Determining satisfiability of 3-SAT in polynomial time

Ortho Flint, Asanka Wickramasinghe, Jay Brasse, Chris Fowler

Abstract

In this paper, we provide a polynomial time (and space), algorithm that determines satisfiability of 3-SAT. The complexity analysis for the algorithm takes into account no efficiency and yet provides a low enough bound, that efficient versions are practical with respect to today's hardware. We accompany this paper with a serial version of the algorithm without non-trivial efficiencies (link: polynomial3sat.org).

1 Introduction

The Boolean satisfiability problem (or SAT), was the first problem to be proven NP-complete by Cook in 1971 [1]. 3-SAT is one of Karp's original 21 NP-complete problems [2]. At last count, there are over $3,000$ NP-complete problems considered to be important. Any problem in NP can be presented as a SAT problem. A SAT problem can be transformed into a 3-SAT problem, where any increase in the number of clauses is a polynomial bound. We provide a deterministic polynomial time algorithm that decides satisfiability of 3-SAT. We also provide in Big O notation the complexity of the algorithm without efficiencies. The original serial version is called *naive*, by which we mean, none of the efficiencies that could be used for dramatic reductions in work are present. However, this version is quite useful for research, as we can attest. There are many non-trivial efficiencies that can be incorporated.

Note well, that the intent of an informal presentation of the algorithm, is so that many outside the math/computer science community might attempt a

read. For some of the mathematical objects used, we give a colloquial name, in hope of creating conceptual permanence.

2 Preliminaries and Definitions

Definition 2.1. *A 3-SAT is a collection of literals or variables (usually represented by integers), in groups called clauses where no clause has more than 3 literals, and at least one clause does have 3 literals. If it's possible to select exactly one literal from each clause such that no literal l, and its negation $-l$ (or denoted $\neg l$, meaning not l), appear in the collection of chosen literals, we say that the 3-SAT is satisfiable, otherwise we say it's unsatisfiable. For satisfiability, the collection of literals chosen is called a solution. Note that the size of a solution set is smaller than the size of the collection, if the collection had at least two clauses of which the same literal was chosen.*

It is important to note, that our definition of a solution (definition 2.10), is inextricably tied to our constructs for a collection of clauses.

When we think of literals (also called *atoms*), we can consider an edge joining two vertices, each with an associated literal, if and only if, it's not a literal and its negation. Under no circumstance would a literal and its negation be connected by an edge. There are also no edges between two literals from the same clause. So conceptually, there exist edges that connect every literal to every other literal with the two restrictions that were just stated. Then it follows, that a collection of literals for some solution is such that, a literal from each clause is connected to every other literal from that collection. In graph theory, such a graph is called a *complete* graph K_n, n being the number of vertices, which here it's also the number of clauses. We shall denote this graph as: K_C where c is the number of clauses.

Definition 2.2. *An edge-sequence is an ordered sequence with elements 1 and 0. The ordering is an ordering of the clauses, with indexing: C_1, C_2, C_3, ... , C_c where a corresponding C_i has its literals ordered the same way for each sequence constructed for a 3-SAT. An edge-sequence I, for an edge with endpoints labelled x and y, where $x \neq -y$, the literals associated with the endpoints, is denoted by $I_{x,y}$. The endpoints must always be from different*

clauses. We call the positions in $I_{x,y}$ that correspond to a clause C_i the cell C_i. The cells C_j and C_k containing the endpoints, x and y for $I_{x,y}$ have only one entry that is 1 in the positions associated to x and y. When an edge-sequence is constructed, a given position in $I_{x,y}$ is 1 if the associated literal is not $-x$ or $-y$. The initial construction of $I_{x,y}$ is subject to certain rules defined in 2.8 and 2.9, which may produce more zero entries. Lastly, removing one or more cells from $I_{x,y}$ is again a (sub) edge-sequence, denoted by $I_{x,y}{}^*$, if the cells containing the endpoints for $I_{x,y}$ remain.

Definition 2.3. *We call a literal x, an edge-pure literal, if its negation does not appear in an edge-sequence. ie. there are zero entries in the positions associated with literal $-x$ in the edge-sequence, or no $-x$ exists. Note that the literals associated to the endpoints are necessarily edge-pure literals.*

Definition 2.4. *We call a literal x, an edge-singleton, if it has only one position in an edge-sequence. We say that only one position exists in each endpoint cell which correspond with the endpoints. And we call a literal a singleton, if it appears in only one clause for a given collection of clauses.*

Definition 2.5. *A loner cell contains just one 1 entry for some literal. And a loner clause contains just one literal.*

Definition 2.6. *A vertex-sequence is an ordered sequence with elements 1 and 0. The ordering is an ordering of the clauses, with indexing: C_1, C_2, C_3, ... , C_c where a corresponding C_i has its literals ordered the same way for each sequence constructed for a 3-SAT. A vertex-sequence V, for a vertex associated with literal x, is denoted by V_x. We call the positions in V_x that correspond to a clause C_i the cell C_i. The cell C_j containing the vertex x for V_x has only one entry that is 1 in the position associated to x. When a vertex-sequence is constructed, a given position in V_x is 1 if the associated literal is not $-x$. The initial construction of V_x is subject to certain rules defined in 2.8 and 2.9, which may produce more zero entries. Removing one or more cells from V_x is again a (sub) vertex-sequence, denoted by $V_x{}^*$, if the cell containing x remains.*

Definition 2.7. *When an entry 1 in an edge-sequence or a vertex-sequence, becomes zero, we call it a bit-change. If a bit-change has occurred in an edge-sequence or a vertex-sequence, we say the sequence has been refined, or a refinement has occurred. A zero entry never becomes a 1 entry.*

It's worth noting here that if an edge-sequence $I_{a,b}$ has a zero entry in some position for a literal c, then there is no K_C, using literals a, b and c together. In fact, this is what a bit-change is documenting in an edge-sequence.

Definition 2.8. *The loner cell rule, LCR, is that no negation of a literal belonging to a loner cell can exist in an edge-sequence or vertex-sequence. If such a scenario exists in an edge-sequence $I_{x,y}$ or a vertex-sequence V_x where z is the loner cell literal, then all positions associated with literal $-z$ incur a bit-change. If this action of a bit-change for $-z$, creates another loner cell where the negation of the literal in the newly created loner cell is still present in $I_{x,y}$ or V_x the action of a bit-change for the negation is repeated. Hence, to be LCR compliant may be recursive, but all refinements are permanent for any edge or vertex sequence.*

LCR compliancy is determined for an edge-sequence $I_{x,y}$ or a vertex-sequence V_x if either $I_{x,y}$ or V_x is being constructed. LCR compliancy is determined after any intersection between two of more edge-sequences is performed. LCR compliancy is determined if an edge-sequence or vertex-sequence incurred any refinement.

Definition 2.9. *The K-rule is that no cell from an edge-sequence $I_{x,y}$ or a vertex-sequence V_x can have all zero entries. If this is the case, then $I_{x,y}$ or V_x, equals zero, and $I_{x,y}$ is removed from its S-set, or V_x is removed from the vertex-sequence table. Note that their respective removals, is a refinement.*

K-rule compliancy is determined for an edge-sequence $I_{x,y}$ or a vertex-sequence V_x, if either $I_{x,y}$ or V_x are being constructed. K-rule compliancy is determined after any intersection between two of more edge-sequences is performed. K-rule compliancy is determined if an edge-sequence or vertex-sequence incurred any refinement. And finally, the K-rule is violated if all the vertex-sequences associated with a clause, are zero. In such a case, it's reported that the 3-SAT is unsatisfiable.

Definition 2.10. *A solution for a collection of c clauses must have a corresponding collection of edge-sequences, for some K_C. More precisely, the intersection of all the edge-sequences together, for a K_C, does not equal zero. ie. A solution K_C exists if $\bigcap_{i,j} I_{i,j} \neq 0$, where i and j are every pair of endpoints from the collection of edge-sequences for a K_C.*

A K_P, $p < c$, exists if the set of all sub edge-sequences $\mathcal{P}$ for K_P are such that the intersection of $\mathcal{P}$ does not equal zero. ie. A K_P exists if $\bigcap_{i,j} {I_{i,j}}^* \neq 0$, where i and j are every pair of endpoints from the collection of sub edge-sequences for K_P. It is to be understood that an edge-sequence for a K_C or K_P, means the edge-sequence associated with the edge for a K_C or K_P.

