Focused Random Walk with Configuration Checking and Break Minimum for Satisfiability

Chuan Luo[1] Shaowei Cai[2] Wei Wu[1,3] Kaile Su[3,4]

[1]Key Laboratory of High Confidence Software Technologies, Peking University
[2]Queensland Research Laboratory, National ICT Australia
[3]College of Mathematics, Physics and Information Engineering, Zhejiang Normal University
[4]Institute for Integrated and Intelligent Systems, Griffith University

September, 2013

Outline

Outline

The SAT Problem

- A set of Boolean variables: $X = \{x_1, x_2, ..., x_n\}$.
- Literals: $x_1, \neg x_1, x_2, \neg x_2, ..., x_n, \neg x_n$
- A set of Clauses: $x_1 \vee \neg x_2, x_2 \vee x_3, x_2 \vee \neg x_4, \neg x_1 \vee \neg x_3 \vee x_4, ...$
- A Conjunctive Normal Form (CNF) formula:

$$F = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$$

- The satisfiability problem (SAT): test whether all clauses in F can be satisfied by some consistent assignment of truth values to variables $X \rightarrow \{0, 1\}$.

Importance of SAT

- Theoretical importance
  - Its the first NP-Complete problem discovered by Cook in 1971
- Application everywhere
  - Very-large-scale integration
  - Bounded Model Checking
  - AI Planning
  - Theorem Proving
  - Software modeling and verification
  - ...

Solving SAT

- Complete methods for SAT: DPLL procedure [Davis and Putnam 1960; Davis, Logemann and Loveland 1962]
  - Branch and cut
  - Resolution
  - Learning and non-chronological backtracking, 1996
- Incomplete methods for SAT: Local search [Selman et al. 1994]
  - Modify the solution step by step
  - Unable to prove unsatisfiability, but may find solutions for a satisfying problem quickly
  - Better choice when the knowledge about the problem domain is rather limited

Local Search for SAT

- View the solution space as a set of points connected to each other.
- There is a cost function which needs to be minimized that can be computed for each point, i.e. number of unsatisfied clauses.
- Local search involves starting at some point in the solution space, and moving to neighboring points in an attempt to lower the cost function.

Local Search for SAT

Local search for SAT works as follows:

- starts with a truth assignment, which is usually randomly generated.
- iteratively flips the value of a variable, until it seeks out a satisfying assignment or time out.

The essential part: decide which variable to be flipped in each step.

Notations

Two variables are neighbors when they appear in at least one clause.
As x and y are variables, we define two notations as follows:
$N(x) = \{y \mid y \in V(F),\ y \text{ and } x \text{ are neighbors}\}$
$CL(x) = \{c \mid c \text{ is a clause which } x \text{ appears in}\}$

Common properties of a variable x:

- $make(x)$: the number of currently unsatisfied clauses that would become satisfied by flipping variable x.
- $break(x)$: the number of currently satisfied clauses that would become unsatisfied by flipping variable x (used in WalkSAT).
- $score(x)$: the increase in the total number of satisfied clauses by flipping variable x (can be understood as $make(x) - break(x)$).

Outline

Configuration Checking

Configuration checking (CC) techniques have proven successful in SLS algorithms for SAT [Cai and Su 2011; Luo, Su and Cai 2012].

The main idea of configuration checking is to forbid flipping any variable whose circumstance information has not been changed since its last flip.

In the context of SAT, the first definition of configuration was based on neighboring variables [Cai and Su 2011]. In this work, we adopt the alternative definition of configuration based on clause states [Luo, Su and Cai 2012].

Configuration Checking

Definition 1

Given a CNF formula F and a complete assignment α to $V(F)$, the *configuration* of a variable $x \in V(F)$ is a vector *configuration*(x) consisting of the states of all clauses in $CL(x)$ under assignment α.

Configuration Checking for SAT: when selecting a variable to flip, for a variable $x \in V(F)$, if the configuration of x has not changed since x's last flip, then it should not be flipped.

