Constraint Satisfaction Problems

Greedy Local Search

Albert-Ludwigs-Universität Freiburg

Stefan Wölfl, **Christian Becker-Asano**, and **Bernhard Nebel**

December 8 & 10, 2014

Greedy local search

Constraint solving techniques so far discussed:

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

⇒ approximate solutions, for example, by greedy local search methods

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

1 Stochastic Greedy Local Search

- Escaping Local Minima

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

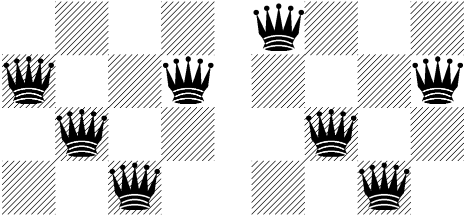
SLS $(N, \boldsymbol{max\_tries}, \boldsymbol{cost})$:

Input: a constraint network N, a number of tries *max_tries*, a cost function *cost*

Output: A solution of N or "failure"

```
repeat max_tries times
    instantiate a complete random assignment $a = (d_1, \ldots, d_n)$
    repeat
        if $a$ is consistent then return $a$
        else   let $Y$ be the set of assignments that differ from $a$ in
               exactly one variable-value pair (i.e., change one $v_i$'s value
               $d_i$ to a new value $d_i'$)
            $a \leftarrow$ choose an $a'$ from $Y$ with maximal cost improvement
        endif
    until current assignment cannot be improved
endrepeat
```

Example: SLS



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

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

Improvements

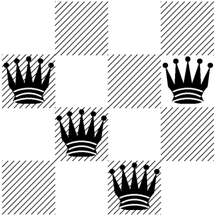
In principal, there are two ways for improving the basic SLS-algorithm:

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

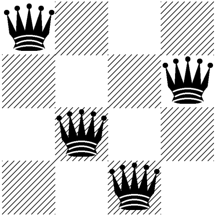
Heuristics for escaping local minima

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

Example: Plateau search

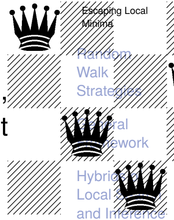


$c(a) = 4$



$c(a) = 1$

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



$c(a) = 1$

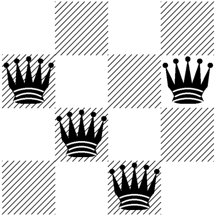
Heuristics for escaping local minima

- 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.

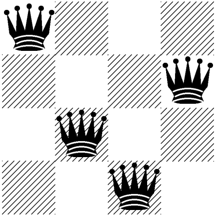
  Idea: if no change reduces the cost of the assignment, increase the weight of those constraints that are violated by the current assignment.

Example: Constraint weighting

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

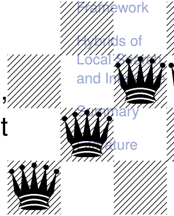


$c(a) = 4$



$c(a) = 1$

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



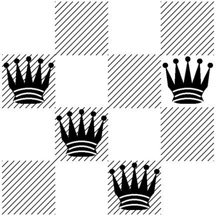
$c(a) = 5$

Heuristics for escaping local minima

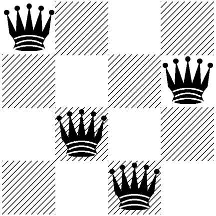
- Tabu search: prevent cycling over assignments of the same cost. For this, maintain a list of “forbidden” assignments, called tabu list (usually a list of the last *n* variable-value assignments). The list is updated whenever the assignment changes. Then changes to variable assignments are only allowed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

Example: Tabu search

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



$c(a) = 4$



$c(a) = 1$

…local optimum

$c(a) = 2$

2 Random Walk Strategies

- WalkSAT
- Simulated Annealing

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 (by Selman, Kautz, & Cohen: WALKSAT/SKC)
- 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 (N, ***max_flips***, ***max_tries***):

Input: a constraint network N, numbers *max_flips* (flips) and *max_tries* (tries)
Output: “true” and a solution of N, or
“failure” and some inconsistent best assignment

```
$a' \leftarrow$ a complete random assignment
repeat max_tries times
    $a = (d_1, \ldots, d_n) \leftarrow$ a complete random assignment
    repeat max_flips times
        if $a$ is consistent then return “true” and $a$
        else select a violated constraint $C$ with scope $s$
            with probability $p$: choose an arbitrary variable-value pair $(v_i, d')$,
                $v_i \in s, d_i \neq d'$
            else (with probability $1 - p$): choose a variable-value pair $(v_i, d')$,
                $v_i \in s, d_i \neq d'$, that maximizes the number of satisfied
                constraints when $v_i$'s value in $a$ is changed to $d'$
            $a \leftarrow (a$ with $v_i \mapsto d')$
        endif
    endrepeat
    compare $a$ with $a'$ and retain the better one as $a'$
endrepeat
return “failure” 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 ($T \geq 0$).
- The temperature T is decreased over time (schedule): more random moves are allowed at the beginning and less such moves at the end.

Simulated annealing

SimulatedAnnealing (*N*, ***schedule***):

Input: a constraint network *N*, a cost function *cost* and a schedule *schedule* mapping time to temperature
Output: A solution candidate to *N*