Definition 2.11. *A S-set is a collection of edge-sequences whose endpoints are from two clauses, C_i and C_j where $i \neq j$. The number of constructed edge-sequences to be a S-set is $|C_i||C_j|$ minus the non edge-sequences of the form: $I_{l,-l}$. For 3-SAT, there can be at most 9 edge-sequences in a S-set.*

Before we work through an example, we must define what it means to take an intersection or union of two or more edge-sequences. No intersections or unions are taken with vertex-sequences.

Definition 2.12. *We take the intersection or union of two n length edge-sequences, A and B, by comparing position i of A and B, using the Boolean rules for intersections (denoted by $\cap$), and unions (denoted by $\cup$), for all positions, $i = 0, 1, 2, \ldots, n-1$.*

Recall that the entry for position i of A and B, is either 1 or 0.

Then, for an intersection, we have:

$1_A \cap 0_B = 0_A \cap 1_B = 0_A \cap 0_B = 0$. *And* $1_A \cap 1_B = 1$.

And for a union we have:

$1_A \cup 0_B = 0_A \cup 1_B = 1_A \cup 1_B = 1$. *And* $0_A \cup 0_B = 0$.

In general, an intersection or union between two or more edge-sequences, is a multi-edged sequence. There is one exception described in section 3, where an intersection or a union of intersections of edge-sequences, always produces an edge-sequence.

Example

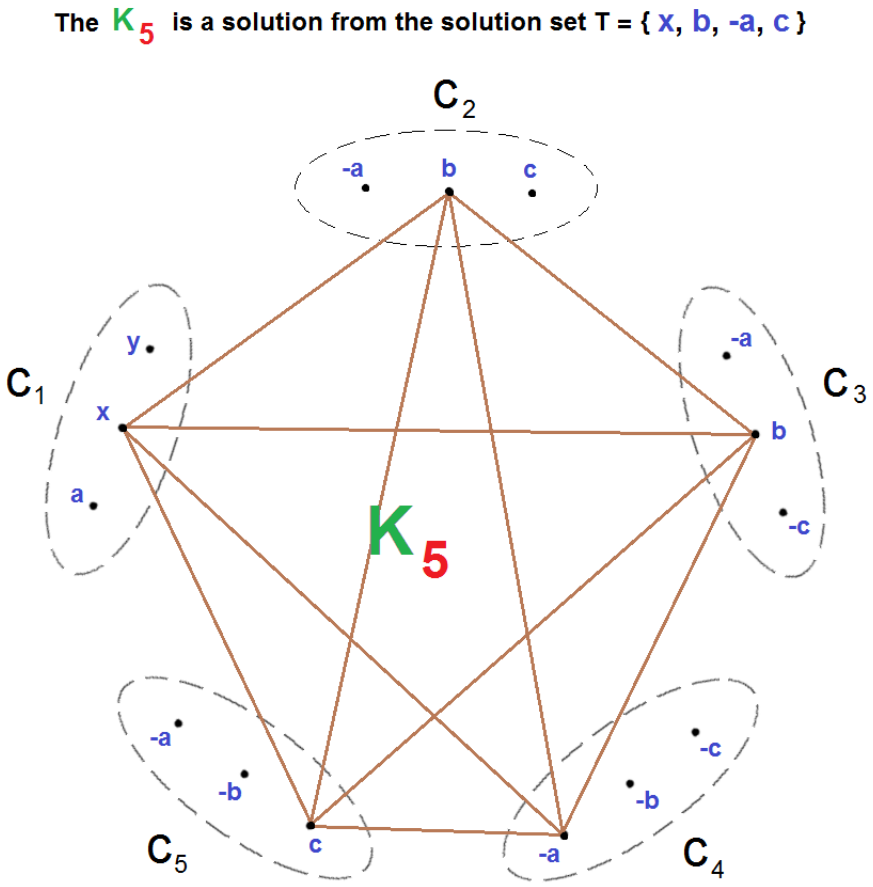


Figure 1: A 3-SAT with 5 clauses

We believe it is useful to think of 3-SATs in terms of their corresponding graphs. For example, Figure 1 (the only figure in the paper), depicts a 3-SAT with five clauses. The clauses are numbered and each clause has three literals. We say that a clause with three literals, thus three vertices, is *size* 3. For 3-SAT, clauses can only be of size 1, 2 or 3. Although there would be an edge between any two literals not belonging to the same clause and not between a literal and its negation, we have chosen to show just the edges for a K_5 from the solution set $T = \{x, b, -a, c\}$ in Figure 1.

Of course, the challenge when presented with a 3-SAT is to find at least one K_C, when it's a proper subgraph, or to determine that no K_C exists.

This example is to demonstrate the construction of edge-sequences for a S-set. We also provide one example of an intersection and another of a union between two edge-sequences.

For a S-set with edge-sequences formed using clauses C_i, C_j, we write: $S_{i,j}$. Recall, that an edge's endpoints are always labelled by their associated literals. Then, the edge-sequence I, for an edge with endpoints with labelling a, b, is denoted by $I_{a,b}$. Below, we add sub-subscripts for our example (Figure 1), 3-SAT's edge-sequences, to indicate which clauses the endpoints are from. And, within the (ordered) sequences, the subscripts indicate which literal received a zero or a one.

Our example has 10 S-sets and at most 9 edge-sequences in each. With respect to the sub-subscripts, all the edge-sequences are unique. We will show just the initial 8 edge-sequences (there is no $I_{a_1,-a_2}$, a literal and its negation), for $S_{1,2}$. We will also construct $I_{b_3,-c_4}$ from $S_{3,4}$ for the purpose of showing intersections and unions of edge-sequences.

Then, the constructed edge-sequences for $S_{1,2}$, before LCR and K-rule determination is:

$$\overbrace{}^{C_1} \quad \overbrace{\phantom{0_{-a}, 0_b, 1_c}}^{C_2} \quad \overbrace{\phantom{0_{-a}, 1_b, 1_{-c}}}^{C_3} \quad \overbrace{\phantom{0_{-a}, 0_{-b}, 1_{-c}}}^{C_4} \quad \overbrace{\phantom{0_{-a}, 0_{-b}, 1_c}}^{C_5}$$

$$I_{a_1,b_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 1_b, 0_c \mid 0_{-a}, 1_b, 1_{-c} \mid 0_{-a}, 0_{-b}, 1_{-c} \mid 0_{-a}, 0_{-b}, 1_c)$$

$$I_{a_1,c_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 0_b, 1_c \mid 0_{-a}, 1_b, 0_{-c} \mid 0_{-a}, 1_{-b}, 0_{-c} \mid 0_{-a}, 1_{-b}, 1_c)$$

$$I_{x_1,-a_2} : (0_a, 1_x, 0_y \mid 1_{-a}, 0_b, 0_c \mid 1_{-a}, 1_b, 1_{-c} \mid 1_{-a}, 1_{-b}, 1_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

$$I_{x_1,b_2} : (0_a, 1_x, 0_y \mid 0_{-a}, 1_b, 0_c \mid 1_{-a}, 1_b, 1_{-c} \mid 1_{-a}, 0_{-b}, 1_{-c} \mid 1_{-a}, 0_{-b}, 1_c)$$

$$I_{x_1,c_2} : (0_a, 1_x, 0_y \mid 0_{-a}, 0_b, 1_c \mid 1_{-a}, 1_b, 0_{-c} \mid 1_{-a}, 1_{-b}, 0_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

$$I_{y_1,-a_2} : (0_a, 0_x, 1_y \mid 1_{-a}, 0_b, 0_c \mid 1_{-a}, 1_b, 1_{-c} \mid 1_{-a}, 1_{-b}, 1_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

$$I_{y_1,b_2} : (0_a, 0_x, 1_y \mid 0_{-a}, 1_b, 0_c \mid 1_{-a}, 1_b, 1_{-c} \mid 1_{-a}, 0_{-b}, 1_{-c} \mid 1_{-a}, 0_{-b}, 1_c)$$

$$I_{y_1,c_2} : (0_a, 0_x, 1_y \mid 0_{-a}, 0_b, 1_c \mid 1_{-a}, 1_b, 0_{-c} \mid 1_{-a}, 1_{-b}, 0_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

Observe below, that after LCR and K-rule are applied, edges I_{a_1,b_2} and I_{a_1,c_2} will be zero. Since the loner cells C_4 and C_5 in I_{a_1,b_2} negate each other, applying LCR to either cell (we apply to first encountered), causes the other cell to be an empty cell which violates the K-rule. Similarly, the loner cells C_3 and C_4 in I_{a_1,c_2} negate each other, producing the same outcome.

