Constraint Satisfaction Problems

Greedy Local Search

Malte Helmert 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)

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

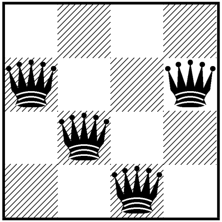
The SLS-Algorithm

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"

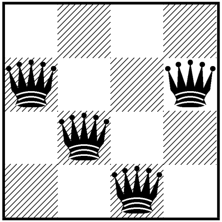
repeat max_tries times
    instantiate a complete random assignment $\bar{a} = (a_1, \dots, a_n)$
    repeat
        if $\bar{a}$ is consistent **then return** $\bar{a}$
        else let Y be the set of assignments that differ from $\bar{a}$ in exactly one variable-value pair (i. e., change one v_i value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

Example

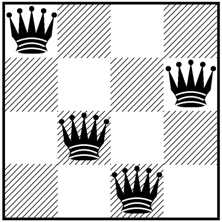


$c(a) = 4$

Example

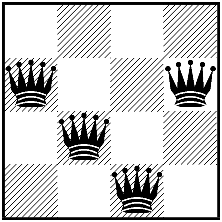


$c(a) = 4$

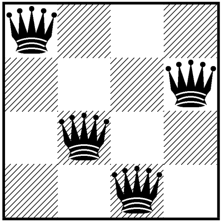


$c(a) = 1$

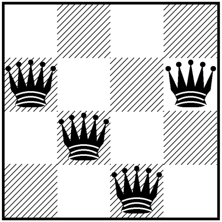
Example



$c(a) = 4$

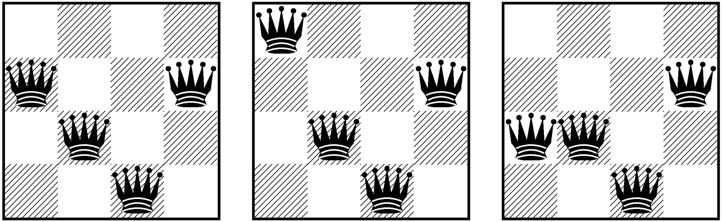


$c(a) = 1$



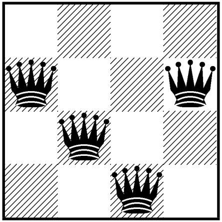
$c(a) = 4$

Example

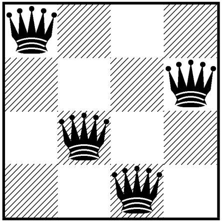


$c(a) = 4$ $c(a) = 1$ $c(a) = 2$

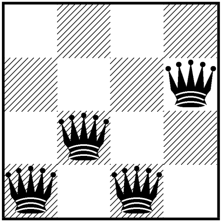
Example



$c(a) = 4$

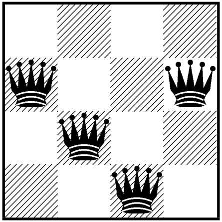


$c(a) = 1$

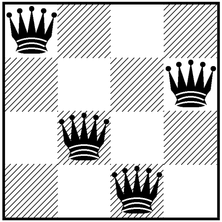


$c(a) = 3$

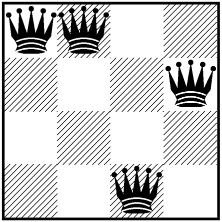
Example



$c(a) = 4$

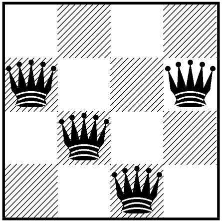


$c(a) = 1$

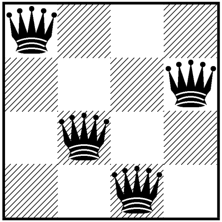


$c(a) = 1$

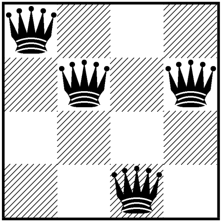
Example



$c(a) = 4$

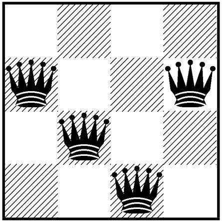


$c(a) = 1$

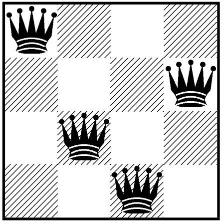


$c(a) = 2$

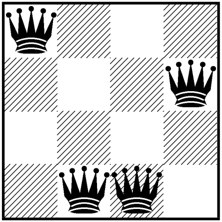
Example



$c(a) = 4$

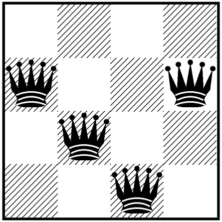


$c(a) = 1$

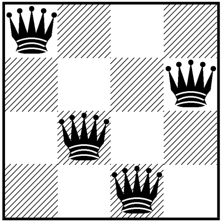


$c(a) = 2$

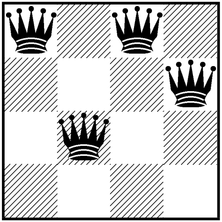
Example



$c(a) = 4$

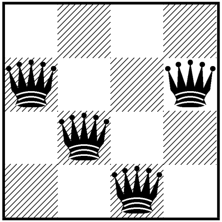


$c(a) = 1$

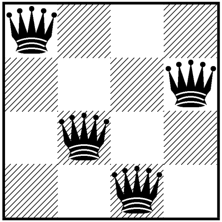


$c(a) = 2$

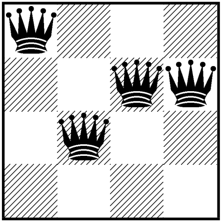
Example



$c(a) = 4$

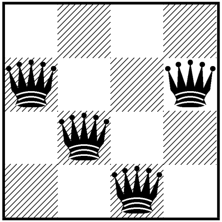


