Strategies

Random Walk

Random walk strategy:

- combines random walk search with a greedy approach (bias towards assignments that satisfy more constraints)
- instead of making greedy moves in each step, sometimes perform a random walk step
- for example, start from a random assignment. If the assignment is not a solution, select randomly an unsatisfied constraint and change the value of one of the variables participating in the constraint.

WalkSAT

WalkSAT:

- initially formulated for SAT solving
- turns out to be very successful (in empirical studies)
- based on a two-stage process for selecting variables: in each step select first a constraint violated by the current assignment; second make a random choice between
  a) changing the value of one of the variables in the violated constraint;
  b) minimizing in a greedy way the break value, i.e., the number of new constraints that become inconsistent by changing a value

  The choice between (a) and (b) is controlled by a parameter p (probability for (a))

WalkSAT ($\mathcal{C}$, **max_flips**, **max_tries**):

Input: a constraint network $\mathcal{C}$, numbers max_flips (flips) and max_tries (tries)
Output: "true" and a solution of $\mathcal{C}$, or
"false" and some inconsistent best assignment

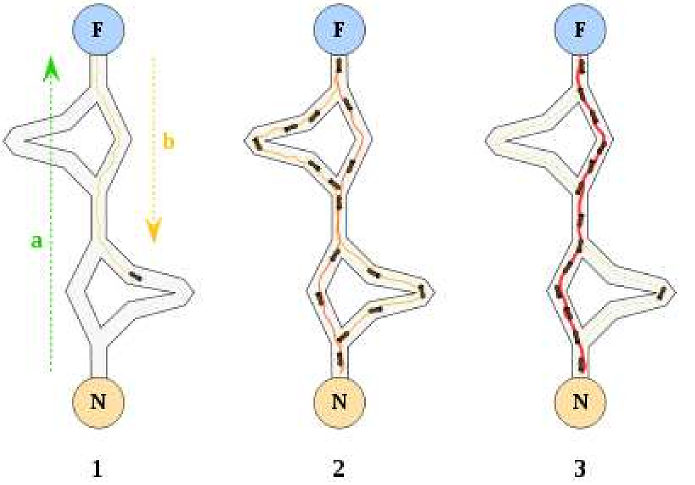
$\overline{a}' \leftarrow$ a complete random assignment
repeat max_tries times
    $\overline{a} \leftarrow$ a complete random assignment
    repeat max_flips times
        if $\overline{a}$ is consistent **then return** "true" and $\overline{a}$
        else select a violated constraint R_S with scope S
            with probability p: choose an arbitrary variable-value pair (x, a'),
                $x \in S, \overline{a}[x] \neq a'$
            else (with probability $1 - p$): choose a variable-value pair (x, a'),
                $x \in S, \overline{a}[x] \neq a'$, that maximizes the number of satisfied
                constraints when x's value in $\overline{a}$ is changed to a'
            $\overline{a} \leftarrow \overline{a}$ with $x \mapsto a'$
        endif
    endrepeat
    compare $\overline{a}$ with $\overline{a}'$ and retain the better one as $\overline{a}'$
endrepeat
return "false" and $\overline{a}'$

Simulated Annealing

Simulated Annealing:

- *Idea:* over time decrease the probability of doing a random move over one that maximally decreases costs. Metaphorically speaking, by decreasing the probability of random moves, we "freeze" the search space.
- At each step, select a variable-value pair and compute the change of the cost function, δ, when the value of the variable is changed to the selected value. Change the value if δ is not negative (i.e., costs do not increase). Otherwise, we perform the change with probability $e^{-\delta/T}$ where T is the temperature parameter.
- If the temperature T decreases over time, more random moves are allowed at the beginning and less such moves at the end.

Simulated Annealing to its best: Ant Colony Optimization

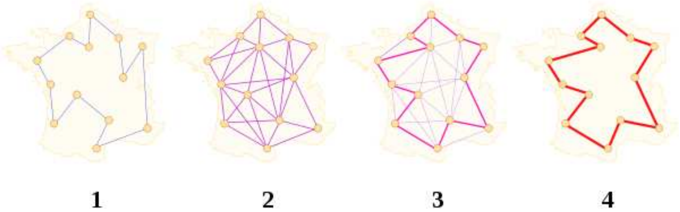


Courtesy of the wikipedia page.

