The Mathematica® Journal

Generating Minimally Unsatisfiable Conjunctive Normal Forms

Robert Cowen

A method of generating minimally unsatisfiable conjunctive normal forms is introduced. A conjunctive normal form (CNF) is minimally unsatisfiable if it is unsatisfiable and such that removing any one of its clauses results in a satisfiable CNF.

Introduction

Ivor Spence [1] introduced a method for producing small unsatisfiable formulas of propositional logic that were difficult to solve by most SAT solvers at the time, which we believe was because they were usually minimally (i.e. just barely) unsatisfiable. Kullmann and Zhao [2] claim that minimally unsatisfiable formulas are "the hardest examples for proof systems." We will generalize Spence's construction and show that it can be used to generate minimally unsatisfiable propositional formulas in conjunctive normal form, that is, formulas that are unsatisfiable but such that the removal of even a single clause produces a satisfiable formula. In addition to increasing our understanding of the satisfiability problem, these formulas have important connections to other combinatorial problems [3].

Definitions

We assume the reader has at least a minimal acquaintance with propositional logic and truth tables. An *interpretation I* of a propositional formula is an assignment of truth values to its propositional variables. A propositional formula is *satisfiable* if there is an interpretation I that makes it true when evaluated using the usual truth table rules. A *literal* is a propositional variable or a negated propositional variable. A *clause* is a disjunction of literals; if it contains exactly k literals, we call it a *k-clause*. A *conjunctive normal form* (or *CNF*) is a conjunction of clauses. A k-CNF is a conjunction of k-clauses.

For example, $(\neg p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r)$ is a 3-CNF. It is often convenient to think of CNFs as a list of lists of literals; in this format, the 3-CNF example would be written as `{{¬ p, q, r}, {p, ¬ q, ¬ r}}`. This way of writing CNFs is quite common in computer science and is the approach that we took in [4], where we showed how the famous Davis–Putnam algorithm for satisfiability testing could be easily programmed in Mathematica. `SatisfiableQ`, Mathematica's built-in function for satisfiability testing, requires replacing "¬" by "!", "∨" by "||" and "∧" by "&&"; so in Mathematica the 3-CNF example is written as `(! p || q || r) && (p || ! q || ! r)`. In this article, we adopt the "list of lists" approach for programming purposes and then convert to Mathematica's format when testing for satisfiability with `SatisfiableQ`.

Constructing Unsatisfiable CNFs

In this section, we show how to generalize a construction of Ivor Spence [1] that produces unsatisfiable 3-CNFs that are relatively short but take a relatively long time to verify that they are indeed unsatisfiable using standard computer programs, even though it is relatively easy, as we shall show, to demonstrate that they are unsatisfiable. (Perhaps humans are not replaceable by computers, after all!)

Given positive integers g and k, suppose the $2\,(k-1)\,g+1$ propositional variables $p_1, p_2, \ldots, p_{2\,(k-1)\,g+1}$ are partitioned in order into $g-1$ sets of size $2\,k-2$ and one set of size $2\,k-1$. For each cell of the partition, form all k-clauses from the p's in that cell and let C_1 be the conjunction of all these k-clauses. If C_1 is to be true under an interpretation I, no more than $k-1$ of the p-variables from each partition cell can be false, since if k were false, the k-clause containing exactly these p-variables would be false, as would their conjunction C_1. Thus no more than $(k-1)\,g$ of the p-variables can be false under I.

Next let $q_1, q_2, \ldots, q_{2\,(k-1)\,g+1}$ be a random permutation of the p's and partition these q's just as the p's were partitioned. However, this time, for each cell of the partition, form all k-clauses from the negated q-variables in that cell and let C_2 be the conjunction of all these k-clauses. Reasoning as before, no more than $(k-1)\,g$ q-variables can be true under I.

