SOUTHWESTERN UNIVERSITY

Brown Working Papers in the Arts & Sciences

Volume IX

(2009)



k-Bounded Positive Not All Equal LE3-SAT

Richard Denman, Associate Professor of Mathematics and Computer Science, and Barbara M. Anthony
Department of Mathematics and Computer Science
Southwestern University
Georgetown, TX 78627
denman@southwestern.edu

Recommended Citation:

Richard Denman and Barbara M. Anthony, (2009) "*k*-Bounded Positive Not All Equal LE3-SAT," Brown Working Papers in the Arts and Sciences, Southwestern University, Vol. IX. Available at: http://www.southwestern.edu/academics/bwp//vol9/denman-vol9.pdf.

k-Bounded Positive Not All Equal LE3-SAT

Barbara M. Anthony and Richard Denman
Department of Mathematics and Computer Science
Southwestern University, Georgetown, TX

2nd February 2010

Abstract

The not all equal (NAE) 3-SAT problem is known to be NP-complete even in the absence of negated variables (see [11]), a variant known as positive (or monotone) NAE 3-SAT. In this article, we investigate a related category, dubbed positive NAE LE3-SAT (PNAE LE3-SAT), in which each clause has *at most* three variables, none of which is negated, and an assignment is sought that will satisfy such an expression in a way that meets the NAE restriction. We show that PNAE LE3-SAT is NP-complete even when no variable occurs more than three times (3B-PNAE LE3-SAT), and that PNAE LE3-SAT is in complexity class P when no variable occurs more than twice (2B-PNAE LE3-SAT).

Keywords: Positive NAE 3-SAT; clausal graph; polynomial complexity; nondeterministic polynomial complexity; NP-complete

1 Introduction

We begin by defining a weakened variant of 3-SAT, called **LE3-SAT**, for "less than or equal 3-SAT", in which a boolean expression in conjunctive normal form has no clause with more than three disjuncts. Imposing the usual NAE condition on LE3-SAT defines the variant **NAE LE3-SAT** (not all equal LE3-SAT). An additional requirement that all variables must be positive (no negated variables) defines the variant **PNAE LE3-SAT** (positive not all equal LE3-SAT). A further restriction that no variable occurs in more than k clauses defines the variant **kB-PNAE LE3-SAT** (k-bounded positive not all equal LE3-SAT). One advantage of the weaker definitions of these LE3-SAT variants is that the reductions showing NP-completeness become much simpler, and require a much smaller increase in the number of fresh variables and the number of auxiliary clauses that must be introduced. For example, converting the SAT instance expression $(a, \bar{b})$ to an instance of Positive NAE 3-SAT, using the methods of [9] results in an expression with the 20 clauses and 22 variables listed below.

$$(a, b_1, d_1), (d, c, f), (a, b_1, e_1), (e, c_1, f)$$
$$(b, b_1, t_b), (b, b_1, u_b), (b, b_1, v_b), (t_b, u_b, v_b),$$
$$(c, c_1, t_c), (c, c_1, u_c), (c, c_1, v_c), (t_c, u_c, v_c),$$
$$(d, d_1, t_d), (d, d_1, u_d), (d, d_1, v_d), (t_d, u_d, v_d),$$
$$(e, e_1, t_e), (e, e_1, u_e), (e, e_1, v_e), (t_e, u_e, v_e)$$

In these clauses, the four fresh variables b_1, c_1, d_1, and e_1 were introduced to play the roles of complements for b, c, d, and e, respectively, and the various twelve subscripted t, u, and v variables were introduced to enforce those relationships. Using the reduction in [10] leads to an even larger expression with 280 clauses and 216 variables. By contrast, converting it to a PNAE LE3-SAT instance, namely $(a, c, d), (b, c)$, requires only two clauses and four variables.

As can be seen from this example, the weaker restriction makes it easy to remove negations by introducing fresh variables that serve as negations.

2 NP-completeness

The LE3-SAT problem is easily seen to be NP-complete by a reduction from SAT.

Theorem 2.1 *The LE3-SAT problem is NP-complete.*

Proof. The proof is by reduction from SAT. It is a trivial modification of the classical reduction from SAT to 3-SAT (see [10]). Namely, clauses that have size exceeding two are reduced in the classical way[10], while clauses of size one or two are left unchanged. ■

The NAE LE3-SAT problem can now be shown to be NP-complete by a reduction from LE3-SAT.

