SOUTHWESTERN UNIVERSITY

Brown Working Papers in the Arts & Sciences

Volume VIII

(2008)

Using Clausal Graphs to Determine the Computational Complexity of k-Bounded Positive 1-in-3 SAT

Richard Denman and Stephen Foster
Department of Mathematics and Computer Science
Southwestern University
Georgetown, TX 78627
denman@southwestern.edu

Recommended Citation

Richard Denman and Stephen Foster, (2008) "Using Clausal Graphs to Determine the Computational Complexity of k-Bounded Positive 1-in-3 SAT," Brown Working Papers in the Arts and Sciences, Southwestern University, Vol. VIII. Available at: http://www.southwestern.edu/academic/bwp/vol8/denman-vol8.pdf.

Using Clausal Graphs to Determine the Computational Complexity of *k*-Bounded Positive 1-in-3 SAT

Richard Denman and Stephen Foster
Department of Mathematics and Computer Science
Southwestern University, Georgetown, TX

Abstract

The 1-in-3 SAT problem is known to be *NP*-complete even in the absence of negated variables [1], a variant known as Positive (or Monotone) 1-in-3 SAT. In this note, we use clausal graphs to investigate a further restriction: *k*-Bounded-Positive 1-in-3 SAT (*k*BP 1-in-3 SAT), in which each variable occurs in no more than *k* clauses. We show that for $k = 2$, *k*BP 1-in-3 SAT is in the polynomial complexity class *P*, while for all $k>2$, it is *NP*-complete, providing another way of exploring the boundary between classes *P* and *NP*.

1 Introduction

Satisfiability problems play a central role in complexity theory. From the seminal work of Cook [2] and Levin [3] in showing that SAT is *NP*-complete, to recent work of Tovey [4], and Berman, Karpinski, and Scott [5], there is an extended trail of satisfiability research exploring the boundary between categories *P* and *NP*. Here we explore this boundary by investigating a restriction of the 1-in-3 SAT problem. The 1-in-3 SAT problem is known to be *NP*-complete even in the absence of negated variables [1], a variant known as Positive (or Monotone) 1-in-3 SAT. In this note we define a further restriction, ***k*-Bounded-Positive 1-in-3 SAT** (*k*BP 1-in-3 SAT), in which each variable occurs in at most *k* clauses. We will show that for $k = 2$, *k*BP 1-in-3 SAT is in complexity class *P*, while for all $k>2$, it is *NP*-complete. It is hoped that the study of these two closely related variants will shed additional light on the boundary between categories *P* and *NP*.

2 The Clausal Graph Representation for case $k = 2$

The familiar representation of an instance of 1-in-3 SAT consists of a finite set of variables and a finite set of clauses *C*. Each clause in *C* contains the disjunction of three variables, any of which might be negated, and the Boolean expression *E* to be satisfied is the conjunction of all the clauses in *C*. In addition, a solution is required to have the property that in each clause, exactly one of the variables is assigned to be *TRUE*. In an instance of 2BP 1-in-3 SAT, further restrictions are imposed, such that there are no negated variables, and such that no variable appears in more than two clauses. Such an instance can be represented by a graph *G*, in which each vertex represents a clause, and each edge represents a variable that is shared by the two clauses corresponding to the

vertex endpoints. For example, the Boolean expression consisting of the set of clauses {(*a,b,c*), (*a,d,e*), (*b,f,g*), (*c,d,f*), (*e,h,i*), (*h,j,k*), (*i,k,m*), (*m,j,p*), (*g,n,p*)} would be represented by the graph in Figure 1 below. Note that variable *n* does not appear as an edge in the graph since it is a disjunct in only one clause.

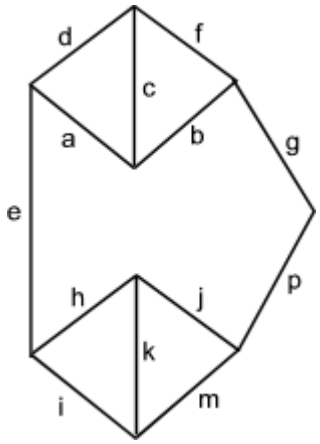


Figure 1

In view of this correspondence, we will use the terms "vertex" and "clause" interchangeably, and also the terms "edge" and "variable". One possible 1-in-3 assignment for the expression of Figure 1 assigns variables *d, b, h, m,* and *n* to *TRUE*, and all others to *FALSE*. Note that this assignment induces a matching on the graph in Figure 1, and that every vertex of degree three is incident to one of the edges assigned to *TRUE*.