```
instantiate a complete random assignment $a = (d_1, \ldots, d_n)$
for $t = 1, 2, 3, \ldots$
    $T \leftarrow schedule(t)$
    if $T = 0$ then return $a$
    $a' \leftarrow$ a complete random assignment
    $\delta \leftarrow cost(a) - cost(a')$
    if $\delta > 0$ then $a \leftarrow a'$
    else $a \leftarrow a'$ with probability $e^{\delta/T}$
endfor
```

3 General Framework

Terminology

Stochastic local search methods can be applied to many combinatorial problems (such as CSP). An abstract characterization of these methods is as follows:

Given a combinatorial problem X a stochastic local search algorithm for solving instances x of X is specified by:

- the search space S_x of x (elements are referred to as locations, positions, or configurations)
- a set of feasible solutions $S_x^* \subseteq S_x$
- a neighborhood relation N_x on S_x representing which positions can be reached from another position in one search step.
- a (finite) set M_x of memory states (representing, e.g., previously visited states)
- . . .

Terminology

- …
- an initialization method $init_x$ that specifies the initialization of the search: the result is a probability distribution over $S_x \times M_x$
- a step function $step_x : S_x \times M_x \to \Pi(S_x \times M_x)$, assigning to each position and memory state a probability distribution over the neighboring positions and memory states
- a termination function $terminate_x : S_x \times M_x \to \Pi(\{0,1\})$, providing a probability distribution of the probability by which the search is terminated when the search has reached a certain position and memory state.

. . . applied to CSP context

Given a constraint network *N*:

- Search space: the set of all complete assignments of *N*
- Solutions: the consistent assignments (solutions) of *N*
- Neighborhood: typically 1-exchange neighborhood, i.e., two positions are considered neighbor if they differ at most in the assignment of a single variable (in SAT: 1-flip neighborhood)
- Initialization: mostly random assignment (with uniform distribution)
- Step function: this is where most algorithms we saw differ i.e., these algorithm use different heuristics for selecting the next step
- . . .

Completeness?

Definition (Hoos)

A stochastic local search algorithm is probabilistically approximately complete (PAC) if on all solvable instances the probability that the algorithm finds a solution of the instance within time t goes to 1 as t goes to ∞.

Notice:

> Assume that the neighborhood relation is connected (each position is reachable from each other position) and all search steps have a probability > 0. Then purely random walk (no heuristic guidance) has the PAC property.

Heuristics

Most heuristics use an evaluation function g mapping assignments to non-negative real numbers such that the global minima of g correspond to solutions.

In the CSP context, g is most of the times simply chosen such that the number of violated constraints are counted (see previous slides).

Most popular heuristics is the min-conflict heuristic: randomly select new value for a variable randomly selected from the variables in some unsatisfied constraint under the current assignment such that the number of unsatisfied constraints is minimized (see SLS).

Random walk steps and restart

To escape local minima, random walk steps are performed (this often guarantees PAC property), in particular, the random walk probability (*noise setting*) must be > 0.

If the random walk steps modify the assignment of variable in an unsatisfied constraint, we say that the random walks are conflict-directed.

Random walks can be combined with restarts . . . (see WalkSAT).

Does this pay off? PAC-property when the number of restarts (max_tries) is fixed? Experimental results crucially depend on instances and the settings of the parameters used.

4 Hybrids of Local Search and Inference

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?

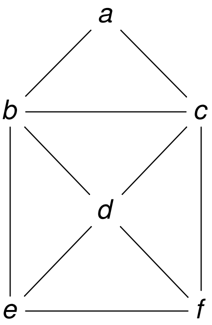
Note:

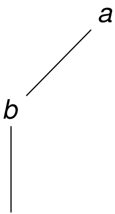
- there are classes of problems, e.g., 3SAT problems, which can easily be solved by a systematic backtracking algorithm, but are hard to be solved via SLS
- consistency-enforcing algorithms can change the costs associated to an arc in the constraint graph drastically: assignments near to a solution (in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

Example:

- Local search on cycle cutsets

Cycle-cutset:an example





Local search on cycle cutsets

Idea for a hybrid algorithm:

1. Determine a cycle cutset
2. Find some assignment for the cutset variables
3. Find assignment for the tree variables that minimizes costs, given the assignment to the cutset variables
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%).

5 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
- Holger H. Hoos & Edward Tsang.
  Local Search Methods,
  Chapter 5 of Handbook of Constraint Programming, Elsevier, 2006