Theorem 2.2 *The NAE LE3-SAT problem is NP-complete.*

Proof. The proof is by reduction from LE3-SAT. Following the reduction methods found in [9] for reducing 3-SAT to NAE 3-SAT, we introduce a fresh variable d for the purpose of "breaking the symmetry" inherent in the NAE variant. Now, each singleton clause (a) can be replaced by (a, d), each two-variable clause (a, b) can be replaced by (a, b, d), and each 3-variable clause (a, b, c) can be replaced by a pair of clauses $(a, b, \bar{z}_i)$ and (z_i, c, d), where z_i is a fresh variable introduced just for this clause. That is, if there are m three-variable clauses $C_1, C_2, ..., C_m$, there will be m fresh variables $z_1, z_2, ..., z_m$ introduced in the construction, one for each C_i. The arguments in [9] for the 3-SAT reduction extend naturally to the LE3-SAT reduction. In particular, the arguments for one-variable and two-variable clause satisfaction are natural extensions of the argument in [9] for the three-variable clause satisfaction. We will include the argument here for the convenience of the reader.

Suppose that we start with an LE3-SAT expression E and use the construction to create an NAE LE3-SAT expression E'. First we will show that if E' has an NAE satisfying assignment, then so does E. If E' has an NAE satisfying assignment, then we can assume by symmetry that in that assignment, the symmetry breaking variable d is set to FALSE (otherwise, invert the assignment of all the variables). Then, for every clause (a, d) in E', a must hold the value TRUE, which will make the corresponding clause (a) true in E. Similarly, for every clause (a, b, d) in E', a and b cannot both be set to FALSE, which will make the corresponding clause (a, b) true in E. Finally, for every pair of clauses $(a, b, \bar{z}_i)$ and (z_i, c, d) in E', a, b, and c cannot all be set to FALSE, which will make the corresponding clause (a, b, c) true in E. Conversely, suppose that E has a satisfying assignment. The first step will be to set $d =$ FALSE. Then for every clause (a) in E, a must hold the value TRUE, in which case the corresponding clause (a, d) is NAE satisfied in E'. Similarly, for every clause (a, b) in E, either a or b must hold the value TRUE, in which case the corresponding clause (a, b, d) is NAE satisfied in E'. Finally, for every clause (a, b, c) in E, either a or b or c must hold the value TRUE, in which case the two corresponding clauses $(a, b, \bar{z}_i)$ and (z_i, c, d) can both be NAE satisfied by setting $z_i = a \vee b$, where $\vee$ represents logical disjunction. ■

The positive (monotone) NAE LE3-SAT (PNAE LE3-SAT) is now shown to be NP-complete, by a reduction from NAE LE3-SAT.

Theorem 2.3 *The Positive NAE LE3-SAT (PNAE LE3-SAT) problem is NP-complete.*

Proof. The proof is by reduction from NAE LE3-SAT. All that is needed here is to replace every negated variable $\bar{a}$ by a fresh variable b, and add a new clause (a, b), to enforce the complement relationship. ■

Finally, the 3-Bounded Positive NAE LE3-SAT (3B-PNAE LE3-SAT) is NP-complete, by a reduction from PNAE LE3-SAT.

Theorem 2.4 *The 3-Bounded Positive NAE LE3-SAT problem (3B-PNAE LE3-SAT) is NP-complete.*

Proof. The proof is by reduction from PNAE LE3-SAT. For each variable a that occurs k times, where $k > 3$, introduce $2(k-1)$ fresh variables $a_1, a_2, ..., a_{k-1}, c_1, c_2, ..., c_{k-1}$. Replace each occurence of a (except the last one) with the corresponding fresh variable a_i, and for each $i < k-1$, add two new clauses (a_i, c_i) and (c_i, a_{i+1}), and finally add clauses (a_{k-1}, c_{k-1}) and (c_{k-1}, a). Then each a_i is in no more than three clauses, a is in exactly two clauses, and each c_i is in exactly two clauses. This process will give a 3B-PNAE LE3-SAT instance that is satisfiable if and only if the original is NAE satisfiable, since the various a_i variables are all logical aliases for variable a. The expansion in the number of variables and clauses in the new expression is less than twice the sum over all variables a of the number of clauses containing a. Since the sum over all variables a of the number of clauses containing a is bounded by $3C$, where C is the number of clauses, the total expansion (in both variables and clauses) cannot exceed twice the number of clauses. Therefore, the reduction is linear, and thus polynomial, in both time and space. ■

Note that the construction of Theorem 2.4 can also be applied to a variable that occurs exactly three times, but the result would include variables that occur exactly three times, so no reduction in the maximum number of occurrences would be accomplished by the construction. Therefore this method cannot be used to show that 2B-PNAE is NP-complete.

