Stochastic greedy local search

Chapter 7

ICS-275
Winter 2016
Example: 8-queen problem
Main elements

- Choose a full assignment and iteratively improve it towards a solution
- Requires a cost function: number of unsatisfied constraints or clauses. Neural networks use energy minimization
- Drawback: local minimas
- Remedy: introduce a random element
- Cannot decide inconsistency
Algorithm Stochastic Local search (SLS)

**Procedure SLS**

**Input:** A constraint network $\mathcal{R} = (X, D, C)$, number of tries $\text{MAX\_TRIES}$. A cost function.

**Output:** A solution iff the problem is consistent, ”false” otherwise.

1. **for** $i=1$ to $\text{MAX\_TRIES}$
   
   • **initialization:** let $\tilde{a} = (a_1, ..., a_n)$ be a random initial assignment to all variables.
   
   • **repeat**
     
     (a) if $\tilde{a}$ is consistent, return $\tilde{a}$ as a solution.
     
     (b) **else** let $Y = \{< x_i, a'_i > \}$ be the set of variable-value pairs that when $x_i$ is assigned $a'_i$, give a significant increase in the cost of the assignment; pick a pair $< x_i, a'_i > \in Y$, $\tilde{a} \leftarrow (a_1, ..., a_{i-1}, a'_i, a_{i+1}, ..., a_n)$ (just flip $a_i$ to $a'_i$).
   
   • **until** the current assignment cannot be improved.

2. **endfor**

3. return **false**
Example 7.1 Consider the formula $\varphi = (A \lor E \lor F) \land (A \lor F \lor E)$
Assume that in the initial assignment, all variables are assigned the value "1". This assignment violates two clauses, the first and the last, so the cost is 2. Next we see that flipping A, E or D will not remove any inconsistency. Flipping C to "0" will satisfy the two violated clauses but will violate the clause $(\neg A \lor \neg B \lor C)$, yielding a cost of 1. Flipping B to $\neg B$ will remove one inconsistency and has a cost of 1 as well. If we flip C to $\neg C$, and subsequently flipping B to $\neg B$ yields a cost of 0 – and a solution.

- Example: z divides y,x,t z = \{2,3,5\}, x,y = \{2,3,4\}, t=\{2,5,6\}
Heuristics for improving local search

- **Plateau search:** at local minima continue search sideways.

- **Constraint weighting:** use weighted cost function
  - The cost $C_i$ is 1 if no violation. At local minima increase the weights of violating constraints.
  \[ F(\tilde{a}) = \sum w_i C_i(\tilde{a}) \]

- **Tabu search:**
  - prevent backwards moves by keeping a list of assigned variable-values. Tie-breaking rule may be conditioned on historic information: select the value that was flipped least recently

- **Automating Max-flips:**
  - Based on experimenting with a class of problems
  - Given a progress in the cost function, allow the same number of flips used up to current progress.
Random walk strategies

- Combine random walk with greediness
  - At each step:
    - choose randomly an unsatisfied clause.
    - with probability $p$ flip a random variable in the clause, with $(1-p)$ do a greedy step minimizing the breakout value: the number of new constraints that are unsatisfied
Figure 7.2: Algorithm WalkSAT

Procedure WalkSAT

Input: A network $\mathcal{R} = (X, D, C)$, number of flips MAX_FLIPS, MAX_TRIES, probability $p$.

Output: True iff the problem is consistent, false otherwise.

1. For $i = 1$ to MAX_TRIES do

2. Compare best assignment with $\bar{a}$ and retain the best.
   (a) \textbf{start} with a random initial assignment $\bar{a}$.
   (b) \textbf{for} $i = 1$ to MAX_FLIPS
     - if $\bar{a}$ is a solution, return \textbf{true} and $\bar{a}$.
     - else,
       i. \textbf{pick} a violated constraint $C$, randomly
       ii. \textbf{choose} with probability $p$ a variable-value pair $< x, a' >$ for $x \in \text{scope}(C)$, or, with probability $1 - p$, a variable-value pair $< x, a'' >$ for $x \in \text{scope}(C)$, or, with probability $1 - p$, a variable-value pair $< x, a'' >$ for $x \in \text{scope}(C)$, and change the value of $x$ when the value of $x$ is changed to $a'$, (minus 1 if the current constraint is satisfied).
       iii. Change $x$'s value to $a'$.
     \textbf{endfor}

3. \textbf{endfor}

4. return \textbf{false} and the best current assignment.
Example of walkSAT:
start with assignment of true to all vars