This strategy is reasonable in terms of avoiding cycles; otherwise, the algorithm is led to a scenario it has recently faced, which is likely to cause a cycle.

An Implementation of Configuration Checking for SAT

Employ an integer array *ConfTimes*.

- $ConfTimes[x] > 0$ means the configuration of x has changed since x's last flip;
- $ConfTimes[x] = 0$ on the contrary.

An Implementation of Configuration Checking for SAT

Employ an integer array *ConfTimes*.

- $ConfTimes[x] > 0$ means the configuration of x has changed since x's last flip;
- $ConfTimes[x] = 0$ on the contrary.

Maintain the *ConfTimes* array

- Rule 1: In the beginning, for each variable $x \in V(F)$, $ConfTimes(x)$ is set to 1.
- Rule 2: Whenever a variable x is flipped, $ConfTimes(x)$ is reset to 0. Then each clause $c \in CL(x)$ is checked whether its state is changed by flipping x. If this is the case, for each variable y ($y \neq x$) in c, $ConfTimes(y)$ is increased by 1.

Outline

The CCBM Heuristic

Definition 2

Given a CNF formula F and a complete assignment α to $V(F)$, a variable x is *configuration changed decreasing* (CCD) if and only if $score(x) > 0$ and $ConfTimes(x) > 0$.

This work utilizes *CCDVars(c)* to denote the set of all CCD variables in clause c.

Definition 3

Given a CNF formula F and a complete assignment α to $V(F)$, for each clause c, a variable x is the *break minimum* (BM) variable of clause c if and only if x appears in clause c and $break(x) = \min\{break(y) \mid y \text{ appears in } c\}$.

This work utilizes *BMVars(c)* to denote the set of all BM variables of clause c.

The CCBM Heuristic

The CCBM heuristic works as follows:

- first select an unsatisfied clause c randomly;
- If *CCDVars(c) is not empty*,
  - select the variable with the greatest *score* in *CCDVars(c)*;
- Otherwise,
  - With probability p, select the variable with the greatest *ConfTimes* in *BMVars(c)*;
  - With probability $1 - p$, select the variable with the greatest *ConfTimes* in clause c;

The FrwCB Algorithm

Algorithm 1: FrwCB

Input: CNF-formula F, *maxSteps*
Output: A satisfiable assignment α of F or *Unknown*
begin
  generate a random assignment α;
  initialize *ConfTimes*(x) as 1 for each variable x;
  for *step* $\leftarrow$ *1* **to** *maxSteps* **do**
    if α *satisfies* F **then return** α;
    $c \leftarrow$ an unsatisfied clause chosen randomly;
    if *CCDVars(c) is not empty* **then**
      $v \leftarrow x$ with the greatest *score*(x) in *CCDVars(c)*, breaking ties by preferring the one with the greatest *ConfTimes*(x);
    else if *with the fixed* **probability** p **then**
      $v \leftarrow x$ with the greatest *ConfTimes*(x) in *BMVars(c)*, breaking ties by preferring the least recently flipped one;
    else
      $v \leftarrow x$ with the greatest *ConfTimes*(x) in clause c, breaking ties by preferring the least recently flipped one;
    flip v and update *ConfTimes*;
  return *Unknown*;
end

Discussion on Differences between Swqcc and FrwCB

The most related work is the Swqcc algorithm [Luo, Su and Cai 2012], which adopts a clause states based configuration checking heuristic named QCC.

The most important difference is that Swqcc and FrwCB adopt different local search paradigms. While Swqcc is a two-mode (GSAT-like + random walk) SLS algorithm, FrwCB is a single-mode (focused random walk) one.

Discussion on Differences between Swqcc and FrwCB

The two algorithms employ different heuristics to pick a variable to flip.