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

The representation of an instance of PNAE LE3-SAT consists of a finite set of variables and a finite set of clauses C. Each clause in C contains the disjunction of at most three variables 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, at least one of the variables is assigned to TRUE, and in each clause with more than one variable, at least one is assigned to be FALSE. In an instance of 2B-PNAE LE3-SAT, one more restriction is imposed: that no variable appears in more than two clauses. Such an instance can be represented by a graph G (see also [7] and [4]), 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 $\{(a,b,c),(a,d,e),(b,f,g),(c,d,f),(e,h,i),(h,j,k),(i,k,m),(j,m,p),(g,n,p)\}$ would be represented by the graph in Figure 1. Note that variable n does not appear as an edge in the graph since it is a disjunct in only one clause.

In view of this correspondence, we will use the terms vertex and clause interchangeably, and also the terms edge and variable. One possible NAE assignment for the expression of

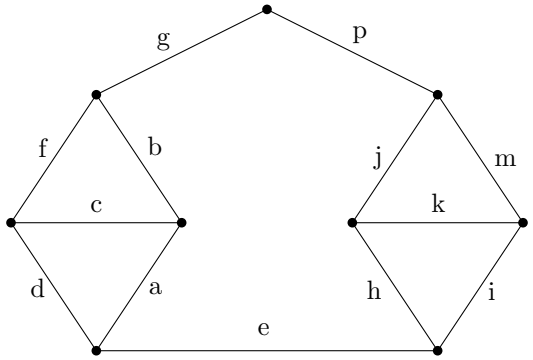


Figure 1: A clausal graph representation of an instance of NAE LE3-SAT.

Figure 1 assigns variables d, b, h, m, f, i and n to TRUE, and all others to FALSE. Note that the expression might have many different assignments conforming to the NAE property; for example, changing the values of either f or i (or both) in the above assignment would still yield an acceptable assignment.

4 Simplifying Assumptions for the Clausal Graph Representation

In this section, we will make several simplifying assumptions about the permissible Boolean expressions and graph representations, in order to allow us to apply adaptations of two powerful graph theoretic results.

First, we will assume that all clauses must contain at least two distinct variables, so that clauses such as (a, a, a) or (a, a) or (a) do not occur. Clearly no NAE LE3-SAT assigment could satisfy any of these clauses, since at least one variable must be TRUE and at least one variable must be FALSE. Hence, any expression containing any such clause would be immediately unsatisfiable.

Without loss of generality, we assume that each variable appears in a clause at most once, eliminating clauses such as (a, a, b) (we have already handled the case of (a, a)). Since at least one variable must be TRUE and at least one variable must be FALSE in any satisfying NAE LE3-SAT assignment, either a must be set to TRUE and b to FALSE, or a to FALSE and b to TRUE, which is identical to the satisfying assignments for (a, b). So we will replace any clause of the form (a, a, b) by the equivalent (a, b). We also assume without loss of generality that we have at most one copy of any clause, after all simplifications have been made.

Next we consider the graph structure that arises from clauses, subject to our simplifications. Since each clause can contain at most one copy of a variable, our graph representation contains no loops. Likewise, since a clause cannot duplicate a variable, and each variable occurs in at most two clauses, there is at most one edge per variable, since edges join clauses containing

the variable. Hence our graph representation contains no multiple edges (edges sharing the same variable label). Note that variables which occur in only one clause will not have a corresponding edge in the graph representation.

However, it is permissible to have parallel edges. For example, suppose that a pair of variables, say a and b, appear together in a pair of clauses, such as in (a, b, c) and (a, b, d). Between this pair of nodes (clauses) would be two edges, one labeled a and one labeled b. In the search for a satisfying assignment, if a and b are assigned the same value, then c and d must both have the opposite value (and thus will be the same). However, if a and b are assigned different values, then, subject to the restrictions imposed by other clauses, c and d could be assigned either the same values or different values.

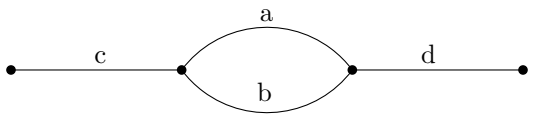


Figure 2: Parallel edges a and b give vertices from left to right representing the four clauses $(c, e), (a, b, c), (a, b, d)$ and (d, f).

Combining these simplifying assumptions will allow us to adapt ideas from the theory of graphs to provide a polynomial algorithm for 2B-PNAE LE3-SAT. To this end, we define a simplified instance of 2B-PNAE LE3-SAT to be one with all the simplifying assumptions described above. These assumptions imply that, in the corresponding graph, there are no loop edges and no multiple edges, though parallel edges may exist. It is important to note that each of these simplifications can be carried out in polynomial time. Likewise, if a NAE 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. Note that such extension may not be unique, but it will always produce a satisfying assignment. In subsequent sections, we assume that we have a simplified instance of 2B-PNAE LE3-SAT, unless specified otherwise.