3 Simplifying Assumptions for the Clausal Graph Representation

In this section, we will make several simplifying assumptions about the permissible Boolean expressions, in order to allow us to apply adaptations of two powerful results from the matching theory of graphs.

First, we will assume that no clause contains three duplicates of the same variable, as in the clause (*a,a,a*). Otherwise, the expression would be immediately unsatisfiable. No 1-in-3 assignment could satisfy this clause, since only one variable in each clause may be *TRUE*.

We will also assume that no clause contains two duplicates of the same variable, as in the clause (*a,a,b).* In the graph representation, this assumption, together with our first assumption above, will imply that there are no loop edges and no multiple edges. If there were such a clause (*a,a,b),* variable *a* would require an assignment of *FALSE*, and variable *b* would require an assignment of *TRUE*. In this case, we will replace all occurrences of variable *a* in the expression by the constant *FALSE*, and all occurrences of variable *b* to *TRUE*. In the graph representation, we will remove the loop edge and any other edges corresponding to variable *a.* this will mean that there will be some vertices that represent a clause in which at least one of the variables is required to be *FALSE*. Similarly, assigning variable *b* to *TRUE* in another clause will require the other variables in that clause to be assigned to *FALSE.* Based on these possibilities, we define the

forcing number of a vertex to be the number of variables in the corresponding clause that are required to be *FALSE.* We will assume that the forcing number of a vertex is smaller than three and that the forcing number does not exceed the difference between three and the degree of the vertex. A forcing number of three would imply that the expression has no satisfying assignment, as would a forcing number in excess of the difference between three and the degree of the vertex. Table 1 below illustrates the eight different allowable combinations of degrees and forcing numbers. The combinations shown in boldface represent vertices that are "crucial", a property that is defined in section 4 below.

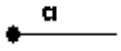
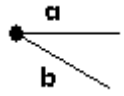
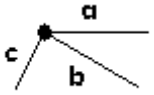
degree	forcing number	sample clause	figure
0	1	(*a, b, FALSE*)	
0	2	(*a, FALSE, FALSE*)	
1	0	(*a, b, c*)	
1	1	(*a, b, FALSE*)	
1	**2**	**(*a, FALSE, FALSE*)**	
2	0	(*a, b, c*)	
2	**1**	**(*a, b, FALSE*)**	
3	**0**	**(*a, b, c*)**	

Table 1

We will also assume that no pair of variables, say *a* and *b*, appear together in any pair of clauses, such as in (*a,b,c*) and (*a,b,d*). Equivalently, the graph may not contain a pair of vertices connected by two edges with different labels (parallel edges). If this does occur, then under all satisfying 1-in-3 assignments, variables *c* and *d* must be given the same

value, so the original expression can be simplified as follows: rename *d* to be *c*, and temporarily remove the two (now identical) clauses. Furthermore, if there previously were two other clauses each containing one of variables *c* and *d*, say (*c,e,f*) and (*d,g,h*), these now become (*c,e,f*) and (*c,g,h*). This change requires those two clauses to be joined by a new edge labeled *c.* Then, if this reduced instance has a satisfying assignment, the original also has a satisfying assignment, in which *d* will be given the same assignment as *c,* and *a* and *b* will be assigned in way that is consistent with the value assigned for *c* and *d.* Likewise, if there is no assignment satisfying the reduced instance, then there can be no assignment to the original.

Combining these simplifying assumptions will allow us to adapt ideas from the matching theory of graphs to provide a polynomial algorithm for the 2BP 1-in-3 SAT. To this end, we define a **simplified** instance of 2BP 1-in-3 SAT to be one with all the simplifying assumptions described above. These assumptions imply that, in the corresponding graph, there are no loop edges, no multiple edges, and no parallel edges, no vertex with forcing number three, and no vertex with forcing number in excess of the difference between three and the degree of the vertex. It is important to note that each of these simplifications can be carried out in polynomial time. Likewise, if a 1-in-3 assignment is found for the simplified instance, it is a polynomial process to reverse each of the simplifications and extend the assignment to any variables restored by these reversals.