Example 7.2 Following our earlier example 7.1.1, we will first select an unsatisfied clause, such as $(\neg B \lor \neg C)$, and then select a variable. If we try to minimize the number of additional constraints that would be broken, we will select $B$ and flip its value. Subsequently, the only unsatisfied clause is $\neg C$ which is selected and flipped.

\[(\neg C), (\neg A \lor \neg B \lor C)(\neg A \lor D \lor E)(\neg B \lor \neg C)\]
Simulated Annealing  (Kirkpatric, Gellat and Vecchi (1983))

- Pick randomly a variable and a value and compute delta: the change in the cost function when the variable is flipped to the value.
- If change improves execute it,
- Otherwise it is executed with probability $e^{-\frac{\delta}{T}}$ where T is a temperature parameter. 
- The algorithm will converge if T is reduced gradually.
Properties of local search

- Guarantee to terminate at local minima
- Random walk on 2-sat is guaranteed to converge with probability 1 after $N^2$ steps, when N is the number of variables.
- Proof:
  - A random assignment is on the average $N^2$ flips away from a satisfying assignment.
  - There is at least $\frac{1}{2}$ chance that a flip of a 2-clause will reduce the distance to a given satisfying assignment by 1 (because the satisfying assignment assigns true to at least one literal in a randomly picked unsatisfied 2-clause, so at least 50% chance a flip will satisfy it).
  - Random walk will cover this distance in $N^2$ steps on the average.
- Analysis breaks for 3-SAT
- Empirical evaluation shows good performance compared with complete algorithms (see chapter and numerous papers)
## Comparing various styles of SLS

<table>
<thead>
<tr>
<th>formula</th>
<th>basic</th>
<th>GSAT walk</th>
<th>noise</th>
<th>Simul. Ann.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vars</td>
<td>clauses</td>
<td>time</td>
<td>flips</td>
<td>R</td>
</tr>
<tr>
<td>100</td>
<td>430</td>
<td>.4</td>
<td>7554</td>
<td>8.3</td>
</tr>
<tr>
<td>200</td>
<td>860</td>
<td>22</td>
<td>284693</td>
<td>143</td>
</tr>
<tr>
<td>400</td>
<td>1700</td>
<td>122</td>
<td>2.6x10^6</td>
<td>67</td>
</tr>
<tr>
<td>600</td>
<td>2550</td>
<td>1471</td>
<td>30x10^6</td>
<td>500</td>
</tr>
<tr>
<td>800</td>
<td>3400</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1000</td>
<td>4250</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2000</td>
<td>8480</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Table 7.1: Comparing noise strategies on hard random 3CNF instances.
The random walk strategy (i.e., GSAT +walk) augments GSAT as follows: with a probability \( p \), pick a variable occurring in an unsatisﬁed clause and ﬂip its truth value. With probability \( 1-p \) do a regular greedy step.

The random noise strategy is the same except the variable can be picked from any clause. Both random walk and random noise diﬀer from WalkSAT in a subtle way. For each strategy the table gives the average time in seconds it took to ﬁnd a satisfying assignment, the average number of ips it required, and \( R \), the average number of restarts needed before ﬁnding a solution. At least 100 random restarts (MAX-TRIES setting in GSAT) were applied on each problem instance (but in some cases the strategy was restarted up to 1,000 times). The parameters of each method were varied and optimized empirically. For details see [257].
Comparing DPLL and local search on circuit synthesis problems

<table>
<thead>
<tr>
<th>formula</th>
<th>id</th>
<th>vars</th>
<th>clauses</th>
<th>DP time</th>
<th>GSAT+w time</th>
<th>WSAT time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2bitadd_12</td>
<td>708</td>
<td>1702</td>
<td>*</td>
<td>0.081</td>
<td>0.013</td>
<td></td>
</tr>
<tr>
<td>2bitadd_11</td>
<td>649</td>
<td>1562</td>
<td>*</td>
<td>0.058</td>
<td>0.014</td>
<td></td>
</tr>
<tr>
<td>3bitadd_32</td>
<td>8704</td>
<td>32316</td>
<td>*</td>
<td>94.1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>3bitadd_31</td>
<td>8432</td>
<td>31310</td>
<td>*</td>
<td>456.6</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>2bitcomp_12</td>
<td>300</td>
<td>730</td>
<td>23096</td>
<td>0.009</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>2bitcomp_5</td>
<td>125</td>
<td>310</td>
<td>1.4</td>
<td>0.009</td>
<td>0.001</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2: Comparing complete DPLL method (DP) with local search strategies on circuit synthesis problems. (Timings in seconds.)
Comparing DPLL against local search