5 Optimum Linking in a Clausal Graph

In order to find a satisfying NAE assignment, we seek a set of variables that may be assigned to be TRUE in such a way that the NAE property is realized. In the graph representation these variables will correspond to a set of edges which we call a *linking* (whose properties we will formalize later), and we set the variables associated with these edges to be TRUE. In Theorem 5.4 we show a correspondence between linkings with certain properties and solutions of a 2B-PNAE LE3-SAT instance, and then use properties of a linking to characterize such solutions. We also give necessary and sufficient conditions that allow us to construct a maximum linking in a clausal graph. Such a linking will provide a satisfying assignment for the corresponding 2B-PNAE LE3-SAT instance if and only if the linking covers all *crucial* vertices, which we will now define.

Definition 5.1 *A **crucial vertex** is a vertex whose degree in the graph representation equals the number of (distinct) variables in the clause.*

Note that all vertices of degree three must be crucial, and no vertices of degree one are crucial.

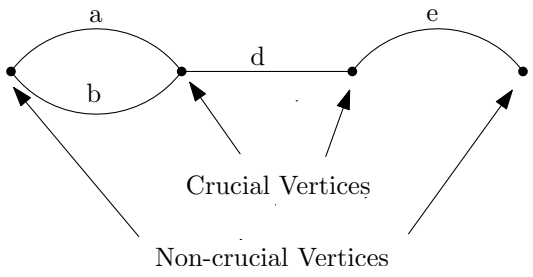


Figure 3: This graph represents the clauses (a, b, c), (a, b, d), (d, e), and (e, f). Since c and f each appear in only one clause, they are not depicted, and (a, b, c) and (e, f) are not crucial vertices.

Definition 5.2 *A **linking** in a graph $G = (V, E)$ is a set of edges $L \subset E$ such that*

- *Each crucial vertex of degree 2 in the graph is incident to at most one edge in L, and*
- *Each vertex of degree 3 in the graph is incident to at most two edges in L.*

Figure 4 illustrates these possible configurations. Necessarily a linking is a set of paths and cycles, as no crucial vertex can be incident with more than two linking edges, and no non-crucial vertex can have degree more than 2. Trivially, the empty set is a linking. We will show that a satisfying assignment can be found for an instance of 2B-PNAE LE3-SAT whenever the clausal graph admits a linking which covers all crucial vertices. We will then give a polynomial algorithm for finding such a linking whenever one exists.

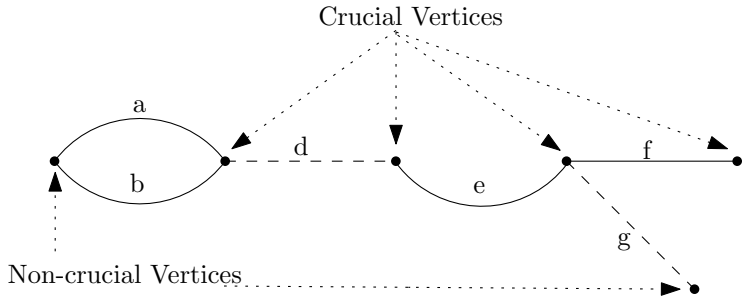


Figure 4: This graph represents the clauses (a, b, c), (b, c, d), (d, e), (e, f, g), (f) and (g, h). The set of edges $\{a, b, e, f\}$ form an optimum linking (see Definition 5.5), though other optimum linkings, such as $\{d, f\}$, are possible.

Definition 5.3 *A linking L* ***covers*** *a vertex v if v is incident with one or more of the edges of L.*

Theorem 5.4 *A simplified* 2B-PNAE LE3-SAT *instance has a solution if and only if there is a linking L in the corresponding clausal graph G such that L covers all the crucial vertices of G.*

Proof. Suppose there is a satisfying assignment for the 2B-PNAE LE3-SAT instance. Then those variables that are assigned to TRUE and that appear in two clauses form a linking in the graph G, since no clause with one or two variables can have more than one variable set to TRUE and no clause with three variables can have more than two set to TRUE in the satisfying assignment. Furthermore, since at least one variable in every clause must be set to TRUE, every crucial vertex will be covered by an edge of this linking.

