The Set Union Model Counter SUMC2

Ivor Spence

Queen's University Belfast
i.spence@qub.ac.uk

Abstract

The Set Union Model Counting solver SUMC2 addresses the problem of counting the number of models satisfying a Conjunctive Normal Form propositional expression. It uses an algorithm which combines counts of models eliminated by each clause to obtain a count of the total number models without explicitly identifying the models. It is a (slightly) modified version of the SUMC1 solver and it was written as an entry to the model counting 2021 competition [3].

1 Introduction

SUMC2 is new version of the SUMC1 solver written specifically to test an idea about how to count the number of models which satisfy a propositional expression. SUMC1 was written to enter into the first model counting competition [2]. The solver is based around counting how many models are eliminated by each clause because they fail to satisfy it, and using an extended mathematical result giving the cardinality of a union of sets to determine how many models are eliminated overall. The GNU Multiple Precision Arithmetic Library (GMP [1]) is used to manipulate the large integer values which arise because the possible number of models is so large.

The solver calculates exactly how many satisfying models there are but overall its performance is not expected to be competitive with the best other solvers when considered across the full range of benchmarks, in particular in an environment in which memory is constrained. It is however possible that it will perform well on small benchmarks which other solvers may find difficult.

The source code of the solver is available at https://www.github.com/ivor-spence/sumc.

2 Mathematical Basis

We use the following notation:

$N \in \mathbb{N}$	Maximum variable
$V = \{1 \dots N\}$	Set of variables
$L = \{v \in V\} \cup \{-v \vert v \in V\}$	Set of literals
$C \subset \mathcal{P}(L)$	Set of clauses
$M \subset \mathcal{P}(L)$	Set of models (assignments of values to variables)
$E : C \to \mathcal{P}(M)$	The set of models eliminated by a clause

We assume that:

- no clause contains two clashing literals
- no model contains two clashing literals
- every model m gives a value to each variable, i.e. $|m| = N$.

The starting point for the algorithm is to recognise that the total number of possible models using N variables is 2^N. For any clause $c \in C$, the models eliminated are those for which there is a clash with every literal in c. That leaves a further $N - |c|$ variables which can be either positive or negative. The number of models eliminated by c is therefore

$$|E(c)| = 2^{N-|c|} \tag{1}$$

For example if $N = 3$, the clause $\{1, 2\}$ eliminates the 2 assignments $\{-1, -2, -3\}$ and $\{-1, -2, 3\}$. So $E(\{1, 2\}) = \{\{-1, -2, -3\}, \{-1, -2, 3\}\}$ and $|E(\{1, 2\})| = 2 = 2^{3-2} = 2^{N-|\{1,2\}|}$.

We wish to find the number of models which satisfy *every* clause, which means that we wish to count how many models are eliminated by *any* clause. We need to use the union of all the sets of eliminated models and so the model count MC is

$$2^N - \left| \bigcup_{c \in C} E(c) \right| \tag{2}$$

Thus we need to find the cardinality of a union of multiple sets.

2.1 Cardinality of a set union

The cardinality of the union of two sets can be determined by

$$|A \cup B| = |A| + |B| - |A \cap B|$$

For three sets we have

$$|A \cup B \cup C| = |A| + |B| + |C| - (|A \cap B| + |A \cap C| + |B \cap C|) + |A \cap B \cap C|$$

In general, if S is a collection of sets and $n \leq |S|$ we denote the collection obtained by the intersections of every n elements of S by

$$\bigcap_n S \qquad \text{(and simlarly for } \bigcup_n S\text{)}$$

For example

$$\bigcap_2 \{A, B, C\} = \{A \cap B, A \cap C, B \cap C\}$$

Then

$$\left| \bigcup_{s \in S} \right| = \sum_{i=1}^{|S|} \left((-1)^{i+1} \sum_{s \in \bigcap_i S} |s| \right) \tag{3}$$

That is, the cardinality of the union is the sum of the cardinalities of the sets, minus the sum of the cardinalities of all intersections of two sets, plus the sum of the cardinalities of all intersections of three sets et cetera.

2.2 Count of eliminated models

To apply equation (3) to the problem of counting eliminated models, we need to determine the cardinality of the intersection of the sets of models eliminated by the clauses in a set. This is the models eliminated by every clause in the set. Suppose S is a set of clauses. If any pair of clauses in S clash on any literal there will be no model eliminated by both these clauses. Assuming that there are no such clashes it can be seen that

$$\bigcap_{c \in S} E(c) = E(\bigcup_{c \in S} c) \tag{4}$$

Thus for example the models eliminated by both $\{1,2\}$ and $\{1,3\}$ are exactly those eliminated by $\{1,2,3\}$. We use the symbols e to represent a set of eliminated models and u to represent a set of literals obtained by taking the union of a set of non-clashing clauses. Combining these results we get:

$$\begin{aligned}
MC &= 2^N - \sum_{i=1}^{|C|} \left((-1)^{i+1} \sum_{e \in \bigcap_i E(C)} |e| \right) && \text{(by (2) and (3))} \\
&= 2^N - \sum_{i=1}^{|C|} \left((-1)^{i+1} \sum_{u \in \bigcup_i C} |E(u)| \right) && \text{(by (4))} \\
&= 2^N - \sum_{i=1}^{|C|} \left((-1)^{i+1} \sum_{u \in \bigcup_i C} 2^{N-|u|} \right) && \text{(by (1))} \\
&= \sum_{i=0}^{|C|} \left((-1)^{i} \sum_{u \in \bigcup_i C} 2^{N-|u|} \right) && \text{(taking } \bigcup_0 C = \{\{\}\}) \qquad (5)
\end{aligned}$$

where the unions in equation (5) are only over sets of clauses with no clashing literals.

