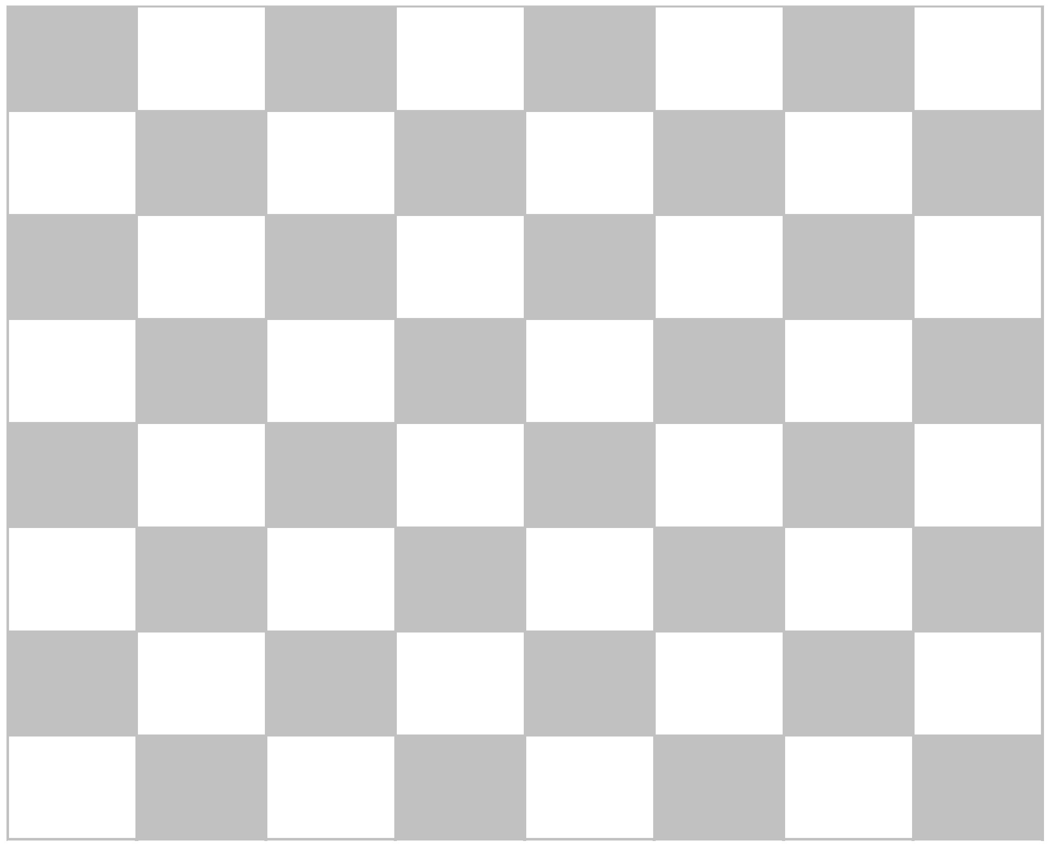
Stochastic greedy local search

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 clasuses. 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 MAX_TRIES. A cost function.

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

1. **for** i=1 to MAX_TRIES
   - **initialization: let** $\bar{a} = (a_1, ..., a_n)$ be a random initial assignment to all variables.
   - **repeat**
     (a) **if** $\bar{a}$ is consistent, return $\bar{a}$ 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 [illegible] in the cost of the assignment; pick a pair $< x_i, a'_i > \in Y$,
     $\bar{a} \leftarrow (a_1, ..., a_{i-1}, a'_i, a_{i+1}, ..., a_n)$ (just flip a_i to a'_i).
   - **until** the current assignment cannot be improved.
2. **endfor**
3. return **false**

Example: CNF

Example 7.1 Consider the formula $\varphi =$ [illegible] Assume that in the [illegible]". 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 $(\neg A \vee \neg B \vee C)$, 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.

$$F(\bar{a}) = \sum w_i C_i(\bar{a})$$

- 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) **start** with a random initial assignment $\bar{a}$.
   (b) **for** i=1 to MAX_FLIPS
   - **if** $\bar{a}$ is a solution, return **true** and $\bar{a}$.
   - **else,**
     i. **pick** a violated constraint C, randomly
     ii. **choose** with probability p a variable-value pair $< x, a' >$ for $x \in scope(C)$, or, [illegible] when the value of x is changed to a', (minus 1 if the current constraint is satisfied).
     iii. Change x's value to a'.
3. **endfor**
4. return **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 $(\neg B \vee \neg C)$, 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. □

$$(\neg C),(\neg A \vee \neg B \vee C)(\neg A \vee D \vee E)(\neg B \vee \neg C)$$

