Quick start guide for SATenstein-LS

Ashiqur R. KhudaBukhsh
University of British Columbia
Department of Computer Science

February 26, 2009

1 Introduction

`SATenstein-LS`is a highly-parameterized (41 parameters in total) generalized local search algorithm that is built by combining components from existing high-performance SLS (Stochastic Local Search) algorithm. `SATenstein-LS`can be configured to instantiate a broad range of existing high-performance SLS-based SAT solvers, and also billions of novel algorithms. `SATenstein-LS`is built on top of UBCSAT [Tompkins and Hoos, 2004] framework and it is highly recommended to go through the quick start guide `http://www.satlib.org/ubcsat/quickstart.html` of UBCSAT to gain more familiarity with `SATenstein-LS`.

2 Configuration

The source code for UBCSAT with `SATenstein-LS`implemented in it is available at /ubc/cs/project/arrow/projects/FORLIN/SATenstein-LS/ubcsat/. In the same folder you can find the executable (ubcsat). UBCSAT always requires two key command-line parameters:

- an algorithm to use, specified with the [-alg] parameter, and
- a SAT instance file to solve, specified with [-i] parameter

To run `SATenstein-LS`, use `satenstein` as the algorithm. So, for an instance 'sample.cnf' the simplest execution command for `SATenstein-LS`is

```
> ubcsat -alg satenstein -i sample.cnf
```

Once you execute this command, in the output, along with the run result (e.g. number of flips, runtime), you will notice a list of parameters. The execution command you just performed, ran `SATenstein-LS`will all the parameters set to their default values. You can find the default value of each parameter mentioned along with the parameter name. Out of these 44 listed parameters, it is recommended not to change the values for *varinfalse*, *sapsthresh* and *rwpwalk*. The remaining 41 parameters are used to control the behavior of `SATenstein-LS`. All our tuning experiments were performed on these 41 parameters. Similar to UBCSAT, the values for these parameters can be specified by [-*paramname* value]. For example, the execution command for running `SATenstein-LS`with parameter *adaptive* set to 0 will be

```
> ubcsat -alg satenstein -i sample.cnf -adaptive 0
```

Section 3 presents detailed description of all the parameters.

3 Configurable Parameters

Table 1 and 2 presents the categorical and integer parameters of `SATenstein-LS`respectively along with a short description of each parameter. In the 'active when' column unless otherwise mentioned for more than one parameters the conditional operator is AND.

Parameter	Active When	Domain	Description
performrandomwalk	Base level parameter	{0,1}	If true, module 1 is performed
promisinglist	Base level parameter	{0,1}	If true, module 2 is performed
singleclause	promisinglist in {0}	{0,1}	If true, module 3 is performed else module 4
decreasingvariable	promisinglist in {1}	{1,2,3,4,5,6,7,8,9,10,11}	Specifies choice for selecting variable from promising list (see Table 4)
heuristic	promisinglist in {1} or singleclause in {1}	{1,2,3,4,5,6,7,8,9,10,11,12,13}	Specifies the heuristic (see Table 3)
adaptive	Base level parameter	{0,1}	When true, performs the adaptive versions of the available heuristic
adaptivenoisescheme	adaptive in {1} promisinglist in {1}	{1,2}	Specifies adaptive noise mechanisms.
adaptwalkprob	adaptive in {1}	{0,1}	If true, walk probability or diversification probability of a heuristic is adaptively tuned.
tabusearch	Base level parameter	{0,1}	If true, tabu variables are not chosen for flipping.
clausepen	Base level parameter	{0,1}	If true, clause penalties are computed.
selectclause	{7}: clausepen in {1}	{1,7}	1 selects an unsat clause uniformly at random. 7 selects an unsat clause with a probability proportional to its clause penalty.
randomwalk	performrandomwalk in {1}	{1,3,4,5}	1 randomly selects a variable from an unsat clause. 3 selects the least recently flipped variable from an unsat clause. 4 selects the least-number of times flipped variable from an unsat clause. 5 selects the variable with least VW2 weight from an unsat clause.
adaptiveprom	promisinglist in {1}	{0,1}	If true, performs adaptive versions of Novelty variants to select variable from promising list.
adaptpromwalkprob	promisinglist in {1} adaptiveprom in {1}	{0,1}	If true, walk probability or diversification probability of Novelty variants used on promising list is adaptively tuned.
scoringmeasure	promisinglist in {0} singleclause in {0}	{1,2,3}	Specifies the scoring measure. 1 uses makecount - breakcount 2 uses makecount 3 uses -breakcount
tiebreaking	promisinglist in {1} decreasingvariable in {1,4,5}	{1,2,3,4}	1 breaks tie randomly. 2 breaks tie in favor of the least recently flipped variable. 3 breaks tie in favor of least number of times flipped variable. 4 breaks tie in favor of variable with least VW2 score.
updateschemepromlist	promisinglist in {1}	{1,2,3}	1 and 2 follow G^2WSAT 3 follows `gNovelty+`
smoothingscheme	clausepen in {1}	{1,2}	When singleclause is in {1} : 1 performs smoothing for only random 3SAT instances. with a fixed smoothing probabiblity 0.4. 2 provides smoothing for all instances. When singleclause is in {0} : 1 performs SAPS like multiplicative smoothing. 2 performs PAWS like additive smoothing.