3 Basic Algorithm

The main loop of Algorithm 1 constructs all non clashing unions of clauses. As each new clause is considered, the list of generated unions is effectively doubled in size (except for cases with clashing literals) to include each previous union both with and without adding the new clause. In this way all the clause unions in equation (5) are explicitly constructed.

4 Optimisations

If it were not for the fact that some new unions are not created because of a clash between c and u on line 7 in Algorithm 1 the length of the union list would double for each iteration of the outer for loop and even with this reduction the algorithm is hopelessly inefficient. However, several improvements are possible.

4.1 Unit clause propagation

The first step is to propagate any unit clauses in the same way that SAT-solvers do. This standard technique can significantly reduce the size of some problems.

Algorithm 1 Basic model counting algorithm for clauses C with N variables

```
1:  count ← 2^N
2:  sign ← -1
3:  unionList ← [{}]
4:  for c ∈ C do
5:    nextUnionList ← unionList
6:    for u ∈ unionList do
7:      if c does not clash with u then
8:        newUnion ← c ∪ u
9:        count ← count + sign * 2^(N−|newUnion|)
10:       nextUnionList ← nextUnionList ++ newUnion
11:     end if
12:   end for
13:   sign ← -sign
14:   unionList ← nextUnionList
15: end for
16: print count
```

4.2 c ∪ u ⊇ c'

If the new union c ∪ u created on line 8 is a superset of one of the original members C which have not yet been processed, this new union can be safely ignored. To see this, note that for subsequent unions created from c ∪ u, the same value can be obtained with and without using c'. These two versions will have the number of clauses differing by 1, so their signs will be opposite and the corresponding contributions to the count will cancel out.

4.3 Combining generated clauses

It can be that line 8 generates a union which has been generated before. It is possible to associate with each generated union the number of times it is used and only keep one copy. Thus the lists unionList and newUnionList become sets.

4.4 Processing a variable has finished

For any given variable, there is a clause c in the main loop (line 4) which is the last one to include that variable. From this point on there is no need to distinguish generated clauses on the basis of this variable and the size of the set of generated clauses is correspondingly reduced.

4.5 Sorting variables and clauses

As the main loop progresses through the list of original clauses then, we consider a variable to become *active* when it first appears in a clause, and *inactive* again when all clauses mentioning it have been processed. The new unions generated only need to contain active variables which means that it is beneficial to minimise the number of these. We first relabel the variables using a force-based linear arrangement approximation algorithm as used in the tts solver [4] and then sort the clauses according to their variables.

5 Revised Algorithm

In the revised algorithm, instead of a single global count of models we associate such a value with each union of clauses. An abstract version of the final algorithm is:

Algorithm 2 More efficient model counting algorithm for clauses C with N variables

```
1: propagate unit clauses
2: sort variables and clauses
3: count({}) ← 2^N
4: sign ← -1
5: unionSet ← { {} }
6: for c ∈ C do
7:    nextUnionSet ← unionSet
8:    for u ∈ unionSet do
9:       if c does not clash with u then
10:         newUnion ← c ∪ u
11:         newUnion ← newUnion \ any newly inactive variables
12:         if ∄ unprocessed c' ∈ C · newUnion ⊇ c' then
13:            count(newUnion) ← count(newUnion) + sign * 2^(N−|newUnion|)
14:            nextUnionSet ← nextUnionSet ∪ {newUnion}
15:         end if
16:      end if
17:   end for
18:   sign ← -sign
19:   unionSet ← nextUnionSet
20: end for
21: totalCount ← 0
22: for u ∈ unionSet do
23:    totalCount ← totalCount + count(u)
24: end for
25: print totalCount
```

6 Modifications since SUMC1

SUMC2 contains two small modifications since SUMC1. These result in slightly improved performance.

6.1 Variable ordering

The parameters for the force-based variable ordering algorithm have been modified by significantly increasing the initial forces applied.

6.2 Reclaiming memory

Memory allocated by the GMP routines is recovered more aggressively.

7 Conclusions

SUMC2 is not currently capable of solving many of the publicly released benchmarks from the MC2021 competition, but there are some benchmarks for which it seems to be one of the few successful solvers. A detailed analysis will be completed once the full competition results are available. It is anticipated that focussing on the variable sorting method will be the next stage of development.

References

[1] Torbjörn Granlund et al. The gnu multiple precision arithmetic library.

[2] Markus Hecher and Johannes K. Fichte. Model counting 2020. 2020.

[3] Markus Hecher and Johannes K. Fichte. Model counting 2021. 2021.

[4] Ivor Spence. tts: A sat-solver for small, difficult instances. *Journal on Satisfiability. Boolean Modeling and Computation*, 4(2):173–190, 2008.