Let $C = C_1 \wedge C_2$. If C is to be true under some interpretation I, both C_1 and C_2 must be true under I; thus no more than $(k-1)\,g$ p-variables can be false and no more than $(k-1)\,g$ q-variables can be true under I. Since the q-variables are permuted p-variables, it follows that no more than $(k-1)\,g$ p-variables can be true under I. However, $(k-1)\,g + (k-1)\,g = 2\,(k-1)\,g < 2\,(k-1)\,g + 1$, the number of p-variables in C! Thus C is an unsatisfiable CNF, because there is no interpretation of all its $2\,(k-1)\,g+1$ p-variables that makes C true.

Minimally Unsatisfiable CNFs

Suppose next that we drop one of the clauses in C, say for example, $p = p_1 \vee p_2 \vee \ldots \vee p_k$; let $C_1{}^* = C_1 \backslash p$ and let $C^* = C_1{}^* \wedge C_2$. Let I be an interpretation that assigns false to $p_1, p_2, \ldots, p_k$ and true to the remaining variables in the first p-cell. As long as no more than $k-1$ p-variables in each of the remaining cells of the partition of the p's are assigned the value false, $C_1{}^*$ would be true under I. Whether or not C_2 and hence C^* are true under I depends on whether I also has the property that at most $k-1$ p-variables in each cell of the partition of the randomly permuted variables (the q's) are assigned the value true under I. While this is unlikely for any given interpretation I, there are so many interpretations satisfying $C_1{}^*$ that it is most likely that some such interpretation I has this property and the reduced CNF C^* will then be true under I.

We will investigate this intuitive argument. For C^* to be true, the k p-variables $p_1, p_2, \ldots, p_k$ can now be false in the first cell, as long as each of the remaining $g-1$ cells in the p-partition has at most $k-1$ false variables; thus $k + (k-1)(g-1)$ p-variables can be false and, as above, $(k-1)\,g$ p-variables can be true. However, $k + (k-1)(g-1) + (k-1)\,g = 2(k-1)\,g + 1$, and the argument showing C to be unsatisfiable cannot be applied to C^*.

Programming

First we allow for different choices for the parameters g and k. Initially we set $k = 3$ and $g = 10$. The next several steps serve to introduce the variables p_i and partition them into cells.

```
In[34]:= k = 3; g = 10;

In[35]:= P = Table[pi, {i, 2 (k - 1) g + 1 }]

Out[35]= {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15,
          p16, p17, p18, p19, p20, p21, p22, p23, p24, p25, p26, p27, p28,
          p29, p30, p31, p32, p33, p34, p35, p36, p37, p38, p39, p40, p41}
```

Define the k, g partition of the p-variables.

```
In[36]:= kgPartition[P_, k_, g_] :=
          TakeList[
           P,
           Append[Table[2 k - 2, {g - 1}], 2 k - 1]
          ]
```

Here is the *p*-partition for our example.

```
In[37]:= C1 = kgPartition[P, 3, 10]

Out[37]= {{p1, p2, p3, p4}, {p5, p6, p7, p8}, {p9, p10, p11, p12},
          {p13, p14, p15, p16}, {p17, p18, p19, p20},
          {p21, p22, p23, p24}, {p25, p26, p27, p28}, {p29, p30, p31, p32},
          {p33, p34, p35, p36}, {p37, p38, p39, p40, p41}}
```

Next we generate, negate and partition the *q*-variables.

```
In[38]:= C2 = kgPartition[! # & /@ RandomSample@P, 3, 10]

Out[38]= {{! p8, ! p2, ! p30, ! p12}, {! p37, ! p5, ! p36, ! p40},
          {! p11, ! p13, ! p24, ! p31}, {! p33, ! p7, ! p35, ! p41},
          {! p16, ! p19, ! p32, ! p21}, {! p39, ! p3, ! p20, ! p25},
          {! p27, ! p38, ! p29, ! p28}, {! p18, ! p34, ! p15, ! p17},
          {! p10, ! p26, ! p4, ! p6}, {! p23, ! p9, ! p22, ! p1, ! p14}}
```