Table 1: *Categorical parameters of SATenstein-LS.*

Parameter	Active When	Domain considered	Description
tabu	tabusearch in {1}	{1, 3, 5, 7, 10, 15, 20}	Specifies tabu step-length.

phi	adaptive in {1}
	singleclause in {1} or
	adaptive in {1}
	promisinglist in {1}
		{3,4, 5,6,7,8,9,10}	Parameter for adaptively setting noise.

theta	adaptive in {1}
	singleclause in {1} or
	adaptive in {1}
	promisinglist in {1}
		{3,4, 5,6,7,8,9,10}	Parameter for adaptively setting noise.

promphi	promisinglist in {1}
	adaptiveprom in {1}
	decreasingvariable in {7,8,9,10,11}
		{3,4, 5,6,7,8,9,10}	Parameter for adaptively setting noise.

promtheta	promisinglist in {1}
	adaptiveprom in {1}
	decreasingvariable in {7,8,9,10,11}
		{3,4, 5,6,7,8,9,10}	Parameter for adaptively setting noise.

maxinc	singleclause in {0}
	promisinglist in {0}
	clausepen in {1}
	smoothingscheme in {2}
		{5,10,15,20}	Parameter for PAWS.

Table 2: *Integer parameters of SATenstein-LS and the values considered during ParamILS tuning.*

Parameter Value	Heuristic	Related Parameters
1	Novelty [Mcallester et al., 1997]	novnoise
2	Novelty+ [Hoos, 2002]	novnoise, wp
3	Novelty++ [Li and Huang, 2005]	novnoise, dp
4	Novelty++‘ [Li et al., 2007]	novnoise, dp
5	R-Novelty [Mcallester et al., 1997]	novnoise
6	R-Novelty+ [Hoos, 2002]	novnoise, wp
7	VW1 [Prestwich, 2005]	wpwalk
8	VW2 [Prestwich, 2005]	s, c, wp
9	WalkSAT-SKC [Selman et al., 1994]	wpwalk
10	Noveltyp [Li et al., 2007]	novnoise
11	Novelty+p [Li et al., 2007]	novnoise, wp
12	Novelty++p [Li et al., 2007]	novnoise, dp
13	Novelty++'p [Li et al., 2007]	novnoise, dp

Table 3: *List of heuristics chosen by the parameter* heuristic *and related parameters.*

4 Example Parameter Configurations

Lets see how can we configure some known algorithms with `SATenstein-LS`. Here we give brief outline for the algorithm. For detailed description of the algorithms please refer to the cited papers.