Then, applying LCR to I_{a_1,b_2}, we have:

$$I_{a_1,b_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 1_b, 0_c \mid 0_{-a}, 1_b, 1_{-c} \mid \overbrace{0_{-a}, 0_{-b}, 1_{-c}}^{LCR} \mid \overbrace{0_{-a}, 0_{-b}, 0_c}^{All\ Zeroes})$$

Now, we apply K-rule to I_{a_1,b_2}, thereby $I_{a_1,b_2} = 0$:

$$I_{a_1,b_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 1_b, 0_c \mid 0_{-a}, 1_b, 1_{-c} \mid 0_{-a}, 0_{-b}, 1_{-c} \mid 0_{-a}, 0_{-b}, 0_c)$$

Applying LCR to I_{a_1,c_2}, we have:

$$I_{a_1,c_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 0_b, 1_c \mid \overbrace{0_{-a}, 1_b, 0_{-c}}^{LCR} \mid \overbrace{0_{-a}, 0_{-b}, 0_{-c}}^{All\ Zeroes} \mid 0_{-a}, 1_{-b}, 1_c)$$

Now, we apply K-rule to I_{a_1,c_2}, thereby $I_{a_1,c_2} = 0$:

$$I_{a_1,c_2} : (1_a, 0_x, 0_y \mid 0_{-a}, 0_b, 1_c \mid 0_{-a}, 1_b, 0_{-c} \mid 0_{-a}, 0_{-b}, 0_{-c} \mid 0_{-a}, 1_{-b}, 1_c)$$

The other 6 edge-sequences of $S_{1,2}$ listed above, are LCR and K-rule compliant. Thus, the construction of the edge-sequences resulted in only 6 edge-sequences for $S_{1,2}$. We note here that if a S-set had no edge-sequences after construction where LCR and K-rule was applied, it would mean there is no solution for the collection of clauses given. We show a K_5 in Figure 1, so all 10 S-sets must exist, because each S-set has an edge-sequence for a K_5.

Below, we show the result of doing $I_{x_1,c_2} \cap I_{b_3,-c_4}$ and $I_{x_1,c_2} \cup I_{b_3,-c_4}$.

The LCR and K-rule compliant edge-sequence $I_{b_3,-c_4}$ of $S_{3,4}$ is:

$$I_{b_3,-c_4} : (0_a, 1_x, 1_y \mid 1_{-a}, 1_b, 0_c \mid 0_{-a}, 1_b, 0_{-c} \mid 0_{-a}, 0_{-b}, 1_{-c} \mid 1_{-a}, 0_{-b}, 0_c)$$

And, the LCR and K-rule compliant edge-sequence I_{x_1,c_2} is:

$$I_{x_1,c_2} : (0_a, 1_x, 0_y \mid 0_{-a}, 0_b, 1_c \mid 1_{-a}, 1_b, 0_{-c} \mid 1_{-a}, 1_{-b}, 0_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

The intersection, $I_{x_1,c_2} \cap I_{b_3,-c_4}$ is :

$$(0_a, 1_x, 0_y \mid 0_{-a}, 0_b, 0_c \mid 0_{-a}, 1_b, 0_{-c} \mid 0_{-a}, 0_{-b}, 0_{-c} \mid 1_{-a}, 0_{-b}, 0_c)$$

Recall that we always determine LCR and K-rule compliancy after any intersection. Thus, the intersection: $I_{x_1,c_2} \cap I_{b_3,-c_4} = 0$, because at least one cell violated the K-rule. Here, both C_2 and C_4 violated the K-rule.

The LCR and K-rule compliant union: $I_{x_1,c_2} \cup I_{b_3,-c_4}$ is:

$$(0_a, 1_x, 1_y \mid 1_{-a}, 1_b, 1_c \mid 1_{-a}, 1_b, 0_{-c} \mid 1_{-a}, 1_{-b}, 1_{-c} \mid 1_{-a}, 1_{-b}, 1_c)$$

Observe that $I_{x_1,c_2} \cup I_{b_3,-c_4}$ is a multi-edged sequence. We remark that if a collection of edge-sequences are LCR and K-rule compliant, then their union must also be LCR and K-rule compliant, so compliancy is assured after a union is performed. The reason is simply that there is no union of LCR and K-rule compliant edge-sequences, that could create a new LCR scenario, or cause a violation of the K-rule, via the Boolean rules.

3 Description of the algorithm

In this section we describe the basic algorithm. The description of the algorithm will consist of describing pre-processing and how the ordered sequences are compared with each other and what actions are to be taken based on those comparisons. The section to follow proves the correctness of this scheme and that it stops in polynomial time, when applied to any instance of 3-SAT.

Pre-processing

Pre-processing begins by building the first tables from a given DIMACS file. While doing this, we learn of redundancies that might as well be eliminated. However, it's worth noting that the algorithm to process a 3-SAT does not require any redundancies to be removed.

Definition 3.1. *Let $\mathcal{H}$ be a given collection of clauses. We call a literal x, a pure literal, if its negation $-x$, does not appear in $\mathcal{H}$. ie. $\nexists -x$ in $\mathcal{H}$.*

Definition 3.2. *A clause of the form: $(l, -l, x)$ or $(l, -l)$ is called a quantum clause. A quantum clause is a possible randomly generated clause.*

To start, we order the literals represented by integers, within each clause. If there exists among the given clauses, two or more clauses which are the same clause, but the literals appear in a different order, it will be discovered. Only one copy of each unique clause is required to process a given 3-SAT. We learn if any clause has literal duplication: (a, a, a) or (a, b, b) or (a, a). Randomly generated clauses could be of this form, which we reduce to: (a), (a, b) and (a), respectively. When we have ordered the clauses, we have also documented each literal and its negation, and to which clauses they are associated. At this point we will have discovered any *pure* literals. The clauses containing one or more *pure* literals, can be removed, where one of the *pures*, represents its clause. We can remove a clause with a *pure* because any solution found with the remaining clauses, can include all the *pures*, since no solution includes their negations. We will also discover any *quantum* clauses, and they can be removed because any solution for the remaining clauses can always add a literal from a *quantum*. For example, let a *quantum* clause be

$q = (l, -l, x)$, and there is a solution with the remaining clauses which may use l or $-l$ or neither. For neither, l or $-l$ can be selected to be part of the solution.

Removing *pures* and *quantums* can be recursive, as it was for our example.

If there exists among the collection of clauses given, one or more clauses containing just one literal, we remove these clauses. Obviously, the single literal is the only literal to represent the clause for any solution. And, since these literals must be in every solution, their negations are removed from the collection of clauses given. This action may also be recursive. If this action empties a clause of all its literals, then there is no solution for the collection of clauses given.

There is a possible redundancy that we ignore, which we call *Global Inclusion*. Suppose among the given clauses, there exist two clauses: $C_i = (a, b, x)$ and $C_j = (a, b)$. It's clear that if a solution exists, then either literal a or b must be part of that solution, thus C_i is redundant and could be removed. With respect to *Global Inclusion*, in context of processing a 3-SAT, a cell C_k may no longer have 1 entries for every literal of clause C_k. For example, let clause C_k have three literals a, b and z. Now, suppose after some processing, that the entry for z in cell C_k is zero, in all edge-sequences. Then, cell C_i can be removed from all sequences. And, after all processing is complete, it can be determined if a or b will be representing clause C_i.

Construction of the edge and vertex sequences

Let $n \leq 3c$, where c is the number of remaining clauses and n is the sum of the sizes, for the c clauses. Then, after pre-processing, the n vertex-sequences, grouped by clause association, are constructed first, as outlined in definition 2.6, and then LCR and K-rule are applied. It would be common practice to order the clauses, and the literals within each clause, and use this same ordering for both the edge and vertex sequences.