We join `C1` and `C2` and form all *k*-sets from the result.

```
In[39]:= Flatten[Subsets[#, {k}] & /@ Join[C1, C2], 1]

Out[39]= {{p1, p2, p3}, {p1, p2, p4}, {p1, p3, p4}, {p2, p3, p4},
          {p5, p6, p7}, {p5, p6, p8}, {p5, p7, p8}, {p6, p7, p8},
          {p9, p10, p11}, {p9, p10, p12}, {p9, p11, p12}, {p10, p11, p12},
          {p13, p14, p15}, {p13, p14, p16}, {p13, p15, p16}, {p14, p15, p16},
          {p17, p18, p19}, {p17, p18, p20}, {p17, p19, p20}, {p18, p19, p20},
          {p21, p22, p23}, {p21, p22, p24}, {p21, p23, p24}, {p22, p23, p24},
          {p25, p26, p27}, {p25, p26, p28}, {p25, p27, p28}, {p26, p27, p28},
          {p29, p30, p31}, {p29, p30, p32}, {p29, p31, p32}, {p30, p31, p32},
          {p33, p34, p35}, {p33, p34, p36}, {p33, p35, p36}, {p34, p35, p36},
          {p37, p38, p39}, {p37, p38, p40}, {p37, p38, p41}, {p37, p39, p40},
          {p37, p39, p41}, {p37, p40, p41}, {p38, p39, p40}, {p38, p39, p41},
          {p38, p40, p41}, {p39, p40, p41}, {! p8, ! p2, ! p30},
          {! p8, ! p2, ! p12}, {! p8, ! p30, ! p12}, {! p2, ! p30, ! p12},
          {! p37, ! p5, ! p36}, {! p37, ! p5, ! p40}, {! p37, ! p36, ! p40},
          {! p5, ! p36, ! p40}, {! p11, ! p13, ! p24}, {! p11, ! p13, ! p31},
          {! p11, ! p24, ! p31}, {! p13, ! p24, ! p31}, {! p33, ! p7, ! p35},
          {! p33, ! p7, ! p41}, {! p33, ! p35, ! p41}, {! p7, ! p35, ! p41},
          {! p16, ! p19, ! p32}, {! p16, ! p19, ! p21}, {! p16, ! p32, ! p21},
          {! p19, ! p32, ! p21}, {! p39, ! p3, ! p20}, {! p39, ! p3, ! p25},
          {! p39, ! p20, ! p25}, {! p3, ! p20, ! p25}, {! p27, ! p38, ! p29},
          {! p27, ! p38, ! p28}, {! p27, ! p29, ! p28}, {! p38, ! p29, ! p28},
          {! p18, ! p34, ! p15}, {! p18, ! p34, ! p17}, {! p18, ! p15, ! p17},
          {! p34, ! p15, ! p17}, {! p10, ! p26, ! p4}, {! p10, ! p26, ! p6},
          {! p10, ! p4, ! p6}, {! p26, ! p4, ! p6}, {! p23, ! p9, ! p22},
          {! p23, ! p9, ! p1}, {! p23, ! p9, ! p14}, {! p23, ! p22, ! p1},
          {! p23, ! p22, ! p14}, {! p23, ! p1, ! p14}, {! p9, ! p22, ! p1},
          {! p9, ! p22, ! p14}, {! p9, ! p1, ! p14}, {! p22, ! p1, ! p14}}
```

`SpenceList` puts these steps together. The argument `s` is a permutation of $\{1, 2, 3, \ldots, 2(k-1)g+1\}$.

```
In[40]:= SpenceList[k_, g_, s_] := Module[
           {P},
           P = Table[p_i, {i, 2 (k - 1) g + 1 }];
           Flatten[
            Subsets[#, {k}] & /@
             Join[
              kgPartition[P, k, g],
              kgPartition[! p_# & /@ s, k, g]
             ],
            1]
          ] /; Sort@s == Range[2 (k - 1) g + 1]
```

For the experiments that follow, leaving out the third argument uses a random permutation.

```
In[41]:= SpenceList[k_, g_] :=
          SpenceList[k, g, RandomSample@Range[(2 k - 2) g + 1]]
```