4 Optimum Matching in a Clausal Graph

In order to find a satisfying 1-in-3 assignment, we seek a set of variables that may be assigned to be true in such a way that the 1-in-3 property is realized. In the graph representation these variables will correspond to a mutually disjoint collection of edges, which is known as a matching. The matching will have certain characteristics, as specified in Theorem 1 below. We will say that a matching *M* **covers** a vertex *b* if *b* is incident with one of the edges of *M*, and we will define a **crucial** vertex as one that either has degree three, or has degree two and forcing number one, or has degree one and forcing number two. Note that an alternative definition for a crucial vertex is a vertex for which the degree and forcing number sum to three (which is not possible for a vertex of degree zero). Thus a non-crucial vertex will correspond to a clause that contains at least one variable that is not forced to FALSE, and that does not appear in any other clause. The three crucial vertex types are shown in boldface in Table 1, and the remaining five types are non-crucial.

Theorem 1: A simplified (in the sense of section 3 above) 2BP 1-in-3 SAT instance has a solution if and only if there is a matching *M* in the corresponding clausal graph *G* such that *M* covers all the crucial vertices of *G*.

> Proof: If there is a satisfying assignment, then those variables that are assigned to *TRUE*, and that appear in two clauses, form a matching in the graph that covers all the crucial vertices of *G*. Conversely, if there is

such a matching, then the assignment can be completed, because the only remaining uncovered vertices will be non-crucial vertices. These vertices must represent clauses with at least one variable that appears in no other clause. Each of these "singleton" variables can be assigned to be *TRUE*, and then all remaining unassigned variables may be assigned to false, to complete the assignment.

We will use Theorem 1 to guide an adaptation of the well-known polynomial algorithm [6] of J. Edmonds for finding a maximum matching in a graph. A maximum matching is defined as a matching which contains a maximum number of edges, but the needs of a matching for our clausal graphs are slightly different. In particular, we seek a matching that covers a maximum number of crucial vertices. It is easy to see in Figure 2 that a maximum matching, represented here by the bold edges, might not provide a 1-in-3 assignment, even when one is possible. This difference leads us to make the following definitions.

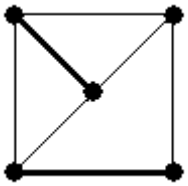


Figure 2

A matching M is **optimum** if no other matching covers more crucial vertices than M. An alternating path is a **trading path** if it begins at an uncovered crucial vertex and ends at a non-crucial vertex (whether covered or not). Here we use the standard definition of an **alternating path** for M: a path in which each internal vertex is incident with exactly one edge of M. The well-known maximum matching theorem of Berge [7], which is utilized by Edmond's algorithm, readily adapts from a statement about a maximum matching to a statement about an optimum matching.

Theorem 2: A matching M is optimum if and only if there is no trading path for M.

Proof: If there is a trading path, then modifying the matching by successive reversals along the path will produce a matching M' which covers more crucial vertices than M, so M is not optimum. This is the contrapositive of the forward direction of the double implication. The proof of the other direction continues to follow the proof of Berge's theorem. That is, suppose that M is not optimum. Then there is a matching M' covering more crucial vertices than M. Since the symmetric difference $M \oplus M'$ consists of components that are either paths or circuits, at least one of these components, say P, must be a path from a crucial vertex covered only by M' to a non-crucial vertex, covered by either M or M'. In either case, P will serve as a trading path for M.

5 Adapting Edmond's Algorithm to Optimum Matchings: Trading Paths

The only adaptations required in Edmond's algorithm [6] are as follows: to always start at an uncovered crucial vertex, and to end the breadth-first search for an alternating path when a non-crucial (whether covered or uncovered) vertex is encountered. The change in focus from the search for augmenting paths to the slightly weaker search for trading paths is required for the problem at hand. Determining whether or not there is a matching containing a maximum number of edges (that is, a maximum matching) is not sufficient to determine satisfiability. Rather what is needed is to determine whether or not it is possible to find a matching that covers a maximum number of the crucial vertices; that is, whether or not there is an optimum matching.

Shrinking blossoms and unshrinking them proceeds exactly as with Edmonds' algorithm.

6 Dropping the Positivity Requirement: 2-Bounded 1-in-3 SAT

It is important to note that since the above algorithm accommodates vertices with a positive forcing number, it is possible to drop the requirement that all variables be positive, since every negated variable $\overline{a}$ can be replaced by a fresh variable b, and a new clause *(a, b, false)* can be introduced to provide the complement relationship. Thus the problem of deciding the 1-in-3 satisfiability of a CNF Boolean expression with 3 disjuncts per clause also has a polynomial solution, provided that no variable appears, in either positive or negated form, in more than two clauses. We will refer to this problem as the 2-Bounded 1-in-3 SAT problem, or 2B 1-in-3 SAT for short (dropping the "positive" property).