Simulated Annealing to its best: Ant Colony Optimization

- An ant runs at random around the colony;
- If it discovers a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone;
- These pheromones are attractive, nearby ants will be inclined, with a given percentage, to follow the track;
- Returning to the colony, these ants will strengthen the route;
- If there are two routes to reach the same food source then the shorter one will be traveled by more ants;
- The short route will be increasingly enhanced, and therefore become more attractive;
- The long route will disappear because pheromones are volatile;
- Eventually, all the ants have chosen the shortest route.

Simulated Annealing to its best: Ant Colony Optimization



Courtesy of the wikipedia page.

Hybrid algorithms

Hybrids of Local Search and Inference

SLS-algorithms can also be combined with inference methods. For example, apply SLS only after preprocessing a given CSP instance with some consistency-enforcing algorithm.

Idea: Can we improve SLS by looking at equivalent but more explicit constraint networks?

Note:

- there are classes of problems, e.g., 3SAT problems, which can easily be solved by a systematic backtracking algorithm, but are hard to be solved via SLS
- consistency-enforcing algorithms can change the costs associated to an arc in the constraint graph drastically: assignments near to a solution (in terms of costs) may be very far from a solution after applying inference methods

Example:

- Local search on cycle cutsets

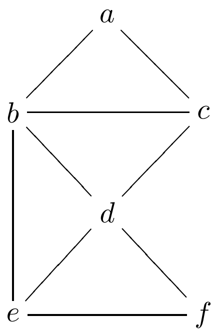
Local Search on Cycle Cutsets

Idea for a hybrid algorithm:

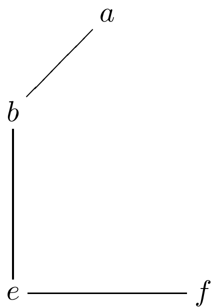
1. Determine a cycle cutset
2. Find some assignment for the cutset variables
3. Propagate values, i.e., find assignment for the tree variables that minimize costs (how do we do that?)
4. Do stochastic local search by varying the cutset variables only
5. Continue with step 3 if there was some improvement
6. Otherwise stop

Usually outperforms pure SLS, provided the cutset is small ($\leq$ 30%).

Cycle-cutset:an example



Cycle-cutset:an example



Now, the remaining constraint graph is backtrack-free.

MinCostTree $(\mathcal{C}, Y, Z, \overline{y})$:

Input: constraint network $\mathcal{C}$, cutset variables Y and tree variables Z with $Y \cup Z = V$ and a partial assignment $\overline{y}$ to the cutset variables
Output: assigment $\overline{z}$ to the variables Z minimizing constraint violations
Comment: $R_{z_i,z_j}(a_i, a_j) = 1$ if $(a_i, a_j) \in R_{z_i,z_j}$, otherwise it is 0.
Compute costs for z_i under $\overline{y}$ for each $a_i \in \mathrm{dom}(z_i)$: $C_{z_i}(a_i, \overline{y})$

foreach $y_i \in Y$ **do** $C_{y_i}(\overline{y}[i], \overline{y}) \leftarrow 0$ **endfor**
foreach $z_i \in Z$ going from leaves to the roots **do**
$C_{z_i}(a_i, \overline{y}) \leftarrow$
$\sum_{z_j \text{child of} z_i} \min_{a_j \in \mathrm{dom}(z_j)}(C_{z_j}(a_j, \overline{y}) + R_{z_i,z_j}(a_i, a_j))$
endfor
foreach $z_i \in Z$ going from the roots to the leaves **do**
$\overline{z}[i] \leftarrow \arg\min_{a_i \in \mathrm{dom}(z_i)}(C_{z_i}(a_i, \overline{y}) + R_{z_i,z_{p_i}}(a_i, a_{p_i}))$
provided z_{p_i} is the parent of z_i
endfor
return $\overline{z}$

Summary

Properties of Stochastic Local Search

SLS algorithms . . .

- are anytime: the longer the run, the better the solution they produce (in terms of a cost function counting violated constraints)
- terminate at local minima
- cannot be used to prove inconsistency of CSP instances

However, WalkSAT can be shown to find a satisfying assignment with probability approaching 1, provided the procedure can run long enough (exponentially long) and provided such an assignment exists.

Literature

Rina Dechter.
Constraint Processing,
Chapter 7, Morgan Kaufmann, 2003