4.1 Novelty+

Lets start with a simple algorithm `Novelty+` [Hoos, 1999]. This algorithm uses module 3 `(-singleclause 1 -promisinglist 0)` and does not use promising list. Table 3 shows that Novelty+ has two heuristic specific parameters *novnoise* and *wp*, and they have default values of 0.5 and 0.01 respectively. Table also shows that *heuristic* will be 2. `Novelty+` does not use any adaptive mechanism `(-adaptive 0)`.

Hence, the command to configure `SATenstein-LS`as `Novelty+` will be

```
 > ubcsat -alg satenstein -i sample.cnf -selectclause 1 -promisinglist 0
-adaptive 0 -heuristic 2 -wp 0.01 -novnoise 0.5
```

You may want to have a look at the default values of these parameters and you will find that the default values for *selectclause*, *novnoise* and *wp* are the same as we used to run `Novelty+`. Hence, this command can be simplified as

```
 > ubcsat -alg satenstein -i sample.cnf -promisinglist 0
-adaptive 0 -heuristic 2
```

Simple, isn't it? You may want to run `adaptNovelty+` [Hoos, 2002]. For that, all that you need to do is to change the value of *adaptive*.

The command to configure `SATenstein-LS`as `adaptNovelty+` is:

```
 > ubcsat -alg satenstein -i sample.cnf -promisinglist 0

 -adaptive 1 -heuristic 2
```

Parameter Value	Selection strategy
1	Checks for freebie (breakcount 0) variables in the promising decreasing variable stack. If there are freebies then selects among them according to the tie-breaking scheme. If there is not any freebie randomly picks one variable from the promising decreasing variable list
2	Select the variable with highest score
3	Select the least recently flipped variable
4	Select the least number of times flipped variable
5	Select the variable with least VW2 score
6	Randomly select a variable for flipping
7	Use Novelty
8	Use Novelty++
9	Use Novelty+
10	Use Novelty++'
11	Use Noveltyp

Table 4: *Strategies to select a variable from the promising list depending on the the value of parameter decreasingvariable.*

4.2 gNovelty+

gNovelty+ [Pham and Gretton, 2007] uses module 2 `(-promisinglist 1)`, module 1 `(-performrandomwalk 1)`, uses clause-penalties `(-clausepen 1)` and smoothing `(-smoothingscheme 1)` , adaptive mechanism `(-adaptive 1 -adaptwalkprob 0)`. It picks the variable with highest score from a non-empty promising list `(-decreasingvariable 2)` and if the list is empty it performs Novelty as heuristic (with flat moves) `(-heuristic 1 -performalternatenovelty 1)`. In module 1, gNovelty+ randomly picks a variable `(-randomwalk 1)` from a randomly selected false clause `( -selectclause`

`1)` with a probability 0.01 `(-rwp 0.01)`.

The command to configure `SATenstein-LS`as `gNovelty+` will be.

```
 > ubcsat -alg satenstein -inst sample.cnf -selectclause 1
-performrandomwalk 1 -randomwalk 1 -rwp 0.01 -promisinglist 1
-decreasingvariable 2 -updateschemepromlist 3 -adaptive 1
-adaptwalkprob 0 -adaptivenoisecheme 1 -heuristic 1 -smoothingscheme 1
-clausepen 1 -smoothingscheme 1
```

In a similar way, we can simplify this command by eliminating parameters whose values are same their default values.

4.3 SAPS

`SAPS` [Hutter *et al.*, 2002] uses module 4 `(-promisinglist 0 -singleclause 0)`, does not use any adaptive mechanism `(-adaptive 0)`. Ties are broken randomly `(-tiebreaking 1)` and (makecount - breakcount) is used to score a variable `(-scoringmeasure 1)`. It uses clause penalties `(-clausepen 1)` and multiplicative smoothing scheme `(-smoothingscheme 1)`.