Conversely, suppose that there is such a linking L covering all the crucial vertices of G. We begin by assigning every variable corresponding to an edge of L to be TRUE. Then the assignment can be completed because the only remaining uncovered vertices are non-crucial vertices, those with at least one variable that appears in no other clauses. Each of these variables occurring in only one clause can be assigned based on the other variables in the clause, if any. In particular, assign such a variable to TRUE if no other variable in the clause is assigned to true, and to FALSE otherwise. While such an assignment may not be the only satisfying assignment consistent with the linking, it provides one such assignment. ■

Definition 5.5 *A linking L is* ***optimum*** *if no other linking covers more crucial vertices than L.*

Definition 5.6 *An* ***alternating path*** *for a linking L is a path in which each internal vertex is incident with exactly one edge of L on the path.*

Definition 5.7 *An* ***augmenting path*** *for a linking L is an alternating path P for which one endpoint of the path is a crucial vertex u not covered by L and the other endpoint is a crucial vertex that is either not covered by L or is covered by exactly two edges of L: one that is a terminal edge of P and another that is not in P.*

Definition 5.8 *An* ***alternating cycle*** *for a linking L is a cycle C in which each vertex is incident with exactly one edge of $L \cap C$, with the exception of one designated vertex j, which is incident to either zero or two edges of $L \cap C$.*

Definition 5.9 *An* ***augmenting cycle*** *for a linking L is an alternating cycle C whose designated exceptional vertex j is an uncovered crucial vertex of degree 3. Note that this is the case in which j is incident to exactly zero edges of $L \cap C$, and in addition, since j has degree three, there is a third edge E incident to j, and E is not in L.*

Definition 5.10 *An* ***augmenting kite*** *K for a linking L is an alternating cycle C, together with an alternating path joining some uncovered crucial vertex u to the designated exceptional vertex j of C.*

Note the important distinction from an alternating path with respect to matchings. An internal vertex may be incident with multiple edges of linking L; however, only one edge of L intersected with the path can be incident to an internal vertex.

The well-known maximum matching theorem of Berge [1], which is utilized by Edmonds's algorithm, readily adapts from a statement about a maximum matching to a statement about an optimum linking.

Theorem 5.11 *A linking L is optimum if and only if there is no augmenting path P for L, and no augmenting cycle C for L, and no augmenting kite for K for L.*

Proof. First, consider the contrapositive of the forward direction: suppose there exists for L any one of the augmenting objects P, C, or K. Then we can modify the linking L by reversing every edge of the augmenting object to obtain a new linking L'. In each case (path, cycle, and kite), it is easy to check that L' covers all the crucial vertices covered by L, plus at least one more, either node u or node j. Therefore, L is not optimum.

Conversely, suppose that L is not optimum. Then there is a linking L' covering more crucial vertices than L. We want to be able to construct an augmenting object for L. Consider the symmetric difference $L \oplus L'$. For ease of exposition, we will think of the edges of $L - L'$ as being "solid", and those of $L' - L$ as being "dashed", and we will speak of these as being of different "color". Let u be a crucial vertex that is covered by at least one edge of L' but not by any edge of L. Let A be the component of $L \oplus L'$ that contains u. Vertex u is covered by at most two dashed edges since L' is a linking, and by no solid edges. We now proceed by cases.

Case 1: Suppose that u is covered by only one dashed edge $e_1 = (u, v)$. That is, the remaining edges incident to u are neither in L nor in L'. If vertex v is adjacent to another dashed edge, say e_2, then the path P consisting of the single edge e_1 will serve as an augmenting path for L. Otherwise, vertex v is only adjacent to solid edges at level two of a breadth first search (BFS) of A emanating from u. Similarly, if any of the endpoints $w \neq v$ of one of those solid edges e_2 at the second level in the search is adjacent to a solid edge at level three of the search, then the path $P = (e_1, e_2)$ will serve as an augmenting path (see Figure 5). Similarly, if any of the dashed edges at the third level is adjacent to a dashed edge at the fourth level (as, for example, with edge e_3), there will be an augmenting path. Proceeding inductively in this way, we will either find an augmenting path, or we will encounter a sequence of levels of alternating color. Also note that at any level, if two edges of the same color meet in a vertex at the lower level (as, for example, with edge e_4 or edge e_5), an augmenting path will be found.

At each level of the search, if any two nodes are connected by an edge e of the "next" color, then a kite can be constructed (see Figure 6) by joining edge e with the union of the two alternating paths back upwards to u. Note that these two paths will meet at a lowest point j of some previous level, which will serve as the exceptional vertex of the augmenting kite.