Because of `RandomSample`, the negated pieces can change from one run to the next.

```
In[42]:= C3 = SpenceList[3, 10]
```

```
Out[42]= {{p1, p2, p3}, {p1, p2, p4}, {p1, p3, p4}, {p2, p3, p4},
          {p5, p6, p7}, {p5, p6, p8}, {p5, p7, p8}, {p6, p7, p8},
          {p9, p10, p11}, {p9, p10, p12}, {p9, p11, p12}, {p10, p11, p12},
          {p13, p14, p15}, {p13, p14, p16}, {p13, p15, p16}, {p14, p15, p16},
          {p17, p18, p19}, {p17, p18, p20}, {p17, p19, p20}, {p18, p19, p20},
          {p21, p22, p23}, {p21, p22, p24}, {p21, p23, p24}, {p22, p23, p24},
          {p25, p26, p27}, {p25, p26, p28}, {p25, p27, p28}, {p26, p27, p28},
          {p29, p30, p31}, {p29, p30, p32}, {p29, p31, p32}, {p30, p31, p32},
          {p33, p34, p35}, {p33, p34, p36}, {p33, p35, p36}, {p34, p35, p36},
          {p37, p38, p39}, {p37, p38, p40}, {p37, p38, p41}, {p37, p39, p40},
          {p37, p39, p41}, {p37, p40, p41}, {p38, p39, p40}, {p38, p39, p41},
          {p38, p40, p41}, {p39, p40, p41}, {!p28, !p34, !p35},
          {!p28, !p34, !p11}, {!p28, !p35, !p11}, {!p34, !p35, !p11},
          {!p17, !p7, !p16}, {!p17, !p7, !p2}, {!p17, !p16, !p2},
          {!p7, !p16, !p2}, {!p26, !p39, !p5}, {!p26, !p39, !p9},
          {!p26, !p5, !p9}, {!p39, !p5, !p9}, {!p6, !p27, !p1},
          {!p6, !p27, !p33}, {!p6, !p1, !p33}, {!p27, !p1, !p33},
          {!p18, !p15, !p21}, {!p18, !p15, !p24}, {!p18, !p21, !p24},
          {!p15, !p21, !p24}, {!p10, !p25, !p8}, {!p10, !p25, !p41},
          {!p10, !p8, !p41}, {!p25, !p8, !p41}, {!p3, !p32, !p12},
          {!p3, !p32, !p4}, {!p3, !p12, !p4}, {!p32, !p12, !p4},
          {!p30, !p14, !p37}, {!p30, !p14, !p20}, {!p30, !p37, !p20},
          {!p14, !p37, !p20}, {!p29, !p22, !p13}, {!p29, !p22, !p31},
          {!p29, !p13, !p31}, {!p22, !p13, !p31}, {!p40, !p38, !p19},
          {!p40, !p38, !p23}, {!p40, !p38, !p36}, {!p40, !p19, !p23},
          {!p40, !p19, !p36}, {!p40, !p23, !p36}, {!p38, !p19, !p23},
          {!p38, !p19, !p36}, {!p38, !p23, !p36}, {!p19, !p23, !p36}}
```

The function SpenceCNF transforms a list of clauses into an expression that allows for satisfiability testing, as described in the section Definitions.

```
In[43]:= SpenceCNF[C3_] := And @@ (Or @@@ C3)
```

Equivalently, here is a longer form.

```
In[44]:= SpenceCNF[C3_] := Apply[And, Apply[Or, C3, {1}]]
```

To test that a CNF is minimally unsatisfiable, we must show both that it is unsatisfiable and that the removal of any one clause always results in a satisfiable CNF. MinimallyUnsatisfiableQ tests the satisfiability of a CNF in list form by converting it to a logical expression with SpenceCNF and applying the built-in function SatisfiableQ; then it tests if all the formulas with one clause deleted are indeed satisfiable.

```
In[45]:= MinimallyUnsatisfiableQ[C3_] :=
          And[
           Not@SatisfiableQ@SpenceCNF@C3,
           And @@ Table[
             SatisfiableQ@SpenceCNF@Delete[C3, j],
             {j, Length@C3}
            ]
          ]
```

This tests whether C3 is minimally unsatisfiable.

```
In[46]:= MinimallyUnsatisfiableQ@SpenceCNF@C3

Out[46]= True
```

Most of the unsatisfiable formulas generated in this way are, as we shall see, minimally unsatisfiable, but not always. However, we conjecture that the bigger we take k and g, the more likely we are to get a minimally unsatisfiable formula. We have done some experiments on this conjecture and discuss them in the next section.