$c(a) = 1$

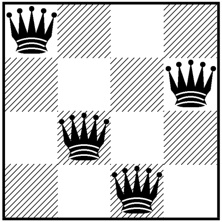


$c(a) = 2$

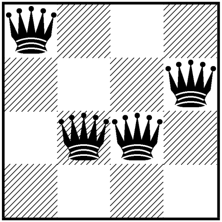
Example



$c(a) = 4$

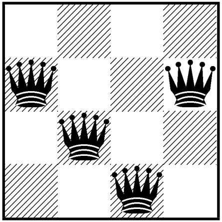


$c(a) = 1$

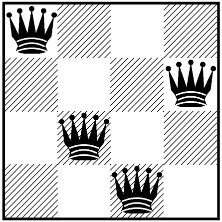


$c(a) = 3$

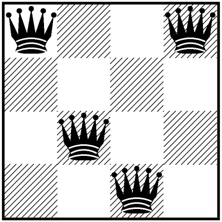
Example



$c(a) = 4$

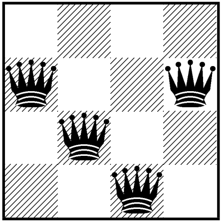


$c(a) = 1$

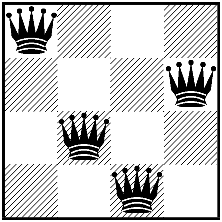


$c(a) = 3$

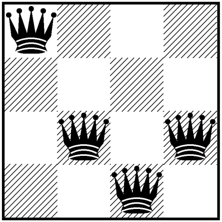
Example



$c(a) = 4$

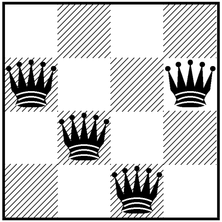


$c(a) = 1$

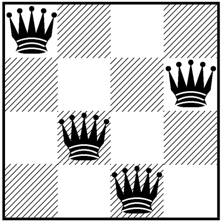


$c(a) = 3$

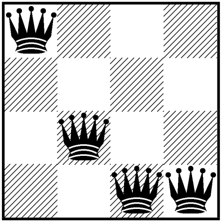
Example



$c(a) = 4$

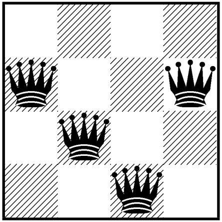


$c(a) = 1$

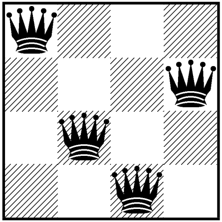


$c(a) = 3$

Example



$c(a) = 4$



$c(a) = 1$

. . . 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
- 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.
- 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.

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 $(a_1, \ldots, a_n)$
repeat max_tries times
  instantiate a complete random assignment $\overline{a}'$
  compare $\overline{a}$ with $\overline{a}'$ and retain the better one as $\overline{a}$
  repeat max_flips times
    if $\overline{a}$ is consistent **then return** "true" and $\overline{a}$
    else select a violated constraint
      with probability p choose an arbitrary variable-value pair (x, a') or,
        with probability $1-p$, choose a variable-value pair (x, a') that
        minimizes the number of new constraints that break when x's
        value is changed to a' (-1 if the current constraint is satisfied)
      $\overline{a} \leftarrow \overline{a}$ with $x \mapsto a'$
    endif
  endrepeat
endrepeat
return "false" and 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.

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., 3-SAT 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

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 a probability near to 1, if such an assignment exists.

Literature

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