If no such edge e is encountered in the search, then the component A is a tree, rooted at u, with alternating colors at each level of the tree. This tree must have at least one leaf node

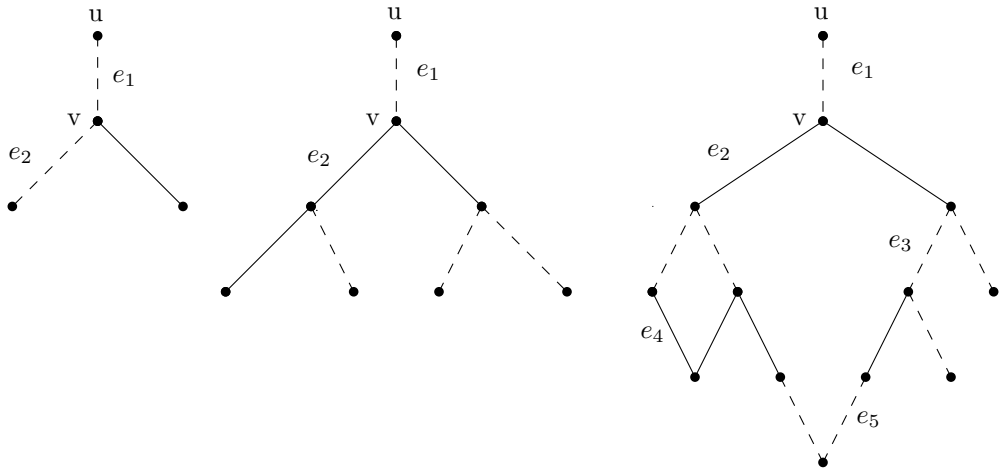


Figure 5: Augmenting paths in the BFS.

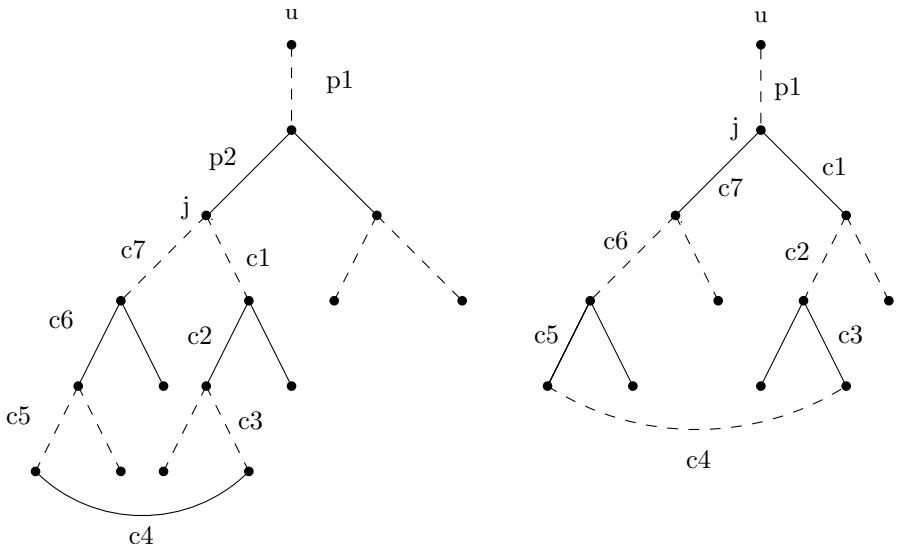


Figure 6: Augmenting kites in the BFS.

x, which is incident to only one edge e' of A. If x is not a crucial vertex, then the path in A from x back to u is an augmenting path, since it is not necessary to cover x. Otherwise, if e' is a dashed edge, we will also have an augmenting path, since both u and x are crucial. Finally, if x is crucial and e' is a solid edge (a member of $L - L'$), then there must be an edge e'' (not in A) incident to x, such that e'' is in L'. If there is no such edge, then $e' \cup L'$ would be a linking covering more crucial vertices (since x is crucial) than L', contradicting the optimum property of L'. Therefore in this final case we also have an augmenting path in A from u down to x (see Figure 7), which completes the proof of Case 1.

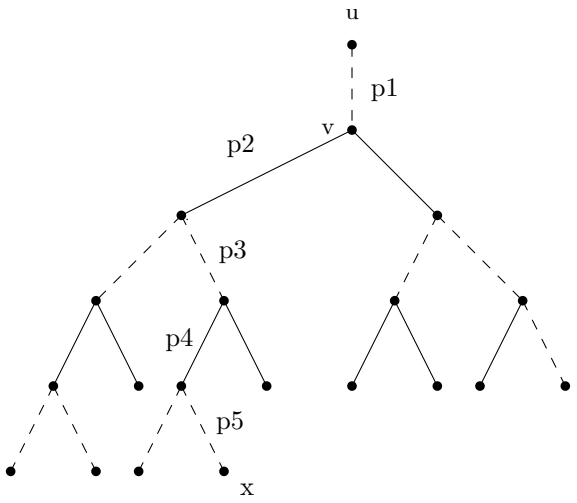


Figure 7: Augmenting path to another uncovered vertex.