If we remove two different clauses instead of one from our unsatisfiable formulas, the result is almost always a satisfiable formula. We define the function MinimallyUnsatisfiableQ2 and experiment with it in the next section.

```
In[47]:= aux[C3_][i_, j_] :=
          SatisfiableQ@SpenceCNF[Delete[C3, {{i}, {j}}]]
```

```
In[48]:= MinimallyUnsatisfiableQ2[C3_] := And[
           Not@SatisfiableQ@SpenceCNF@C3,
           aux[C3] @@@ Subsets[Range@Length@C3, {2}]
          ]
```

Experiments

We run some experiments to investigate the frequency of minimally unsatisfiable CNFs obtained with our Spence generalizations.

```
In[49]:= experiment[k_, g_] :=
          1 / 100 N@Count[
             Table[MinimallyUnsatisfiableQ@
               SpenceCNF@SpenceList[k, g], {i, 100}],
             True
            ]

In[50]:= Table[experiment[2, n], {n, 4, 8}]

Out[50]= {0.4, 0.33, 0.29, 0.27, 0.33}

In[51]:= Table[experiment[3, n], {n, 5, 10}]

Out[51]= {0.65, 0.71, 0.73, 0.8, 0.83, 0.86}

In[52]:= Table[experiment[4, n], {n, 2, 6}]

Out[52]= {0.39, 0.61, 0.75, 0.81, 0.91}
```

The table below summarizes some larger computer experiments we have conducted to test our conjecture that most of the formulas constructed by the above methods are minimally unsatisfiable. The third column, "$N(k, g)$", stands for the number of clauses in the CNF defined by `SpenceCNF@SpenceList[k, g]`. The fourth column gives the percentage of these clauses that were minimally unsatisfiable.

Each line in the table represents results on 500 formulas. For example, the first line constructs 500 3-CNFs, based on $4 \times 5 + 1 = 21$ variables and $8 \times 5 + 12 = 52$ clauses in each. It turned out that 62% were minimally unsatisfiable.

```
In[53]:= Text@Grid[{
            {"k", "g", "N(k,g)", "minimally\nunsatisfiable"},
            {3, 5, 52, 62 "%"}, {3, 8, 76, 77 "%"}, {3, 10, 92, 85 "%"},
            {3, 12, 108, 88 "%"}, {3, 15, 132, 89 "%"},
            {4, 6, 220, 87 "%"}}, Frame → All,
          Alignment → {{Right, Right, Right, Center}}]
```

Out[53]=

k	g	$N(k,g)$	minimally unsatisfiable
3	5	52	62 %
3	8	76	77 %
3	10	92	85 %
3	12	108	88 %
3	15	132	89 %
4	6	220	87 %

Next we look at what happens if we remove two different clauses from our Spence formulas. With $k = 3$, $g = 10$, there are 92 clauses in the Spence CNF; hence $(92 \times 91)/2 = 4186$ distinct ways to remove two different clauses. We count the number of times the resulting CNF is true in 100 trials.

```
In[54]:= Table[
          Total@
           Boole@MinimallyUnsatisfiableQ2@
             SpenceCNF@SpenceList[3, 10],
          {i, 100}
         ]

Out[54]= {4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186,
          4186, 4185, 4186, 4186, 4186, 4185, 4186, 4186, 4186,
          4186, 4186, 4185, 4186, 4186, 4185, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4185, 4186, 4186, 4185,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4185, 4186,
          4186, 4186, 4186, 4158, 4185, 4186, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186,
          4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186, 4186}

In[55]:= N@Mean@%

Out[55]= 4185.64
```

In this trial, we are very close to all Spence CNFs becoming satisfiable after removing two different clauses. We believe this is true in general.