- **Swqcc employs the QCC heuristic:** if there exist candidate variables (described in [Luo, Su and Cai 2012]) for the greedy mode, QCC selects the one with greatest *score*; otherwise QCC first selects an unsatisfied clause c randomly, and then always picks a variable with the greatest *ConfTimes* in c.
- **FrwCB uses the CCBM heuristic:** after selecting an unsatisfied clause c randomly, if there exist candidate variables (i.e., CCD variables) in c, CCBM selects the one with greatest *score*; otherwise CCBM picks either a variable with the greatest *ConfTimes* in c or a variable with the minimum *break* in c.

The experimental analysis will be presented as follows.

Outline

Experiments on Large Random 3-SAT from SC 2011

Instance Class	WalkSAT		probSAT		CCASat		Swqcc		FrwCB	
	#suc	time	#suc	time	#suc	time	#suc	time	#suc	time
3SAT-v2500	95	152	99	88	100	9	100	**6**	100	37
3SAT-v5000	100	31	100	13	100	**11**	100	13	100	12
3SAT-v10000	100	19	100	21	100	19	100	37	100	**10**
3SAT-v15000	100	24	100	24	100	29	100	73	100	**13**
3SAT-v20000	100	35	100	37	100	44	100	118	100	**26**
3SAT-v25000	100	54	100	56	100	73	100	172	100	**36**
3SAT-v30000	100	56	100	63	100	92	100	186	100	**42**
3SAT-v35000	100	122	100	108	100	147	100	279	100	**61**
3SAT-v40000	100	114	100	84	100	125	100	240	100	**56**
3SAT-v50000	99	206	100	145	100	250	99	403	100	**99**

Table Comparative results on large random 3-SAT instances from SAT Competition 2011. Each solver is performed 100 times on each instance class.

Experiments on All Random 3-SAT from SC 2012

# Total Runs	WalkSAT		probSAT		CCASat		Swqcc		FrwCB	
	#suc	time	#suc	time	#suc	time	#suc	time	#suc	time
1200	964	658	1003	598	967	693	986	731	**1043**	**499**

Table Comparative results on all the random 3-SAT instances from the SAT Challenge 2012.

Experiments on Huge Random 3-SAT

Instance Class ($1.0M = 10^6$)	WalkSAT		probSAT		CCASat		FrwCB	
	#suc	avg time	#suc	avg time	#suc	avg time	#suc	avg time
3SAT-v0.1M	20	375	20	266	20	955	20	**227**
3SAT-v0.3M	20	920	20	934	10	9064	20	**393**
3SAT-v0.5M	20	2150	20	1905	0	>10000	20	**789**
3SAT-v1.0M	20	4691	20	4358	0	>10000	20	**1865**
3SAT-v1.5M	20	7696	20	6838	0	>10000	20	**3248**
3SAT-v2.0M	3	9964	15	9360	0	>10000	**20**	**4197**
3SAT-v2.5M	0	>10000	0	>10000	0	>10000	**20**	**5045**
3SAT-v3.0M	0	>10000	0	>10000	0	>10000	**20**	**6463**
3SAT-v3.5M	0	>10000	0	>10000	0	>10000	**20**	**7797**
3SAT-v4.0M	0	>10000	0	>10000	0	>10000	**15**	**9530**

Table Comparative results on huge random 3-SAT instances. Each solver is performed 20 times for each instance class. **Swqcc fails to solve any instance in this benchmark, so we do not report its results.**

Experiments on Random 5-SAT and Random 7-SAT

Instance Class	WalkSAT		probSAT		CCASat		Swqcc		FrwCB	
	#suc	time	#suc	time	#suc	time	#suc	time	#suc	time
5SAT-v500	250	17.7	250	9.0	250	**7.0**	250	37.8	250	13.1
7SAT-v90	250	28.7	250	37.4	250	43.2	250	**14.4**	250	25.8

Table Comparative results on 5-SAT and 7-SAT instances. The number of total runs on each instance class is 250.

