sgen1: A generator for small but difficult satisfiability instances

Ivor Spence
School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast
i.spence@qub.ac.uk

March 2009

1 Introduction

This is a development of the previous generator sgen and the new version generates both satisfiable and unsatisfiable instances of the satisfiability problem using the DIMACS standard format.

It is invoked with the command

```
sgen1 sat|unsat -n num-of-variables [-s random-seed] [-m model-file]
```

and writes the generated instance to standard output. In all cases the variables in the generated instance are partitioned into small groups and for each group there are clauses to define relationships among the group. This is repeated for two or more paritions.

2 Unsatisfiable instances

The technique here is the same as the one used on the previous sgen generator. The required number of variables is first increased until it is of the form $4g + 1$ where $g \in \mathbb{N}$ - thus the generated instance will contain at least the requested number of variables, but may contain up to three more.

If the number of variables is 4g+1 then the number of clauses is 8g + 12. Each clause has three literals, so if the number of variables is n, then as n increases the number of clauses is approximately 2n and the number of literals is approximately 6n.

Generating small yet difficult unsatisfiable instances requires a balance between the following constraints:

- To keep the instance short, each clause should eliminate a large number of possibilities.

- To make the instance hard to solve, the variables in each clause should not be "related", that is occur together in other clauses.

Unfortunately, these conflict. Having unrelated variables tends to preclude a clause eliminating a large number of possible assignments. In the spirit of Hirsch's hgen8 program the compromise used here is to group the variables into sets of size four (in two different groupings) and have multiple clauses re-use the variables in each group.

Partition the variables into $(g-1)$ groups of size four and one of size five. For each group of size four, generate all possible 3-clauses of positive literals, i.e.

$$(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee c \vee d) \wedge (b \vee c \vee d)$$

This permits at most two variables from the set $\{a, b, c, d\}$ to be `false`. For the group of size five again generate all possible 3-clauses of positive literals (10 clauses), meaning that again only two variable from this group can be `false`. In total therefore, only $2(g-1)+2 = 2g = (n-1)/2$ variables can be `false`.

Now partition the variables into a different collection of $(g-1)$ groups of size four and one of size five and again for each group of variables generate all possible 3-clauses except this time use all negative literals. Thus now only $(n-1)/2$ variables can be `true`. Taking these two sets of clauses together it can be seen that it is not possible to assign a value to every variable since at most $(n-1)/2$ can be `true` and also at most $(n-1)/2$ can be `false`. Thus the generated instance is unsatisfiable.

3 Satisfiable instances

To generate difficult satisfiable instances force the number of variables to be a multiple of five and then partition into g groups of size five. The instance will contain $10g$ binary clauses and $2g$ 5-clauses, giving a total of $12g = 12n/5$ clauses and $6n$ literals.

For each group, generate all possible binary clauses of negative literals (10 clauses for each group). This permits at most one `true` variable per group, that is a maximum of $g = n/5$ `true` variables overall.

Repartition into groups of five, and for each group generate one 5-clause of all the positive literals. The $n/5$ `true` variable might be enough to satisfy these clauses if they can be allocated as one per group. Repeat this with another partition. Again, it might be possible to allocate one `true` variable per group.

If more and more collections of 5-clauses of positive literals are added it is less and less likely that the formula will remain satisfiable. Empirical results indicate that two collections give the most difficult instances for their size.

4 Partitioning

For both satisfiable and unsatisfiable cases, to create difficult instances we need to ensure that there is as little connection as possible between the different parti-

tions. For the first partition natural ordering is used, e.g. $\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \ldots$. Second and subsequent partitions are obtained by using simulated annealing, with a weight function which tries to minimise the extent to which the original partition is reflected in the second one. To ensure that satisfiable instances are created when requested, an initial choice of the $n/5$ `true` variables is made and each group in the partition is 'seeded' with one of these to guarantee that a satisfying model can be found. If the option -m model-file is chosen when generating a satisfiable instance, a satisfying model will be written to model-file.

Different partitions can be forced by using -s to specify the seed for the random number generation. If -s is omitted a value of 1 is used.

5 Results

Table 1 gives an example of a 9-variable unsatisfiable formula and a 10-variable satisfiable one. Unsatisfiable formulae of this kind were used as benchmarks in the SAT2007 competition and one with 732 literals (117 variables) was the shortest benchmark not be solved during the competition. The satisfiable formulae are more recent and no independent results are yet available but it appears that even the best solvers fail with benchmarks of size ≈ 1500 literals. Both unsatisfiable and satisfiable formulae have been submitted to the SAT2009 competition.

Unsatisfiable (9 variables)	Satisfiable (10 variables)
p cnf 9 28	p cnf 10 24
-2 -3 -4 0	-1 -2 0
-1 -3 -4 0	-1 -3 0
-1 -2 -4 0	-1 -4 0
-1 -2 -3 0	-1 -5 0
-5 -6 -7 0	-2 -3 0
-5 -6 -8 0	-2 -4 0
-5 -6 -9 0	-2 -5 0
-5 -7 -8 0	-3 -4 0
-5 -7 -9 0	-3 -5 0
-5 -8 -9 0	-4 -5 0
-6 -7 -8 0	-6 -7 0
-6 -7 -9 0	-6 -8 0
-6 -8 -9 0	-6 -9 0
-7 -8 -9 0	-6 -10 0
4 2 6 0	-7 -8 0
7 2 6 0	-7 -9 0
7 4 6 0	-7 -10 0
7 4 2 0	-8 -9 0
3 8 5 0	-8 -10 0
3 8 9 0	-9 -10 0
3 8 1 0	9 6 3 5 1 0
3 5 9 0	4 2 8 10 7 0
3 5 1 0	7 5 3 9 2 0
3 9 1 0	10 4 8 1 6 0
8 5 9 0	
8 5 1 0	
8 9 1 0	
5 9 1 0	

Table 1: Example instances