Case 2: Suppose that u is covered by two dashed edges. Then proceed as in Case 1, with a breadth first search of A emanating from u. The details of this case are identical to those of Case 1, with the following exception. When two nodes at the same level are connected by an edge of the "next" color, the two paths from those nodes back up to the root node u might not meet until they get all the back up to u, in which case we will have an augmenting cycle, rather than an augmenting kite. However, if the two paths do meet in a point j lower than u, then we will have an augmenting kite as before (see Figure 8).

Note that there is no need for a case in which vertex u is covered by three dashed edges, since the dashed edges are members of linking L', and no vertex can be adjacent to three edges of a linking. ■

6 Finding Optimum Linkings by Augmentation

In light of Theorem 5.11, we seek to specify a polynomial algorithm that, given a linking L, is guaranteed to find an augmentation (path, cycle, or kite), if one exists. The algorithm will be simply:

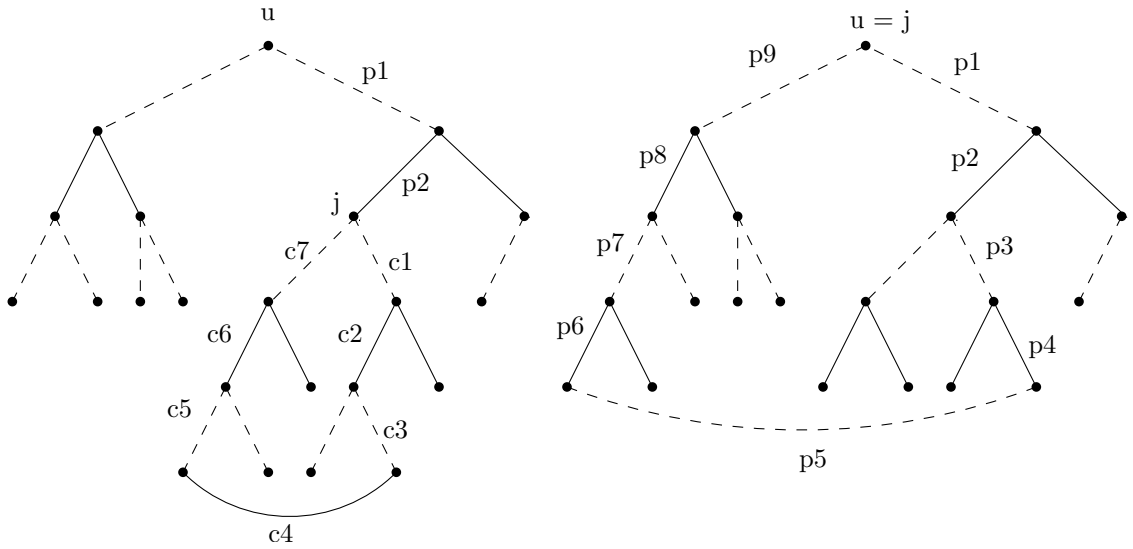


Figure 8: Augmenting kite and cycle.

Algorithm 1 : Finding an augmentation, if one exists

```
changed ← true
while (there is an uncovered crucial vertex and changed) do
  changed ← false
  unmark all uncovered crucial vertices
  while (there is an unmarked uncovered crucial vertex u and (not changed)) do
    perform a search rooted at u, seeking an augmentation
    if (no augmentation is found rooted at u) then
      mark u
    else
      apply the augmentation
      changed ← true
    end if
  end while
end while
```

When this cycle terminates there are no augmentations remaining, so the linking L is now optimum, so it suffices to now check whether any crucial vertices remain uncovered. If not, the corresponding expression is NAE satisfiable by Theorem 5.4. Otherwise the expression is not NAE satisfiable. Given this algorithm, it remains to provide a subroutine for seeking an augmentation rooted at any given uncovered crucial vertex u, in a way that is guaranteed to find an augmentation whenever one exists. The description of this subroutine and it's proof of correctness require a new definition.