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 ½ 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

formula		GSAT									Simul. Ann.		
		basic			walk			noise					
vars	clauses	time	flips	R	time	flips	R	time	flips	R	time	flips	R
100	430	.4	7554	8.3	.2	2385	1.0	.6	9975	4.0	.6	4748	1.1
200	860	22	284693	143	4	27654	1.0	47	396534	6.7	21	106643	1.2
400	1700	122	2.6×10^6	67	7	59744	1.1	95	892048	6.3	75	552433	1.1
600	2550	1471	30×10^6	500	35	241651	1.0	929	7.8×10^6	20	427	2.7×10^6	3.3
800	3400	*	*	*	286	1.8×10^6	1.1	*	*	*	*	*	*
1000	4250	*	*	*	1095	5.8×10^6	1.2	*	*	*	*	*	*
2000	8480	*	*	*	3255	23×10^6	1.1	*	*	*	*	*	*

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 unsatised 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 di®er from WalkSAT in a subtle way. For each strategy the table gives the average time in seconds it took to ¯nd a satisfying assignment,[2] the average number of °ips it required, and *R*, the average number of restarts needed before ¯nding 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

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

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

Comparing DPLL against local search

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

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

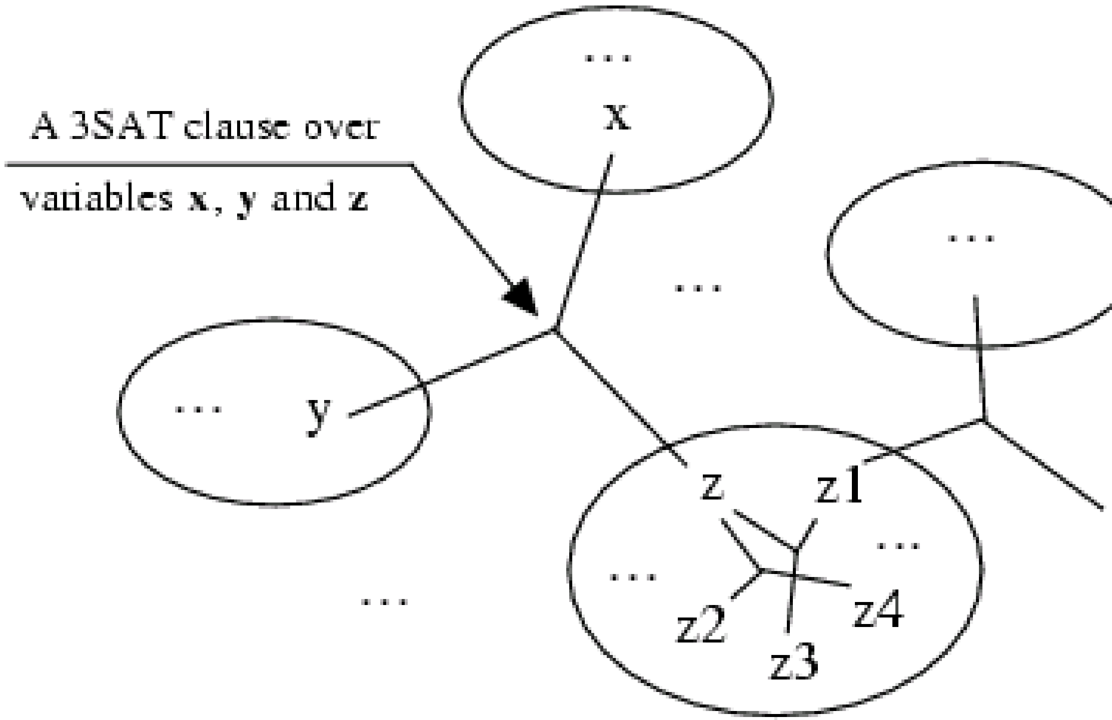
- **Structured** (hierarchical 3SAT cluster structures) vs. **(uniform) random**.