After pre-processing, the remaining clauses not removed, have at most, 9 edge-sequences constructed from them pairwise. The edge-sequences are constructed as outlined in definition 2.2, and then LCR and K-rule are

applied. Observe that the number of edge-sequences would be less than $\binom{n}{2}$. It must always be less than, because we did not subtract the over count of non-existent edges with i) both endpoints in the same clause or ii) the non edge-sequences between a literal and its negation. Of course, if each clause had just one literal and there was a solution, pre-processing would have presented the solution or pre-processing removed all literals from at least one clause, establishing unsatisfiability. Either way, no edge-sequences would have been constructed. Each pair of clauses from the collection of c clauses, forms a S-set. Thus, the number of S-sets is $\binom{c}{2}$, where any S-set contains at most, 9 edge-sequences. We shall denote a S-set with the indices of the two clauses used to construct its edge-sequences. ie. $S_{i,j}$ has edge-sequences whose endpoints are from clauses C_i and C_j.

Before we describe in detail the **Comparing** of S-sets, we list the refinement rules. The refinement rules below, refer to a bit-change occurring, an edge removed from its S-set, or a literal having a zero entry in every edge and vertex sequence, all being the result of the Comparing of S-sets. Given a collection of clauses, if any of these refinements occurred that were not the result of the Comparing of S-sets, it was the result of LCR and K-rule compliancy, while constructing the edge-sequences and vertex-sequences. We shall apply the actions outlined in the refinement rules even when constructing the vertex and edge-sequences. The example demonstrated that certain edge-sequences were zero upon construction, as they failed LCR and K-rule compliancy. To be systematic, we would first construct all the vertex-sequences as described in definition 2.6. Then, apply LCR and K-rule to all of the vertex-sequences. If the actions outlined in the refinement rules can be taken on any vertex-sequence, we do so. Next, we construct all the edge-sequences as described in definition 2.2. Then, apply LCR and K-rule to all of the edge-sequences. If the actions outlined in the refinement rules can be taken on any vertex or edge sequence, we do so. This process may be recursive.

It is to be understood that an edge-sequence $I_{x,y}$ may be called just an edge for simplicity and that a vertex may be referred to by its associated literal.

The Refinement Rules $1, 2, 3$ and 4

When a refinement occurs and one of the rules $1, 2, 3$ or 4 actions are to be applied to an edge-sequence or a vertex-sequence, LCR and K-rule compliancy must always be determined afterward. Note that the actions for the refinement rules could be recursive and that all refinements are permanent.

Rule 1: a bit-change occurred

Let I_{a_i,b_j} where i denotes cell i, and j denotes cell j, be an edge-sequence where at least, one 1 entry for a position (occurrence) associated with literal c became zero. Then, we change 1 entries to zero for all positions associated with c in I_{a_i,b_j}. If there was just one position for c, then no action is taken.

Now, we consider all edges $I_{a,b}$ that are not I_{a_i,b_j}. ie. at least one endpoint is not from either cell i or j. We will change a 1 to zero for every occurrence of a literal who had at least one occurrence become zero in I_{a_i,b_j} for all edge-sequences labelled $I_{a,b}$. The result is all $I_{a,b}$ which of course includes I_{a_i,b_j} have the same set of literals whose corresponding entries are zero.

Rule 2: the follow up to a bit-change

If $I_{a,b}$ has zero entries for c, then $I_{a,c}$ will have zero entries for b, and $I_{b,c}$ will have zero entries for a. All three edge-sequences are documenting the same fact, namely, that there is no solution with a, b and c together. So, if an edge-sequence $I_{a,b}$ for which at least one entry became zero, say for literal c, we apply rule 1. Then, in $I_{b,c}$ and $I_{a,c}$ we make all 1 entries zero, that correspond to literals a and b respectively.

Rule 3: $I_{x_i,y_j} = 0$, and was removed from its S-set $S_{i,j}$

Let I_{x_i,y_j} be an edge that was removed from S-set, $S_{i,j}$. The sub-subscripts for the edge-sequence I_{x_i,y_j} indicate that one endpoint is from cell i, and the other endpoint is from cell j. First, we remove all occurrences of I_{x_m,y_n} where $m+n \neq i+j$, from their respective S-sets.

Next, we update all the vertex-sequences for literals x and y by: All the vertex-sequences for x have all positions associated with y incur a bit-change. And, all the vertex-sequences for y have all positions associated with x incur a bit-change.

Recall, that any kind of refinement requires testing LCR and K-rule compliancy, even on vertex-sequences. This in turn could cause the vertex-sequence to equal zero. In such a case, we apply all actions outlined in rule 4.

We also ensure that all refined vertex-sequences for a literal x, have the same set of literals whose corresponding entries are zero.

Now, all positions associated with y incur a bit-change, in $I_{x,\#}$ where # is any literal that forms an edge with x. Then, all positions associated with x incur a bit-change, in $I_{\%,y}$ where % is any literal that forms an edge with y.

This whole process may cause another edge removal, in which case we apply rule 3, recursively if need be. This in turn could cause a vertex-sequence to equal zero. In such a case, we apply all actions outlined in rule 4.

Rule 4: a literal's vertex-sequence equals zero

Let a literal x be such that its vertex-sequence violates the K-rule. Then its vertex-sequence V_{x_i}, where x belongs to cell C_i is equal to zero.

If V_{x_i} is now zero, then all V_x vertex-sequences are now evaluated to be zero, and all of them are removed from the vertex-sequence table.

When a vertex-sequence equals zero we do: All positions associated with x incurs a bit-change in every vertex and edge sequence. This action in turn causes all edge-sequences of the form: $I_{x,\#}$, where # is any literal that forms an edge with x, to equal zero.

This whole process may cause another edge-sequence (not of the form $I_{x,\#}$), to be removed, in which case we apply rule 3, recursively if need be. This in turn could cause a vertex-sequence to equal zero. In such a case, we apply all actions outlined here, in rule 4.

The Comparing of the S-sets

Essentially, the Comparing process is the algorithm. All data structures are simply updated based on the outcome of Comparing S-sets with one another.

Definition 3.3. *When every S-set has been Compared with every other S-set, we say that a **run** has been completed. If c clauses are considered, then there are $\binom{c}{2}$ S-sets, thus the number of S-set comparisons for a **run** is $\binom{\binom{c}{2}}{2}$ $< c^4$.*

Definition 3.4. *A **round** is completed if the Comparing process stops because no refinement occurred during an entire **run**. We say that the S-sets are **equivalent** when a round is completed.*

We note that if the first round was not completed, it was the case that the vertex-sequences associated to a clause, were evaluated to be zero, so they were discarded. This violation of the K-rule stops all processing, as there is no solution for the collection of clauses given.

To Compare, we take two S-sets and **determine** if an edge-sequence $I_{x,y}$ from one of the S-sets, can be refined by a union of the intersections between $I_{x,y}$ with each of the edge-sequences, from the other S-set. Either $I_{x,y}$ the edge-sequence under determination, is refined or it remains the same. This is done for each edge-sequence from both of the S-sets, in the same manner. As a matter of practice, we determine in turn, each edge-sequence from one S-set first, and then we determine in turn, each edge-sequence from the other S-set. Below, we construct two S-sets to describe in more detail all the steps to be taken.

Let the S-set: $S_{i,j}$ contain 9 edge-sequences with endpoints from clauses: $C_i = (1, 2, 3)$ and $C_j = (a, b, c)$ giving: $I_{1,a}$, $I_{1,b}$, $I_{1,c}$, $I_{2,a}$, $I_{2,b}$, $I_{2,c}$, $I_{3,a}$, $I_{3,b}$, $I_{3,c}$

Let the S-set: $S_{k,l}$ contain 9 edge-sequences with endpoints from clauses: $C_k = (4, 5, 6)$ and $C_l = (d, e, f)$ giving: $I_{4,d}$, $I_{4,e}$, $I_{4,f}$, $I_{5,d}$, $I_{5,e}$, $I_{5,f}$, $I_{6,d}$, $I_{6,e}$, $I_{6,f}$

If $S_{i,j}$ and $S_{k,l}$ have 9 edge-sequences each, then there were no negations between the literals in C_i and C_j, or between the literals in C_k and C_l, respectively.

