`sgen`: A generator for small but difficult satisfiability instances

Ivor Spence[*]
`i.spence@qub.ac.uk`

December 2006

1 Introduction

The generator takes two arguments and generates an unsatisfiable instance of the satisfiability problem. The first argument indicates the required number of variables (which is reduced until it is of the form $4g + 1$ where $g \in \mathbb{N}_1$) and the second is used as a seed for a random number generator. 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 instances requires a balance among the following constraints:

1. The instance should be unsatisfiable to prevent a "lucky" discovery of a satisfying assignment.
2. To keep the instance short, each clause should eliminate a large number of possibilities.
3. To make the instance hard to solve, the variables in each clause should not be "related", that is occur together in other clauses.

Unfortunately, 2 and 3 are in 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.

2 Satisfiable instances

Consider first the generation of satisfiable instances where the number of variables is of the form $n = 4g$. We partition the variables into g groups of size four and, for each group, generate clauses of the form:

$$(\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee \overline{d}) \wedge (\overline{a} \vee \overline{c} \vee \overline{d}) \wedge (\overline{b} \vee \overline{c} \vee \overline{d})$$

This permits at most two variables from the set $\{a, b, c, d\}$ to be `true` and therefore in total at most $2g = \frac{n}{2}$ variables can be `true`.

Now we partition the variables into a different collection of g groups and, for each group, generate clauses of the form:

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

This time at most two variables from each group can be `false` and overall at most $2g = \frac{n}{2}$ variables can be `false`. Taking these two sets of clauses together it can be seen that overall exactly half of the variable must be `true` and exactly half must be `false`. Instances of this form are not very difficult for current solvers - on an example configuration zchaff took less than 1 second to solve a 52 variable (312 literals) instance.

It is not clear that such instances must be satisfiable, i.e. that there is always an assignment of exactly two variables `true` for each of the first set of groups which also makes exactly two variables `true` for each of the second set of groups, but every such instance tested so far has been satisfiable.

[*]Institute of Electronics, Communications and Information Technology, Queen's University Belfast

3 Unsatisfiable instances

To generate unsatisfiable instances we choose to have an odd number of variables and force $\frac{n+1}{2}$ to be `true` at the same time as forcing $\frac{n+1}{2}$ to be `false` - clearly impossible. We achieve this by replacing the final group of four variables with a group of five variables and, <u>for this group only</u>, by generating clauses of the form

$$(\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{e}) \wedge (\overline{a} \vee \overline{b} \vee \overline{d} \vee \overline{e}) \wedge (\overline{a} \vee \overline{c} \vee \overline{d} \vee e) \wedge (\overline{b} \vee \overline{c} \vee \overline{d} \vee e)$$

This allows at most two from this group to be `true`. Correspondingly for the second round of positive literals we allow at most two from the final group to be `false`. These unsatisfiable instances were much harder to solve. For example, zchaff took more than 8 seconds to solve a 53 variable (348 literals) instance.

4 Partitions

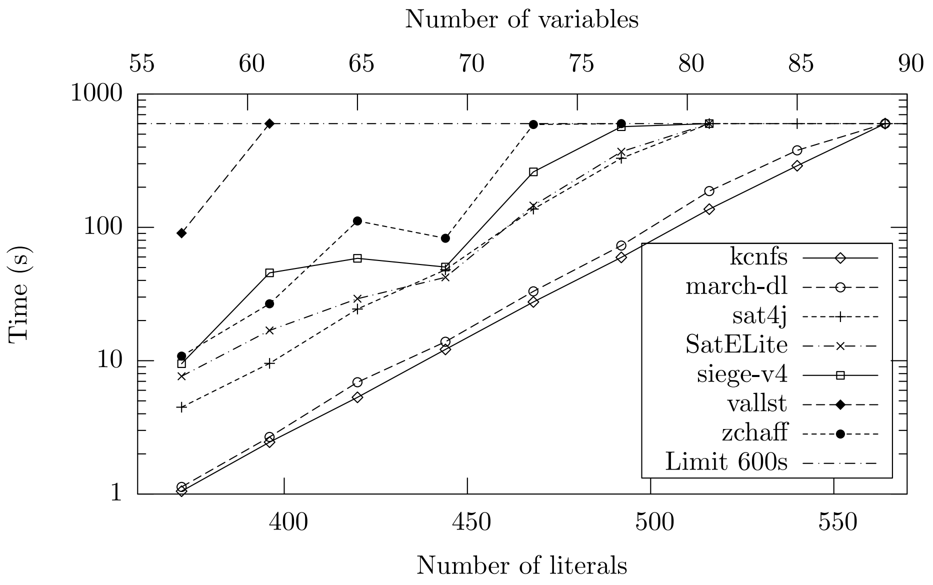
To create difficult instances we need to ensure that there is as little connection as possible between the two partitions. The first partition is the straightforward $\{1, 2, 3, 4\}$, $\{5, 6, 7, 8\}$ etc. The second partition is obtained by using simulated annealing, with a weight function which tries to minimise the extent to which the original partition is reflected in the second one.

5 Conclusions

The generated benchmarks (all unsatisfiable) appear to be amongst the most difficult for their size, both in terms of number of variables and in terms of number of literals. For some well-known solvers running under linux on a 3-GHz processor, the largest benchmarks solved within 600 seconds were as follows:

Solver	**Benchmark**	**No. vars**	**No. lits**	**Solver**	**Benchmark**	**No. vars**	**No. lits**
kcnfs	s85-100.cnf	85	540	siege_v4	s77-100.cnf	77	492
march_dl	s85-100.cnf	85	540	vallst	s57-100.cnf	57	372
sat4j	s77-100.cnf	77	492	zchaff	s73-100.cnf	73	468
SatELite	s77-100.cnf	77	492				

The complete results are given below. Note that the time scale is logarithmic and that a time of 600s means that the solver did not finish within 600s. All of these benchmarks were generated using a seed of 100 and it can be seen that the times for kcnfs and march_dl increase in a very regular manner whereas other solvers found s69-100.cnf (69 variables, 444 literals) surprisingly easy. There is scope for further investigation using different random seeds.



The benchmarks are available from
`http://www.cs.qub.ac.uk/~i.spence/sgen`.