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



http://www.ics.uci.edu/%7Ecsp/r34-gsat-local-consistency.pdf

SLS and SLS on structured problems

Solvable 3SAT cluster structures, 100 instances, MaxFlips = 512K
5 variables per cluster, 50 clusters, 200 clauses between clusters
Restricted Bound-3 Resolution : only original clauses resolved
Running times, number of flips and clauses added are given as an average per problem solved

	Before Resolution			After Resolution					
C/cluster	Solved	Time	Flips	Solved	RBR-3 Time	Total Time	Flips	New Clauses	DP
30	100	0.52 sec	4.5K	100	3.6 sec	3.7 sec	189	1736	1.03 sec
31	100	0.71	5.1K	100	3.88	3.91	176	1731	1.04
32	100	1.03	8.4K	100	4.16	4.20	162	1722	1.09
33	100	1.54	12K	100	4.36	4.39	155	1708	1.11
34	100	3.44	26K	100	4.66	4.70	151	1690	1.15
35	100	6.38	49K	100	4.92	4.95	140	1668	1.18
36	90	21.7	161K	100	5.23	5.26	135	1640	1.19
37	41	35.5	252K	100	5.42	5.45	131	1609	1.23
38	3	28.4	202K	100	5.94	5.97	125	1574	1.27
39	0	-	-	100	5.95	5.98	121	1540	1.29
40	0	-	-	100	6.13	6.17	115	1503	1.29

Table 1: Bound-3 Resolution and GSAT

http://www.ics.uci.edu/%7Ecsp/r34-gsat-local-consistency.pdf

SLS and Local Consistency

N=100, K=8, T=32/64, 200 instances, MaxFlips = 512K								
C	Solvable	Algorithm	Solved	Tries	Flips	PPC Time	Total Time	BJ-DVO
265	88.5 %	GSAT	139	336	147K	0 sec	36 sec	19 min
		PPC + GSAT	152	292	140K	8 sec	66 sec	
270	66 %	GSAT	78	406	191K	0 sec	45 sec	33 min
		PPC + GSAT	83	381	195K	14 sec	92 sec	
N=30, K=64, T=2048/4096, 100 instances, MaxFlips = 128K, C_{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

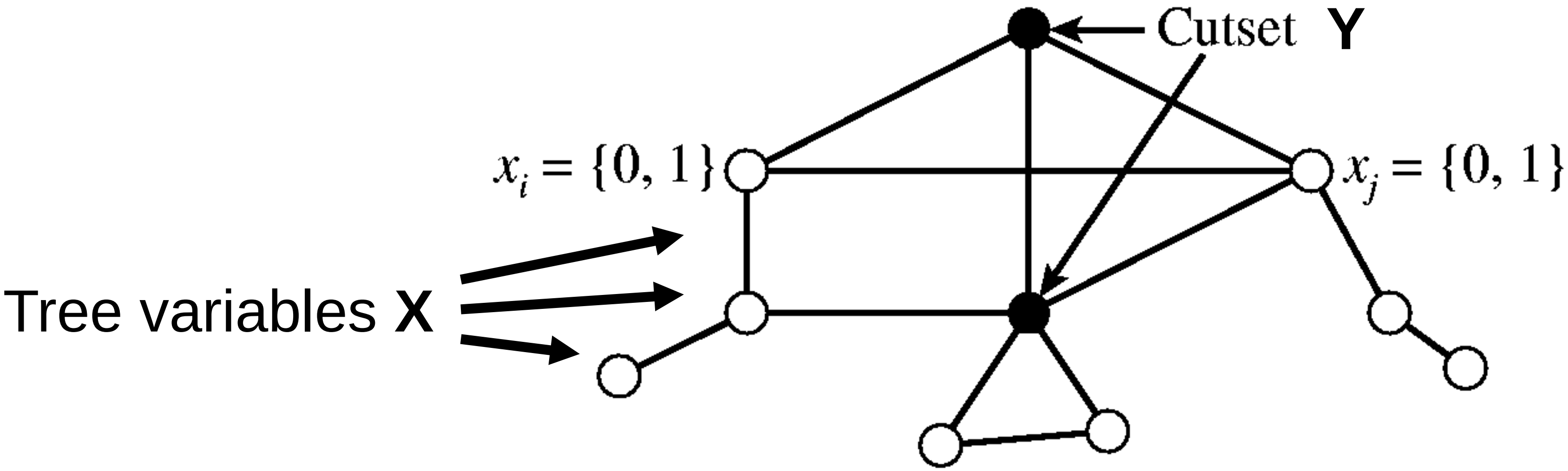
Background:

- 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



$$C_{min} = \min_{Y=y} C(y) = \min_{Y=y} \min_{X=x} \{C(X \mid Y = y)\}$$

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,

   $$C_{z_i}(a_i, \bar{y}) = \sum_{\{z_j | z_j \ child\ of\ z_i\}} min_{a_j \in D_{z_j}} \left(C_{z_j}(a_j, \bar{y}) + R_{z_i,z_j}(a_i, a_j)\right)$$

   (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 asignment,
and then alternate between the two steps:

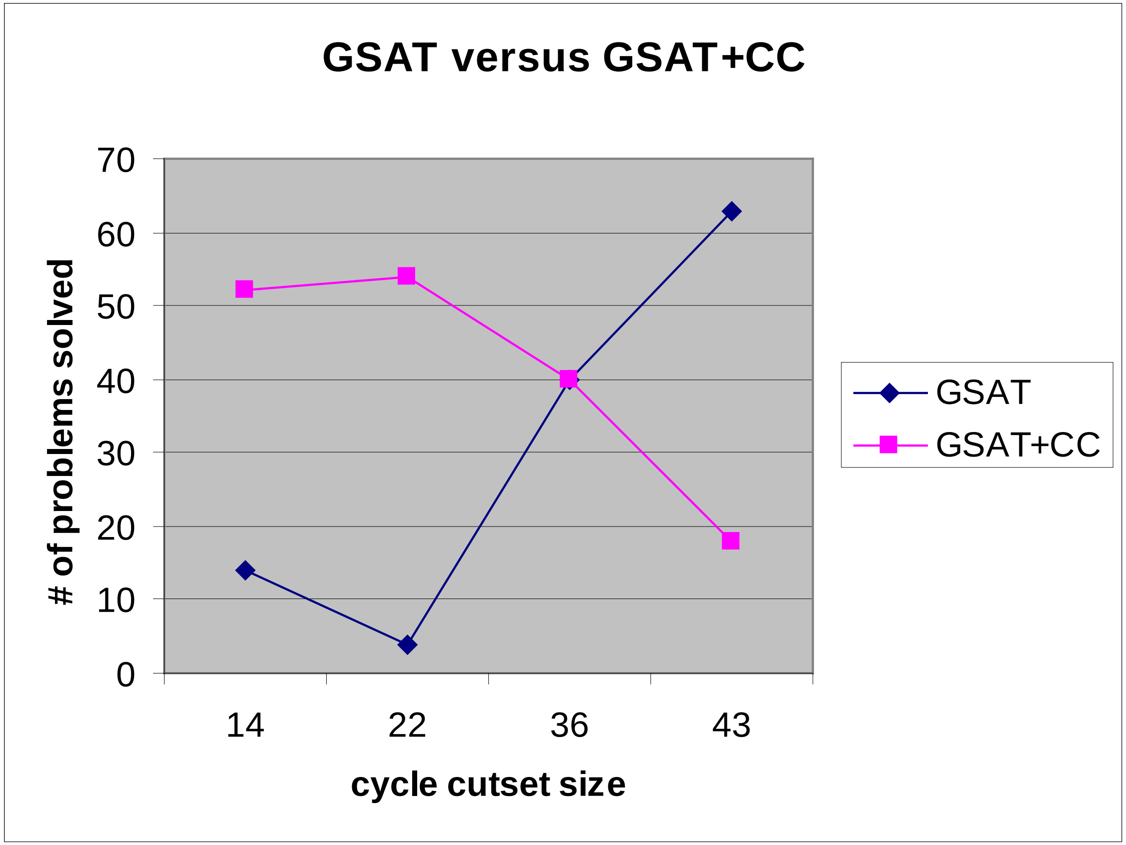
1. Run **Tree algorithm** (arc-consistency+assignment) on the problem with fixed values of cutset variables.
2. Run GSAT on the problem with fixed values of tree variables.

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)



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 ptimization (TSLS) for consistency or satisfiability (See Milchgrub).
- Other projects: start thinking.