We say that **determining** all edge-sequences from one S-set first, is doing one direction denoted by: $S_{k,l} \overset{1}{\rightharpoonup} S_{i,j}$. And, determining all edge-sequences once, for both S-sets, is doing both directions, denoted by: $S_{k,l} \overset{1}{\underset{2}{\rightleftharpoons}} S_{i,j}$

Suppose we **determine** $I_{4,d}$ of $S_{k,l}$ first. Then we have:

$$(I_{1,a} \cap I_{4,d}) \cup (I_{1,b} \cap I_{4,d}) \cup (I_{1,c} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{2,b} \cap I_{4,d}) \cup (I_{2,c} \cap I_{4,d}) \cup (I_{3,a} \cap I_{4,d}) \cup (I_{3,b} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) \leq I_{4,d}$$

The one efficiency present even in the original *naive* version with no refinement rules, was eliminating any unnecessary intersections by one easy check. After an edge-sequence is selected for determination, say I_{x_r,y_s} where $x_r \in C_r$, $y_s \in C_s$, the edge-sequences from the other S-set in the Comparing, that do not have 1 entries for the endpoints x_r and y_s when intersected with I_{x_r,y_s}, will be zero. Recall, that the two cells containing the endpoints, only have a single 1 entry corresponding to the two endpoints' positions, in their respective cells. Thus, if the other edge-sequence does not have a 1 entry for those same positions, the intersection will be zero, due to K-rule violation. So, after the selection of an edge-sequence to be determined, we select the edge-sequences from the other S-set, if they have 1 entries in both positions corresponding to the endpoints of I_{x_r,y_s}. Of course, we could also check to see if there are 1 entries in I_{x_r,y_s} corresponding to the endpoints of the other edge-sequence as well.

Let's suppose then, that every edge-sequence in $S_{i,j}$ above, did have 1 entries for both endpoints 4_k and d_l of $I_{4,d}$. Now suppose 6 intersections become zero, after the intersections were taken and LCR and K-rule was applied to each intersection, and we now have:

$$0 \cup (I_{1,b} \cap I_{4,d}) \cup 0 \cup (I_{2,a} \cap I_{4,d}) \cup 0 \cup 0 \cup 0 \cup 0 \cup (I_{3,c} \cap I_{4,d}) \leq I_{4,d}$$

which is equivalent to: $(I_{1,b} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) \leq I_{4,d}$

Now, we take their union. Of course, LCR and K-rule compliancy need not be checked for any union, since it would not have been possible to create a new loner cell scenario, nor a cell with all zero entries.

Then, to complete the determination of $I_{4,d}$ we need to compare position by position, to see if $I_{4,d}$ has been refined.

More precisely, if we have: $(I_{1,b} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) = I_{4,d}$, then $I_{4,d}$ is unchanged, and we move on to the next edge-sequence to be determined. Or instead, we have: $(I_{1,b} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) < I_{4,d}$, then $I_{4,d}$ has been refined. We need to know which literals incurred a bit-change, and then we apply the appropriate actions of the refinements rules, and as always, followed by testing LCR and K-rule compliancy.

To summarize, an edge-sequence $I_{4,d}$ to be **determined**, has 4 possible scenarios.

1) $I_{4,d} \neq 0$, and is unchanged.

2) $I_{4,d} \neq 0$, and is refined. We determine which literals had a bit-change and follow all appropriate refinement rule actions, recursively if need be.

3) $I_{4,d} = 0$, because $(I_{1,b} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) \leq I_{4,d}$ became zero after an appropriate refinement rule action was taken, and then LCR and K-rule was applied. In this case, edge-sequence $I_{4,d}$ is discarded, and the actions stated in refinement rule 3 are taken, recursively if need be.

4) If each intersection: $(I_{1,b} \cap I_{4,d})$, $(I_{2,a} \cap I_{4,d})$ and $(I_{3,c} \cap I_{4,d})$ had also been zero at the outset, then $I_{4,d} = 0$. And, as with 3), $I_{4,d}$ is discarded and the actions stated in refinement rule 3 are taken, recursively if need be. When an edge-sequence equals zero, it was the case that none of the edge-sequences from a S-set could be part of a solution with the edge-sequence that was being determined.

There are two observations worth noting with respect to the Comparing process. First, that the result of an intersection with just the edge-sequence to be determined, or the union of the intersections between the edge-sequence to be determined, with each of the allowed edge-sequences, from the other S-set, is never a multi-edged sequence. Whether no refinement took place for the edge-sequence being determined, or a great deal of refinement took place, it is still the edge-sequence that is being determined and it does indeed represent an edge-sequence with two particular endpoints.

In section 2, we gave an example of an intersection and a union (from Figure 1), but neither was for an edge-sequence determination. Recall, that the intersection violated the K-rule, thus equals zero, but the union was not evaluated to be zero, and it's properly defined as a multi-edged sequence.

The second observation is that the distributive law for unions and intersections does not apply here. Otherwise, for example, we could say that: $(I_{1,b} \cap I_{4,d}) \cup (I_{2,a} \cap I_{4,d}) \cup (I_{3,c} \cap I_{4,d}) \;=\; (I_{1,b} \cup I_{2,a} \cup I_{3,c}) \cap I_{4,d}$ which is not true in general.

In the algorithm, we provide a Comparing between two S-sets that will do more than once in each direction. If doing the second direction, $S_{k,l} \overset{2}{\leftharpoonup} S_{i,j}$ results in one or more bit-changes, we will do $S_{k,l} \overset{3}{\rightharpoonup} S_{i,j}$ again. And if one or more bit-changes occur in $S_{k,l} \overset{3}{\rightharpoonup} S_{i,j}$ we will do $S_{k,l} \overset{4}{\leftharpoonup} S_{i,j}$ again, and so on until no bit-change occurs while doing a direction. An important note, is that we don't have to do this. If we only did $S_{k,l} \underset{2}{\overset{1}{\rightleftharpoons}} S_{i,j}$ no matter how many bit-changes occurred while doing the second direction, it wouldn't matter. Suppose doing $S_{k,l} \overset{3}{\rightharpoonup} S_{i,j}$ would have had a bit-change. Then, if that scenario still exists in the next **run**, meaning that some other Comparing of S-sets didn't do the refinement in question prior to returning to these two S-sets Comparing, a bit-change would still occur for $S_{k,l} \rightharpoonup S_{i,j}$. We are assuming by doing more than just $S_{k,l} \underset{2}{\overset{1}{\rightleftharpoons}} S_{i,j}$ if prompted by a bit-change occurring during $S_{k,l} \overset{2}{\leftharpoonup} S_{i,j}$ is an efficiency measure.

Recall definition 3.3, that after all pairs of S-sets have been compared with each other, a **run** has been completed, which is $\binom{\binom{c}{2}}{2} < c^4$, for c clauses. Another **run** will commence if any refinement occurred. Eventually, a **run** will incur no refinement, not even a bit-change, at which point a **round** has been completed, and the S-sets are said to be **equivalent**. Or, the algorithm stopped because unsatisfiability had been discovered during **round** 1. Discovering unsatisfiability is when every literal from some clause is such that their vertex-sequences equal zero.

An example of Comparing

The two **Comparings** shown in this example, produce the same outcome as the action for refinement rule 2.

Let the S-set: $S_{i,j}$ contain 9 edge-sequences with endpoints from clauses: $C_i = (a, 2, 3)$ and $C_j = (b, 4, 5)$ giving: I_{a_i,b_j}, $I_{a,4}$, $I_{a,5}$, $I_{2,b}$, $I_{2,4}$, $I_{2,5}$, $I_{3,b}$, $I_{3,4}$, $I_{3,5}$

Now suppose we have clause: $C_k = (c, 6, 7)$. Then, $I_{a_i,c_k} \in S_{i,k}$ and $I_{b_j,c_k} \in S_{j,k}$ Further suppose that I_{a_i,b_j} incurred a bit-change for literal c.