<table>
<thead>
<tr>
<th>formula id</th>
<th>vars</th>
<th>clauses</th>
<th>DP time</th>
<th>GSAT+w time</th>
<th>WSAT time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2bitadd_12</td>
<td>708</td>
<td>1702</td>
<td>*</td>
<td>0.081</td>
<td>0.013</td>
</tr>
<tr>
<td>2bitadd_11</td>
<td>649</td>
<td>1562</td>
<td>*</td>
<td>0.058</td>
<td>0.014</td>
</tr>
<tr>
<td>3bitadd_32</td>
<td>8704</td>
<td>32316</td>
<td>*</td>
<td>94.1</td>
<td>1.0</td>
</tr>
<tr>
<td>3bitadd_31</td>
<td>8432</td>
<td>31310</td>
<td>*</td>
<td>456.6</td>
<td>0.7</td>
</tr>
<tr>
<td>2bitcomp_12</td>
<td>300</td>
<td>730</td>
<td>23096</td>
<td>0.009</td>
<td>0.002</td>
</tr>
<tr>
<td>2bitcomp_5</td>
<td>125</td>
<td>310</td>
<td>1.4</td>
<td>0.009</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 7.2: Comparing complete DPLL method (DP) with local search strategies on circuit synthesis problems. (Timings in seconds.)
Hybrids of local search and inference

- We can use exact hybrids of search+inference and replace search by SLS (Kask and Dechter 1996)
  - Good when cutset is small
- The effect of preprocessing by constraint propagation on SLS (Kask and Dechter 1995)
  - Great improvement on structured problems
  - Not so much on uniform problems
SLS and Local Consistency

- **Structured** (hierarchical 3SAT cluster structures) vs. (uniform) random.

**Basic scheme**:
- Apply preprocessing (resolution, path consistency)
- Run SLS
- Compare against SLS alone

What can we say about local search when we have the minimal network?
SLS and Local Consistency

A 3SAT clause over variables x, y and z
SLS and SLS on structured problems

Table 1: Bound-3 Resolution and GSAT

<table>
<thead>
<tr>
<th>C/cluster</th>
<th>Before Resolution</th>
<th>After Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solved</td>
<td>Time</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.52 sec</td>
</tr>
<tr>
<td>31</td>
<td>100</td>
<td>0.71</td>
</tr>
<tr>
<td>32</td>
<td>100</td>
<td>1.03</td>
</tr>
<tr>
<td>33</td>
<td>100</td>
<td>1.54</td>
</tr>
<tr>
<td>34</td>
<td>100</td>
<td>3.44</td>
</tr>
<tr>
<td>35</td>
<td>100</td>
<td>6.38</td>
</tr>
<tr>
<td>36</td>
<td>90</td>
<td>21.7</td>
</tr>
<tr>
<td>37</td>
<td>41</td>
<td>35.5</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>28.4</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

# SLS and Local Consistency

<table>
<thead>
<tr>
<th>C</th>
<th>Solvable</th>
<th>Algorithm</th>
<th>Solved</th>
<th>Tries</th>
<th>Flips</th>
<th>PPC Time</th>
<th>Total Time</th>
<th>BJ-DVO</th>
</tr>
</thead>
<tbody>
<tr>
<td>265</td>
<td>88.5%</td>
<td>GSAT</td>
<td>139</td>
<td>336</td>
<td>147K</td>
<td>0 sec</td>
<td>36 sec</td>
<td>19 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PPC + GSAT</td>
<td>152</td>
<td>292</td>
<td>140K</td>
<td>8 sec</td>
<td>66 sec</td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>66%</td>
<td>GSAT</td>
<td>78</td>
<td>406</td>
<td>191K</td>
<td>0 sec</td>
<td>45 sec</td>
<td>33 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PPC + GSAT</td>
<td>83</td>
<td>381</td>
<td>195K</td>
<td>14 sec</td>
<td>92 sec</td>
<td></td>
</tr>
</tbody>
</table>

| N=30, K=64, T=2048/4096, 100 instances, MaxFlips = 128K, C\text{crit}=180? |
|---|---|---|---|---|---|---|---|---|
| 163 | PPC + GSAT | 56 | 276 | 59K | 56 sec | 153 sec | * | * |
|     | GSAT | 58 | 247 | 53K | 0 sec | 89 sec | * | * |