7 *NP*-Completeness for Case $k>2$

Consistent with the simplifying assumptions above, we will assume that in each instance of *kBP* 1-in-3 SAT, all the variables in each clause are distinct, and that some of the clauses may contain variables presumed to be assigned *FALSE*.

Theorem 2: Every instance of a Positive 1-in-3 SAT problem can be reduced in polynomial time and space to an instance of 3BP 1-in-3 SAT.

> Proof:
> Let C be the set of clauses for a Positive 1-in-3 SAT expression. Then C represents an instance of *kBP* 1-in-3 SAT, where k is the largest integer for which some variable occurs in k different clauses. Let b be a variable that occurs in k different clauses, for some $k > 2$. Divide these k clauses arbitrarily into two groups of equal size (if k is even), or two groups that differ in size by 1 (if k is odd). In one of the two groups, replace all

occurrences of variable *b* by a fresh variable *c*, introduce another fresh variable *d* and introduce two new clauses (*b, d, false*) and (*c, d, false*), to obtain a new set of clauses *C'* with two more variables, and two more clauses than *C*. This new set of clauses *C'* is equivalent to *C* because variable *c* is just an alias for variable *b*. Also, neither variable *b* nor variable *c* now occur in more than $\lceil k/2 \rceil + 1$ clauses, which is smaller than *k,* provided that $k > 3$. Repeating this replacement for each variable that occurs in *k* different clauses results in a collection of clauses in which every variable occurs in fewer than *k* clauses. In this way, any 1-in-3 SAT expression can be iteratively transformed to one in which no variable occurs in more than 3 conjuncts. The computational complexity of the total reduction for each variable is $O(k * \log(k))$. Since *k* is bounded by the number of clauses, the computational complexity of reducing all the variables is $O(|V| * |C| * \log(|C|))$, where $|V|$ is the number of variables, and $|C|$ is the number of clauses, in the original expression. The space complexity of this algorithm can similarly be shown to be $O(|V| * |C|))$, since each step introduces 2 new variables, and 2 new clauses, and the total additional space introduced in the total reduction for each variable is at most a constant multiple of *k.*

Theorem 2 implies that 3BP 1-in-3 SAT is *NP*-Complete, since Positive 1-in-3 SAT is *NP*-complete. It should be noted that for $k = 3$, $\lceil k/2 \rceil + 1 = 3$, so no reduction is achieved in this case. For this reason, the proof above does not provide a polynomial reduction to 2BP 1-in-3 SAT.

8 Possibilities for Further Exploration

An extension of the polynomial reduction in section 7 above to the case $k = 3$ would imply *P=NP*, as would an extension of the polynomial algorithm of section 5 to the case $k = 3$. Or perhaps case $k = 3$ may be restricted further, in order to continue exploring the boundary between *P* and *NP*. The authors of this note are preparing another note [8] with some results in these directions.

9 References

[1] T. J. SCHAEFER, 1978. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing* (New York), ACM, New York, 216–226.

[2] STEPHEN COOK, 1971. The Complexity of Theorem Proving Procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, 151–158.

[3] LEONID LEVIN, 1973. Universal'nye perebornye zadachi. *Problemy Peredachi Informatsii* **9** (3): 265–266.

[4] CRAIG A. TOVEY, 1984. A simplified NP-complete satisfiability problem. In *Discrete Applied Mathematics*, Volume 8, Issue 1, April 1984, 85-89.

[5] PIOTR BERMAN, MAREK KARPINSKI, AND ALEXANDER D. SCOTT, 2006. Computational complexity of some restricted instances of 3-SAT. In *Discrete Applied Mathematics*, Volume 155, Issue 5, March 2007, 649-653.

[6] JACK EDMONDS, 1965. Paths, trees, and flowers. *Canadian Journal of Mathematics*, Volume 11, 1965, 449-467.

[7] CLAUDE BERGE, 1957. Two Theorems in Graph Theory. In *Proceedings of the National Academy of Sciences USA*, Volume 43, Issue 9, September 1957, 842–844.

[8] RICHARD DENMAN AND STEPHEN FOSTER, 2008. Matchings in Primitive Hypergraphs, in preparation.

[9] M. R. GAREY AND D. S. JOHNSON, 1979. *Computers and Intractability*. W. H. Freeman, New York.