First we consider: $S_{i,j} \underset{1}{\overset{2}{\rightleftharpoons}} S_{i,k}$ where we determine I_{a_i,c_k}

Observe that all edge-sequences: $I_{2,b}$, $I_{2,4}$, $I_{2,5}$, $I_{3,b}$, $I_{3,4}$, $I_{3,5}$ have a zero entry for a_i, so they are not selected for the determination of I_{a_i,c_k}, again because their intersection would be just zero. Since, I_{a_i,b_j} has zero entries for all occurrences of c, then $I_{a_i,b_j} \cap I_{a_i,c_k} = 0$. Thus, the only two edge-sequences from $S_{i,j}$ to determine I_{a_i,c_k} is: $I_{a,4}$ and $I_{a,5}$, both of which have a zero entry for b_j. Therefore, the determination of I_{a_i,c_k} is a refinement where at least b_j has a zero entry. Then after refinement rule 1 is applied, all occurrences of literal b in I_{a_i,c_k} are zero, assuming $I_{a_i,c_k} \neq 0$, after LCR and K-rule compliancy.

Similarly, we consider: $S_{i,j} \underset{1}{\overset{2}{\rightleftharpoons}} S_{j,k}$ where we determine I_{b_j,c_k}

Observe that all edge-sequences: $I_{a,4}$, $I_{a,5}$, $I_{2,4}$, $I_{2,5}$, $I_{3,4}$, $I_{3,5}$ have a zero entry for b_j, so they are not selected for the determination of I_{b_j,c_k}. Since, I_{a_i,b_j} has zero entries for all occurrences of c, then $I_{a_i,b_j} \cap I_{b_j,c_k} = 0$. Thus, the only two edge-sequences from $S_{i,j}$ to determine I_{b_j,c_k} is: $I_{2,b}$ and $I_{3,b}$, both of which have a zero entry for a_i. Therefore, the determination of I_{b_j,c_k} is a refinement where at least a_i has a zero entry. Then after refinement rule 1 is applied, all occurrences of literal a in I_{b_j,c_k} are zero, assuming $I_{b_j,c_k} \neq 0$, after LCR and K-rule compliancy.

Summary: Pre-processing begins with a DIMACS file submission, providing clauses, where each clause is at most size 3. The pre-processing entails ordering the literals within each clause (by any convention desired), and then removing duplicate clauses or literals within a clause. Next, the clauses to be removed have at least one *pure* literal, or they are quantum clauses. This

may be a recursive process. Finally, clauses of size 1, a *loner* clause, are removed as are the negations of the literals from these clauses of size 1 from all the other clauses that were given. This may also be recursive, as a negation may have been in a clause of size 2, thus its removal created a new loner clause. At this point, edge and vertex sequences are constructed from the remaining clauses. The edge-sequences are grouped in their respective S-sets and the vertex-sequences are grouped by their clause association in the vertex-sequence table. When the vertex and edge sequences are LCR and K-rule compliant, the Comparing process begins. The Comparing process stops if: i) a clause was such that all its literals' vertex-sequences are zero, where it's reported that the given collection of clauses has no solution. Or, ii) one or more **runs** take place, where the last **run** had no refinement. This signals the end of a round, and the S-sets are said to be **equivalent**.

Claim : If a 3-SAT $\mathcal{G}$, with c clauses (after pre-processing), has a solution K_C, then the S-sets for $\mathcal{G}$, will be equivalent at the completion of round 1, where each edge-sequence $I_{x,y}$ that appears, belongs to at least one solution. Moreover, an edge-sequence $I_{x,y}$ is such that any literal with a 1 entry in $I_{x,y}$ belongs to at least one K_C with x and y. Lastly, if at least one clause is such that all its literals' vertex-sequences are zero, it means there is no solution for the collection of clauses that were processed, thus unsatisfiability has been discovered, which stops the processing of round 1.

In the next section we establish the claim. This is followed by proving that attaining S-set equivalency is always achieved in polynomial time.

4 Proof of correctness and termination

Comparing and the Refinement rules

The actions of the refinement rules are taken if one of the four refinements occurred, by the Comparing process. We assert that: 1) If by Comparing, a bit-change occurred for one position of a literal x in some edge-sequence I, the Comparing process will eventually cause a bit-change for every position of x in I. 2) All edge-sequences $I_{x,y}$ have the same set of literals with zero

entries via Comparing. 3) If by Comparing, an edge-sequence is evaluated to be zero, then all edge-sequences whose endpoints have the same associated literals, will be evaluated to be zero, by Comparing. And finally, 4) If by Comparing, a literal's vertex-sequence V_{x_i} is evaluated to be zero, then all vertex-sequences for literal x belonging to other clauses, will be evaluated to be zero, by Comparing. Note that we have already shown in the Comparing example, that refinement rule 2 is done by Comparing.

Since we intend to prove the algorithm that does use the refinement rules, then to determine if its actions are merely an efficiency isn't warranted. Moreover, the argument to use the actions of the refinement rules ensures contradiction free edge-sequences. More precisely, if a bit-change occurred for one occurrence of literal c in edge-sequence $I_{a,b}$ by Comparing, then with respect to literals, there is no solution using literals a, b and c together. Therefore, if another occurrence of literal c exists in $I_{a,b}$ it's still the case that there is no solution using literals a, b and c together. Thus, there would be no purpose for other occurrences of c in $I_{a,b}$ to have 1 entries. It follows, that if there exists another $I_{a,b}$ that belongs to another S-set, that there would be no purpose for any occurrence of c to have 1 entries in this $I_{a,b}$. And, if by Comparing, $I_{a,b} = 0$, then with respect to literals, there is no solution using literals a and b together, so all edge-sequences whose endpoints are associated with literals a and b should be zero to be contradiction free. And finally, if a literal's vertex-sequence $V_{x_i} = 0$, by Comparing, meaning no solution exists using literal x, then with respect to literals, the vertex-sequence for every occurrence of x should be zero to be contradiction free. We would prefer to provide the proof of the assertion if there should be any interest.

Another efficiency rule of note, but not part of our algorithm, is with respect to the vertex-sequences. Suppose during the Comparing process, it is determined that $V_x = 0$ say, and then at some later time in the Comparing process, it is determined that $V_{-x} = 0$. This implies that the 3-SAT $\mathcal{G}$, being processed is unsatisfiable. More precisely, should $\mathcal{G}$ have a solution that does not use x or $-x$, it implies that there is also a solution that does use x and another solution that does use $-x$, a contradiction. Therefore, $\mathcal{G}$ must be unsatisfiable, so processing can stop.

Lemma 4.1. *Let $\mathcal{W}$ be a collection of S-sets with LCR and K-rule compliant edge-sequences of length c cells. Then, no edge-sequence from $\mathcal{W}$, belonging to a K_C, is removed by the Comparing process.*

Proof. Suppose there exists a K_C, $\mathcal{M}$. Then, there exists an edge-sequence in each S-set of $\mathcal{W}$, such that $\bigcap_{i,j} I_{i,j} \neq 0$, where i and j are every pair of endpoints for the collection of edge-sequences of $\mathcal{M}$, by definition 2.10. So, if we take any edge-sequence I, belonging to $\mathcal{M}$, it will still exist after the determination of I from every other S-set. The reason is that each S-set $S_{i,j}$, has an edge-sequence I_{x_i,y_j}, call it I', of $\mathcal{M}$, and the intersection $I \cap I'$, preserves all the 1 entries for every literal that is part of the solution, by definition of $\mathcal{M}$. Thus, $I \cap I'$ will not equal zero after LCR and K-rule compliancy, and the union of any additional intersections with I, for any given determination, will be of no consequence. Additionally, I may be refined due to refinement rules taken on it or on other edge-sequences, but this will not effect the 1 entries for the literals that belong to $\mathcal{M}$. That is, if any 1 entry in I for a literal of $\mathcal{M}$, did incur a bit-change, it would imply that another edge or endpoint of $\mathcal{M}$, equals zero, a contradiction. In other words, every determination of I with all the $\binom{c}{2} - 1$ S-sets will at most refine I, since there is an edge-sequence belonging to $\mathcal{M}$, in every S-set, and their intersection, $I \cap I'$, preserves the 1 entries for every literal that is part of the solution $\mathcal{M}$. And, no 1 entry in I for a literal of $\mathcal{M}$, is removed via the actions of the refinement rules on any edge-sequence, otherwise $\mathcal{M}$ did not exist, a contradiction.