Experiments on Structured SAT

Instance Class	#inst.	WalkSAT #suc avg time	probSAT #suc avg time	CCASat #suc avg time	Swqcc #suc avg time	Sattime #suc avg time	FrwCB #suc avg time
ais	4	40 0.3	40 2.6	40 <0.1	40 <0.1	40 0.3	40 0.3
blocksworld	7	70 2.8	70 1.2	70 0.6	63 214.7	70 0.5	70 1.4
gcp	4	40 2.7	21 984.6	40 61.5	32 483.5	40 2.2	40 1.3
jnh	16	160 0.11	160 <0.1	160 <0.1	160 <0.1	160 <0.1	160 <0.1
logistics	4	40 0.3	40 0.1	40 <0.1	40 <0.1	40 <0.1	40 0.2
par8	5	50 8.0	50 23.5	50 16.8	50 4.6	50 <0.1	50 2.1
par8-c	5	50 <0.1	50 <0.1	50 <0.1	50 <0.1	50 <0.1	50 <0.1
par16	5	0 >2000	0 >2000	0 >2000	0 >2000	50 23.7	0 >2000
par16-c	5	6 1898.4	50 264.9	45 310.9	50 161.6	50 16.0	50 99.4
frb50-23	5	42 553.6	38 765.0	40 571.6	39 744.2	49 248.7	47 332.8
frb53-24	5	35 880.4	19 1598.2	34 990.6	20 1550.5	47 539.7	46 624.0
frb56-25	5	34 992.6	24 1364.3	30 1137.8	15 1555.8	43 656.7	40 739.1
frb59-26	5	17 1555.5	10 1697.8	15 1627.6	11 1771.6	34 1094.9	27 1317.2

Table Comparative performance results on the structured instances. Each solver is performed 10 runs for each instance

Experiments of SP+SLS

# Total Runs	SP+WalkSAT		SP+probSAT		SP+CCASat		SP+Swqcc		SP+FrwCB	
	#suc	a.t.sls	#suc	a.t.sls	#suc	a.t.sls	#succ	a.t.sls	#suc	a.t.sls
20	20	810.5	20	685.2	0	>2000	0	>2000	20	**251.7**

Table Comparative results of SP+SLS on random 3-SAT instances with 10^7 variables. 'a.t.sls' means the averaged time of the SLS component in SP+SLS hybrid solver.

Outline

Conclusion

- This work proposes an effective heuristic named CCBM which combines two strategies namely configuration checking and break minimum to improve FRW algorithms.
- This work utilizes the CCBM heuristic to develop a new SLS algorithm called FrwCB.
- FrwCB significantly outperforms state-of-the-art solvers including WalkSAT, probSAT, CCASat and Swqcc on huge random 3-SAT instances with up to 4 million variables.
- FrwCB cooperates well with SP on random 3-SAT instances with 10 million variables.
- The robustness of FrwCB is confirmed by the experimental results on random 5-SAT and 7-SAT instances as well as structured SAT instances.

Outline

References

1 Chuan Luo, Shaowei Cai, Wei Wu, Kaile Su: Focused Random Walk with Configuration Checking and Break Minimum for Satisfiability. In Proc. of CP 2013, pp. 481-496 (2013). [This Work, CCBM for SAT]

2 Chuan Luo, Kaile Su, Shaowei Cai: Improving Local Search for Random 3-SAT Using Quantitative Configuration Checking. In Proc. of ECAI 2012, pp. 570-575 (2012). [QCC for SAT]

3 Shaowei Cai, Kaile Su, Abdul Sattar: Local Search with Edge Weighting and Configuration Checking Heuristics for Minimum Vertex Cover. Artificial Intelligence 175(9-10), 1672-1696 (2011). [Proposing the Configuration Checking Heuristic, CC for MVC]

4 Shaowei Cai, Kaile Su: Local Search with Configuration Checking for SAT. In Proc. of ICTAI 2011, pp. 59-66 (2011). [CC for SAT]

The End

Thank you.