Derangements

In this section we modify our construction to generate only derangements of the q-variables. A *derangement* is a permutation that leaves no number in its original position. There are three resource functions that deal with derangements.

```
In[56]:= ResourceFunction["Derangements"][4]

Out[56]= {{2, 1, 4, 3}, {2, 3, 4, 1}, {2, 4, 1, 3},
          {3, 1, 4, 2}, {3, 4, 1, 2}, {3, 4, 2, 1},
          {4, 1, 2, 3}, {4, 3, 1, 2}, {4, 3, 2, 1}}

In[57]:= ResourceFunction["RandomDerangement"][4]

Out[57]= {2, 1, 4, 3}
```

```
In[58]:= ResourceFunction["DerangementQ"][%]

Out[58]= True
```

It is known that derangements constitute slightly more than a third of the permutations (see [5]), as the following calculation illustrates.

```
In[59]:= Count[
          Table[ResourceFunction["DerangementQ"][
            RandomSample[Range[41]]], {i, 100}], True]

Out[59]= 33
```

We define the function `experimentD` that does "derangement" experiments.

```
In[60]:= experimentD[k_, g_] :=
          1 / 100 N@Count[
             Table[
              MinimallyUnsatisfiableQ@
               SpenceCNF@SpenceList[k, g,
                 ResourceFunction["RandomDerangement"][
                  2 (k - 1) g + 1]],
              {i, 100}],
             True
            ]

In[61]:= experimentD[3, 10]

Out[61]= 0.83

In[62]:= experimentD[3, 5]

Out[62]= 0.66

In[63]:= experimentD[4, 6]

Out[63]= 0.84
```

On the basis of these experiments we conjecture that the derangements are about as likely to produce minimally unsatisfiable formulas as permutations in general.

Conclusion

We have adapted a method of I. Spence [1] to easily obtain large numbers of unsatisfiable CNFs that are usually but not always minimally unsatisfiable. We also ran some experiments to indicate what percentages would be minimally unsatisfiable. In addition, our experiments suggest that if two different clauses are removed rather than one, the resulting formula will almost always be satisfiable. Finally we restricted the random permutations in our construction by requiring them to be derangements and saw that this gave similar percentages of minimally unsatisfiable formulas.

Acknowledgements

I am grateful to the referee for his advice and to the editor, George Beck, for greatly improving my Mathematica coding throughout the paper.

References

[1] I. Spence, "sgen1: A Generator of Small but Difficult Satisfiability Benchmarks," *ACM Journal of Experimental Algorithmics*, **15**, 2010 pp. 1.1–1.15. doi:10.1145/1671970.1671972.

[2] O. Kullmann and X. Zhao, "On Davis–Putnam Reductions for Minimally Unsatisfiable Clause-Sets," *Theoretical Computer Science*, **492**, 2013 pp. 70–87. doi:10.1007/978-3-642-31612-8_ 21.

[3] R. Aharoni and N. Linial, "Minimal Non-Two-Colorable Hypergraphs and Minimal Unsatisfiable Formulas," *Journal of Combinatorial Theory, Series A* **43**(2), 1986 pp. 196–204. doi:10.1016/0097-3165(86)90060-9.

[4] R.Cowen, M. Huq and W. MacDonald, "Implementing the Davis–Putnam Algorithm in Mathematica," *Mathematica in Education and Research*, **10**, 2005 pp, 46–55. www.researchgate.net/publication/246429822_Implementing_the_Davis-Putnam_Algorithm_in_Mathematica.

[5] Wikipedia. "Derangement." (Jul 10, 2010) en.wikipedia.org/wiki/Derangement#Limit_of_ratio_of_derangement_to_permutation_as_n_approaches_∞.

R. Cowen, "Generating Minimally Unsatisfiable Conjunctive Normal Forms," *The Mathematica Journal*, 2020. https://doi.org/10.3888/tmj.22–4.

About the Author

Robert Cowen is a Professor Emeritus at Queens College, CUNY. His main research interests are logic and combinatorics. He has enjoyed teaching students how to use Mathematica to do research in mathematics for many years.

Robert Cowen
16422 75th Avenue
Fresh Meadows, NY 11366
robert.cowen@gmail.com