∎

Observe that the proof for lemma 4.1 establishes that if an edge-sequence $I_{x,y}$ and some literal z belong to a K_C, then z does not incur a bit-change in $I_{x,y}$ by Comparing.

Lemma 4.2. *Let $\mathcal{X}$ be a collection of equivalent S-sets with edge-sequences of length c cells, for a 3-SAT $\mathcal{G}$ with c clauses. Suppose an edge-sequence $I_{x,y}$ from $\mathcal{X}$, is such that its 1 entries correspond to just edge-singletons or singletons, which includes x and y. Then, $I_{x,y}$ belongs to at least one K_C.*

Proof. Let an edge-sequence be $I_{x,y}$. Observe, that if there is a collection $\mathcal{Q}$ of 1 entries, one from each cell, whose corresponding literals are such that no literal and its negation appear, then there is a solution by definition

2.1. And, by lemma 4.1, the edges between those literals which did exist at the outset, will not be removed by the Comparing process, thus there is a corresponding K_C. ie. a solution as defined in 2.10.

Let an edge-sequence $I_{x,y}$ with c cells, that is LCR and K-rule compliant, be such that its 1 entries correspond to just edge-singletons or singletons.

We shall construct a collection $\mathcal{Q}$ by first selecting the literals from all loner cells. This sub-collection is non-empty since endpoints x and y are in loner cells. Recall, that if $I_{x,y}$ is LCR compliant, then the negations of the literals in loner cells do not exist in $I_{x,y}$. Assume that the collection $\mathcal{Q}$ being constructed is not yet size c. Note that at this point, there are only cells with at least two 1 entries, from which to select a literal. For selecting literals from such cells, we make two cases to simplify the description of constructing a collection $\mathcal{Q}$.

Case i): For edge-sequence $I_{x,y}$ there are no edge-pure literals in cells with two or three 1 entries.

Step 1: We select any cell C_i and choose one of the literals, say a, to represent the cell. Next, we find the cell that contains the 1 entry for literal $-a$, say C_j. If cell C_j has a 1 entry for the negation of a literal in C_i, which is not representing C_i then step 1 stops. Suppose C_j does not. Then, there exist one or two 1 entries for literals not $-a$, that are not the negations of any literal with a 1 entry in C_i. Choose one of these literals to represent cell C_j and find the cell C_k that contains the 1 entry for the negation of the literal chosen to represent C_j. If cell C_k has a 1 entry for the negation of a literal in C_i or C_j which is not representing C_i or C_j then step 1 stops. If this is not the case, then continue step 1 as described above, until a cell is finally selected which does have a 1 entry for a literal which is the negation of some literal in a previously selected cell, that does not represent that cell. Should this never occur, then the last cell selected is the c^{th} cell, and there exists at least one 1 entry for a literal that is not the negation of any literal that represents a cell.

Step 2: Suppose step 1 stops and all c cells have not been selected. Then the last cell selected has at least one 1 entry for literal z say, which is the negation of some literal in a previously selected cell, that does not represent that cell. The literal z will represent the last cell selected. Now, we will find

all cells not yet selected, that contain the negations of literals in previously selected cells which did not represent their cells. These negations will be chosen to represent the cells in which they were found. Of course, should more than one negation belong to the same cell, we pick any one to represent the cell. Next, after selecting the cells that contain the negations of literals in previously selected cells, we check these cells to see if there are literals not representing their cells, that are not the negations of any literal among the cells selected thus far. If such literals exist, then we will find all cells not yet selected, that contain the negations of these literals. These negations will be chosen to represent the cells in which they were found. We repeat this part of step 2 as just described, until there are no literals of this kind. At this point, if all c cells have not been selected, then a *closed circuit* exists, meaning the collection of selected cells (except the loner cells), contain a literal and its negation. If this is the case, the collection of cells not yet selected, contain a literal and its negation as well. Therefore, we apply step 1 again on this remaining collection of cells which have no literal representing them. Observe that no literal from the cells of the *closed circuit* are contained in these remaining cells. Step 1 may stop again before all c cells have been selected, in which case, step 2 is applied. This may be recursive until each of the c cells has a literal representing it, thus a $\mathcal{Q}$ of size c will have been constructed.

Case *ii*): For edge-sequence $I_{x,y}$ there exist, one or more cells having two or three 1 entries, that contain at least one edge-pure literal.

After selecting all literals from loner cells, we select all the edge-pure literals from cells having two or three 1 entries. These literals will represent their cell and if a cell has 1 entries for more than one edge-pure literal, any one can be chosen to represent that cell. So, selecting cells with an edge-pure literal may leave the cells with no representative as yet, with edge-pure literals among them. Thus, this process may be recursive. If this recursive process is not all c cells, then it is a *closed circuit*, otherwise the process would continue. We are now in case i) again, so we begin with step 1 for the remaining cells which have no literal representing them thus far. Again, all of the above may be recursive until each of the c cells has a literal representing it, producing a collection $\mathcal{Q}$ of size c. Since we were able to construct a collection $\mathcal{Q}$ of size c for the two possible cases outlined above, it follows that $I_{x,y}$ belongs to at least one K_C.

■

Note well that lemma 4.2 implies that if the stated scenario occurs, in a $\mathcal{W}$ produced for a 3-SAT $\mathcal{G}$, then the remaining processing can finish with a linear time construction of a solution for $\mathcal{G}$.

S-set collections and refinements

We claim that no $K_{C-1} \not\subset K_C$ for any collection of S-sets $\mathcal{W}$ with LCR and K-rule compliant edge-sequences of length c cells, constructed from the c clauses for a 3-SAT $\mathcal{G}$, exists. Suppose otherwise. Then, let $\mathcal{M}$ be a $K_{C-1} \not\subset K_C$, and let clause $C_i = (a, b, c)$ be the clause that does not contain an endpoint for any edge of $\mathcal{M}$. Then at most, only two negations of the literals in C_i could be used for $\mathcal{M}$, since all edge-sequences must be LCR compliant. So, suppose $-a$ and $-b$ are endpoints for edges of $\mathcal{M}$. Then, there exists an edge-sequence $I_{-a,-b}$ for $\mathcal{M}$. However, $I_{-a,-b}$ has a zero entry for every occurrence of literal $-c$, otherwise, $I_{-a,-b}$ would not be LCR compliant, thus, by definition 2.10, $\mathcal{M}$ can't use literal $-c$. Since, there were edges between literal $c \in C_i$ with every literal who is an endpoint for all the edges of $\mathcal{M}$, then by lemma 4.1, the edges between those literals which did exist at the outset, will not be removed by Comparing. This implies that $\mathcal{M} \subset K_C$, contradicting the assumption that $\mathcal{M}$ was not contained. Observe, that the same argument holds, no matter which two literals of C_i had their negations as endpoints for edges of $\mathcal{M}$. Also, if a K_{C-1} uses only one or no negation of a literal from the clause that does not contain an endpoint for any edge of K_{C-1}, then the K_{C-1} is contained in some K_C, by lemma 4.1.

With respect to just literals, there is a scenario sometimes referred to as *atomica*, which is: given a collection of m clauses, there is a solution for any choice of $m-1$ clauses, but no solution for all m clauses. However, the *atomica* scenario never occurs with any $\mathcal{W}$, due to LCR compliancy of the edge-sequences, as demonstrated above. ie. $\nexists$ a K_{C-1} that uses the negations of all the literals from the clause not containing an endpoint for the K_{C-1}. In other words, by construction, all $K_{C-1} \not\subset K_C$ scenarios are eliminated.