Table 2: Partial Path Consistency and GSAT

| Uniform random 3SAT, N=600, C=2550, 100 instances, MaxFlips = 512K |
|---|---|---|---|---|---|
| Algorithm | Solved | Tries | Flips | BR-3 Time | Total Time |
| GSAT | 36 | 63 | 176K | 0 sec | 15.3 sec |
| BR-3 + GSAT | 31 | 45 | 125K | 0.3 sec | 15.0 sec |

Table 3: Bound-3 Resolution and GSAT
SLS and Local Consistency

Summary:

- For structured problems, enforcing local consistency will improve SLS.
- For uniform CSPs, enforcing local consistency is not cost effective: performance of SLS is improved, but not enough to compensate for the preprocessing cost.
SLS and Cutset Conditioning

**Background:**
- Tree algorithm is tractable for trees.
- Networks with bounded width are tractable*.

**Basic Scheme:**
- Identify a cutset such that width is reduced to desired value.
- Use search with cutset conditioning.
Local search on Cycle-cutset

Tree variables $X$

$C_{min} = \min_{Y=y} C(y) = \min_{Y=y} \min_{X=x} \{C(X | Y = y)\}$
Tree Algorithm

Tree Algorithm: minimizing the cost of a tree-like subnetwork:
where $R_{z_i,z_j}(a_i,a_j)$ is the constraint between $z_i$ and $z_j$ and is either 0 if $(a_i,a_j) \in R_{z_i,z_j}$ or 1, otherwise.

**Input:** An arc consistent network $\mathcal{R} = (X,D,C)$. Variables $X$ partitioned into cycle cutset $Y$ and tree variables $Z$, $X = Z \cup Y$. An assignment $Y = \bar{y}$.

**Output:** An assignment $Z = \bar{z}$ that minimizes the number of violated constraints of the entire network when $Y = \bar{y}$.

**Initialization:** For any value $\bar{y}[i]$ of any cutset variable $y_i$, the cost $C_{y_i}(\bar{y}[i], \bar{y})$ is 0.

1. Going from leaves to root on the tree,
   
   (a) for every variable, $z_i$ and any value $a_i \in D_{z_i}$, compute,
   
   $$C_{z_i}(a_i, \bar{y}) = \sum_{\{z_j | z_j \text{ child of } z_i\}} \min_{a_j \in D_{z_j}} (C_{z_j}(a_j, \bar{y}) + R_{z_i,z_j}(a_i,a_j))$$

   (b) endfor
Tree Algorithm (contd)

2. Compute, going from root to leaves, new assignment for every tree variable $z_i$:

   (a) for a tree variable $z_i$, let $D_{z_i}$ be its consistent values with $v_{p_i}$ the value assigned to its parent $p_i$, compute

   $$a_i \leftarrow \arg\min_{a_i \in D_{z_i}} (C_{z_i}(a_i, \bar{y}) + R_{z_i,p_i}(a_i, v_{p_i}))$$

   (b) endfor

3. return $(<z_1,a_1>, ..., <z_k,a_k>)$. 
GSAT with Cycle-Cutset (Kask and Dechter, 1996)

**Input:** a CSP, a partition of the variables into cycle-cutset and tree variables

**Output:** an assignment to all the variables

**Within each try:**
Generate a random initial assignment, and then alternate between the two steps:

1. Run **Tree algorithm** (arc-consistency+assignment) on the problem with fixed values of cutset variables.
2. Run GSAT on the problem with fixed values of tree variables.
Theorem 7.1

Theorem 7.1  The Tree Algorithm in Figure 7.4 is guaranteed to find an assignment that minimizes the number of violated constraints in every tree-like subnetwork, conditioned on the cutset values.
Results GSAT with Cycle-Cutset
(Kask and Dechter, 1996)

GSAT versus GSAT+CC

# of problems solved vs cycle cutset size
SLS and Cutset Conditioning

Summary:

- A new combined algorithm of SLS and inference based on cutset conditioning
- Empirical evaluation on random CSPs
- SLS combined with the tree algorithm is superior to pure SLS when the cutset is small
Possible project

- Program variants of SLS+Inference
  - Use the computed cost on the tree to guide SLS on the cutset. This is applicable to optimization
  - Implement the idea for SAT using off-the-shelves code: unit-resolution from minisat, SLS from walksat.
  - More exciting: use our new code for optimization (TSLS) for consistency or satisfiability (See Milchgrub).

- Other projects: start thinking.