Clearly, the command for running `SAPS` with `SAPS` specific parameters *alpha*, *rho* and *wp* to its default values will be

```
> ubcsat -alg satenstein -i sample.cnf -singleclause 0 -promisinglist 0
  -scoringmeasure 1 -tiebreaking 1 -clausepen 1 -smoothingscheme 1
  -adaptive 0
```

Since `RSAPS` [Hutter *et al.*, 2002] is `SAPS` with reactive mechanism, by toggling the value of *adaptive* in the above command we can configure `RSAPS`.

`PAWS` uses exactly the same configuration as `SAPS` except the smoothing scheme. By changing the value of *smoothingscheme* to 2 in the above command we can configure `PAWS`. The two `PAWS` specific parameters *maxinc* and *pflat* have default values of 10 and 0.15 respectively.

5 Results for different distributions

In our experiments, we found these following `SATenstein-LS`configurations to perform *best* for a given distribution.

5.1 HGEN

```
Best parameter configuration:
adaptwalkprob=0, theta=3, singleclause=1, randomwalk=3,
performrandomwalk=1, selectclause=1, promisinglist=0,
adaptiveprom=0, heuristic=7, adaptive=1, phi=10,
tabusearch=0, rdp=0.15
```

5.2 CBMC(SE)

```
Best parameter configuration:
rwp=0.01, singleclause=0, randomwalk=1, alpha=1.126,
performrandomwalk=1, smoothingscheme=1, rho=1, clausepen=1,
selectclause=1, promisinglist=0, adaptive=1, tiebreaking=1,
tabusearch=0, scoringmeasure=1, wp = 0.01
```

5.3 QCP

```
Best parameter configuration:
singleclause=1, randomwalk=4, performrandomwalk=1,
heuristic=5, clausepen=0, selectclause=1, promisinglist=0,
adaptive=0, rfp=0.07, tabusearch=0, scoringmeasure=1
```

5.4 SW-GCP

```
Best parameter configuration:
singleclause=1, randomwalk=3, performrandomwalk=1,
heuristic=1, clausepen=0, selectclause=1, promisinglist=0,
adaptive=0, tabusearch=0, rdp=0.01, novnoise=0.1
```

5.5 R3SAT

```
Best parameter configuration:
singleclause=0, alpha=1.126, wp=0.04, smoothingscheme=1,
rho=0.17, clausepen=1, selectclause=1, promisinglist=0,
adaptive=1, tiebreaking=2, tabusearch=0, scoringmeasure=1
```

5.6 FAC

```
Best parameter configuration:
adaptive=0, alpha=1.126, clausepen=1, performrandomwalk=0,
ps = 0.033, scoringmeasure=3, selectclause=1, singleclause=0,
smoothingscheme=1, tabusearch=0, tiebreaking=1, wp=0
```

References

[Hoos, 1999] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *in Proc. AAAI-99*, pages 661–666. MIT Press, 1999.

[Hoos, 2002] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. of AAAI-02*, pages 655–660, 2002.

[Hutter *et al.*, 2002] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, pages 233–248, 2002.

[Li and Huang, 2005] C. M. Li and W. Huang. Diversification and determinism in local search for satisfiability. In *Proc. of SAT-05*, pages 158–172, 2005.

[Li *et al.*, 2007] C. M. Li, W. Wei, and H. Zhang. Combining adaptive noise and promising decreasing variables in local search for SAT. Solver description, SAT competition 2007. `http://www.satcompetition.org/2007/adaptG2WSAT.pdf`, 2007.

[Mcallester *et al.*, 1997] David Mcallester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *In Proceedings of AAAI-97*, pages 321–326, 1997.

[Pham and Gretton, 2007] D. N. Pham and C. Gretton. gNovelty+. Solver description, SAT competition 2007. `http://www.satcompetition.org/2007/gNovelty+.pdf`, 2007.

[Prestwich, 2005] S. Prestwich. Random walk with continuously smoothed variable weights. In *Proc. of SAT-05*, pages 203–215, 2005.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. pages 337–343. MIT press, 1994.

[Tompkins and Hoos, 2004] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In *Proc. of SAT-04*, 2004.