Definition 6.1 *An **directional BFS (DBFS)** rooted at a node u in the direction of some subset $\{e_1, ..., e_k\}$ of the edges adjacent to u, is a search in which the first step is taken only in the directions determined by the set of edges $\{e_1, ..., e_k\}$. After that initial step, the usual BFS algorithm takes over to complete the search.*

Algorithm 2 : Seeking an augmentation at uncovered crucial vertex u

if u is adjacent to only two edges, e_1 and e_2 **then**
  perform a DBFS rooted at u, in direction $\{e_1\}$, looking for augmenting path or kite
  perform a DBFS rooted at u, in direction $\{e_2\}$, looking for augmenting path or kite
else {u is adjacent to three edges, e_1, e_2, and e_3}
  perform a DBFS rooted at u, in direction $\{e_1\}$, look for augmenting path or kite
  perform a DBFS rooted at u, in direction $\{e_2\}$, look for augmenting path or kite
  perform a DBFS rooted at u, in direction $\{e_3\}$, look for augmenting path or kite
  perform a DBFS rooted at u, in direction $\{e_1, e_2\}$, look for augmenting kite or cycle
  perform a DBFS rooted at u, in direction $\{e_1, e_3\}$, look for augmenting kite or cycle
  perform a DBFS rooted at u, in direction $\{e_2, e_3\}$, look for augmenting kite or cycle
end if

A correctness argument for this algorithm may proceed by examing two cases:

Case 1: If u is incident to exactly two edges e_1 and e_2, perform a separate DBFS rooted at u in each of the two directions $\{e_1\}$ and $\{e_2\}$. In each DBFS proceed as in proof of Theorem 5.11. If there are any augmentations including u, each must be either an augmenting path or an augmenting kite beginning at u, and including exactly one of e_1 or e_2. The process specified in Case 1 of the proof of Theorem 5.11 must reveal one such augmentation, if one exists(see Figures 5, 6, and 7).

Case 2: If u is adjacent to three edges e_1, e_2, and e_3, perform a separate DBFS rooted at u for each of the three directions $\{e_1\}$, $\{e_2\}$, and $\{e_3\}$, proceeding as in Case 1 of the proof of Theorem 5.11. If there are any augmentations including u, each must be either an augmenting path or an augmenting kite beginning at u, including exactly one of e_1, e_2, or e_3. Then, for every pair $\{e_i, e_j\} \subset \{e_1, e_2, e_3\}$, perform a DBFS simultaneously in the direction of $\{e_i, e_j\}$, proceeding as in Case 2 of of the proof of Theorem 5.11. These searches will seek either an augmenting path including exactly one of e_i, e_j, or an augmenting kite including exactly one of e_i, e_j, or an augmenting cycle including both e_i and e_j. Here, the process specified together in the two cases of the proof of Theorem 5.11 must reveal one such augmentation,

if one exists (see Figures 5, 6, 7, and 8). Furthermore, this algorithm is clearly polynomial by a standard implementation of breadth first search.

The **directional** modification to the standard BFS is a minor change, and does not change the overall complexity of either storage or execution time. Both are still proportional to the number of vertices in the hypergraph.

7 Possibilities for Further Exploration

An extension of the polynomial reduction in Section 2 above to the case $k = 3$ would imply P=NP, as would an extension of the polynomial algorithm of Section 6 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.

References

[1] Claude Berge. Two Theorems in Graph Theory. In Proceedings of the National Academy of Sciences USA, Volume 43, Issue 9, September 1957, 842-844.

[2] Piotr Berman, Marke Karpinski, and Alexander D. Scott. Computational complexity of some restricted instances of 3-SAT. In Discrete Applied Mathematics, Volume 155, Issue 5, March 2007, 649-653.

[3] Stephen Cook. The Complexity of Theorem Proving Procedures. In Proceedings of the 3rd annual ACM Symposium on Theory of Computing, 151-158, 1971.

[4] Richard Denman and Stephen Foster. Using Clausal Graphs to Determine the Computational Complexity of k-Bounded Positive One-in-Three SAT. In Discrete Applied Mathematics Volume 157, Issue 7, April 2009, 1655-1659.

[5] Jack Edmonds. Paths, trees, and flowers. Canadian Journal of Mathematics, Volume 11, 1965, 449-467.

[6] M.R. Garey and D.S. Johnson. Computers and Intractability. W. H. Freeman, New York. 1979.

[7] J. Kratochvil. Complexity of Hypergraph Coloring and Seidel's Switching, Lecture Notes in Computer Science, Volume 2880, 2003, 297-308.

[8] Leonid Levin. Universal'nye perebornye zadachi. Problemy Peredachi Informatsii 9 (3): 265-266, 1973.

[9] C. Moore and S. Mertens. The Nature of Computation, Oxford University Press, ISBN 978-0-19-923321-2, 2009.

[10] Bernard M. Moret. The Theory of Computation. Addison Wesley, ISBN 0-201-25828-4, 1998.

[11] T. J. Schaefer. The complexity of satisfiability problems. In Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 216-226, 1978.

[12] Craig A. Tovey. A simplified NP-complete satisfiability problem. In Discrete Applied Mathematics, Volume 8, Issue 1, April 1984, 85-89.