So, if a literal x is such that it does not belong to any K_C for some $\mathcal{W}$ with edge-sequences of length c cells, then by the claim above, it's known that x does not belong to any K_{C-1} of $\mathcal{W}$, as well. Moreover, x does not belong to any K_P, $p < c-1$, if there exist two or more clauses not having endpoints for

any edge of the K_P, where all together, they have no literal and its negation among them. For example, suppose there are two clauses $C_i = (a, b, c)$ and $C_j = (d, e, f)$, where the literals from C_i and C_j together, do not have a literal and its negation. Now, consider a K_P, so a K_{C-2} denoted $\mathcal{L}$, that has no endpoint for any of its edges in C_i or C_j. Suppose to the contrary that $x \in \mathcal{L}$. Now, $\mathcal{L}$ can at most, only use two negations for the literals of C_i, and at most, only two negations for the literals of C_j. Since, C_i and C_j together do not contain a literal and its negation, there exists a K_2 (an edge), between C_i and C_j for any possible choice of four negations of the literals from C_i and C_j which implies that $\mathcal{L} \subset K_{C-1} \subset K_C \implies x \in K_C$, a contradiction.

Note that every collection of nine or more clauses after pre-processing, have at least a pair of clauses with no literal and its negation among their literals.

Refinement Claim : Suppose a collection of S-sets $\mathcal{W}$ incurred one or more bit-changes called refinement R, by Comparing, at some stage in the Comparing process. Now suppose we imposed in an arbitrary fashion, bit-changes to the edge-sequences of $\mathcal{W}$, and call it refinement S, prior to any Comparing. Then, the same refinement R, would still occur by the Comparing of $\mathcal{W}$, or R is fully or partly subsumed by S.

Proof: We first note that refinements are due to the relationships between the edge-sequences and conversely, refinements cause relationships between the edge-sequences. So, if a collection of edge-sequences cause a bit-change for a literal c, in some edge-sequence $I_{a,b}$ during some order of Comparing, then imposing a bit-change to any of the edge-sequences involved, prior to any Comparing, will still result with no possibility of a K_C, using literals a, b and c together. More precisely, suppose an edge-sequence $I_{a,b}$ incurs a bit-change for a literal c by some order of Comparing. Now suppose that before any Comparing, we imposed any arbitrary refinement S. There are three cases to consider.

The first case is that refinement S, played no role in the bit-change of c in $I_{a,b}$. So, proceeding with the same order of Comparing, still results in a bit-change for c in $I_{a,b}$.

The second case is when the refinement S, does play a role, where during the same sequence of Comparing, S causes the bit-change of c in $I_{a,b}$ to oc-

cur sooner, during the same order of Comparing. The refinement S may have also caused other refinements, including to $I_{a,b}$ during the same order of Comparing.

The third case is when the refinement S, does play a role by subsuming some or all of R. The most extreme would be that S causes $I_{a,b} = 0$, during the same order of Comparing, which is the ultimate refinement of an edge-sequence. If $I_{a,b} = 0$, then there is no possibility of a K_C, using literals a, b, thus no K_C, using literals a, b and c. ie. the bit-change to c in $I_{a,b}$ was subsumed. The only note of interest for the third case is suppose an edge-sequence $I_{x,y}$ did play a role in the refinement of $I_{a,b}$ with no imposed bit-changes. ie. no S. Now suppose that with S, $I_{x,y} = 0$, during the same sequence of Comparing. In such a case, at least the bit-change of c in $I_{a,b}$ still occurs during the same order of Comparing, because $I_{x,y}$ was just one edge-sequence that was part of a union of edge-sequences from its S-set that refined either $I_{a,b}$ or it refined another edge-sequence which refined another, and so on, until some union of edge-sequences then refined $I_{a,b}$. More generally, the remainder of the union of edge-sequences where $I_{x,y} = 0$, determining an edge-sequence I when Comparing, will still refine I, if the union that included $I_{x,y}$ had. Or, it is the case that the remainder of the union of edge-sequences determining an edge-sequence I, can now refine I, because $I_{x,y} = 0$. The case being considered for $I_{a,b}$ above, is the former. In summary, refinements do not prevent other refinements, unless one is subsumed by another.

Additionally, it should be clear that if sub edge-sequences of length p cells for some $\mathcal{W}$ with edge-sequences of length c cells, were to be Compared using just the $\binom{p}{2}$ S-sets and a refinement R incurred, that increasing the length of those sub edge-sequences to c cells, prior to the same order of Comparing, will not prevent refinement R, to occur, while processing the same S-sets. This of course, assumes that some or all of refinement R, was not subsumed while performing the same order of Comparing, due to the addition of $k = c-p$ cells. ie. if the additional k cells played no role wrt. refinement R. The reason is simply that the refinements were due to the relationships between the first p cells for those same edge-sequences, regardless of the size of k.

Recall that even a single bit-change always eliminates at least one K_C possibility. So, if an edge-sequence $I_{a,b}$ incurs a bit-change in some position for a literal c, then there is now, no K_C possible, using literals a, b and c together, by definition 2.10. Equally important to remember, is that the bit-change for c in $I_{a,b}$ does not necessarily eliminate a K_C possibility that does not use literals a, b and c together.

Theorem 4.1. *Let a collection of equivalent S-sets $\mathcal{X}$ with edge-sequences of length c cells, be the result of Comparing a collection of S-sets $\mathcal{W}$. Then, an edge-sequence $I_{x,y}$ from $\mathcal{X}$, is such that a literal with a 1 entry in $I_{x,y}$ belongs to at least one K_C with x and y.*

Recall that there is no collection of equivalent S-sets if unsatisfiability is determined, which is discovered during the first round.

Proof. The proof is by induction on the number of clauses c, and the base case shall be $c = 3$.

Let $c = 3$. Observe that the combinations of clause sizes, for three clauses are: **i**) $3, 3, 3$ **ii**) $3, 3, 2$ **iii**) $3, 2, 2$ **iv**) $3, 3, 1$ **v**) $3, 2, 1$ and **vi**) $3, 1, 1$.

A 3-SAT with three clauses could have as many as 27 edge-sequences, nine in each S-set constructed or as few as 7 edge-sequences for **vi**) above. Note that pre-processing would eliminate **iv**), **v**) and **vi**) due to the loner clauses, nor could 27 edge-sequences be possible without the existence of *pure* literals. However, we exclude the pre-processing routine from the proof.

So, after the application of the LCR and K-rule to the newly constructed edge-sequences, and any applicable action of the refinement rules, the edge-sequences that remain will have one of the three possible forms:

1. $I_{x_1,y_2} : (1_x, 0, 0 \mid 1_y, 0, 0 \mid 1, 0, 0)$

2. $I_{x_1,y_2} : (1_x, 0, 0 \mid 1_y, 0, 0 \mid 1, 1, 0)$

3. $I_{x_1,y_2} : (1_x, 0, 0 \mid 1_y, 0, 0 \mid 1, 1, 1)$

$WLOG$, we can assume that the edge-sequence $I_{x,y}$ has its endpoints in the first and second cell. The third cell must have one, two or three 1 entries, to be K-rule compliant. In **1**, literals x_1 and y_2 belong to one K_3 with some literal having a 1 entry in the third cell, and in **2**, x_1 and y_2 belong to a K_3 with each of the two literals in cell three having a 1 entry. In **3**, literals x_1 and y_2 belong to a K_3 with each of the three literals in cell three having a 1 entry.

Claim: If the three S-sets having edge-sequences with 3 cells, require no more actions from the refinement rules to be taken and are LCR, K-rule compliant, then the S-sets are already equivalent. ie. Only one **run** occurs.

Proof: Let literal x be associated to cell C_1, y to cell C_2 and z to cell C_3. Suppose I_{x_1,y_2} has a 1 entry for z_3. Then, let I_{x_1,z_3} have a 1 entry for y_2 and I_{y_2,z_3} have a 1 entry for x_1. Now, suppose instead that I_{x_1,z_3} has a 0 entry for y_2. Then, by the action of refinement rule 2, I_{x_1,y_2} will have a 0 entry for z_3 and I_{y_2,z_3} will have a 0 entry for x_1. Since, the edge-sequences were such that no actions of the refinement rules applied, it can not be the case. Then, it is the case that $I_{x_1,y_2} \cap I_{x_1,z_3} \cap I_{y_2,z_3} \neq 0$, thus, these three edge-sequences belong to a K_3. Therefore, an edge-sequence $I_{x,y}$ belonging to a collection of three equivalent S-sets $\mathcal{X}$, is such that any literal with a 1 entry in $I_{x,y}$ belongs to at least one K_3 with x and y.

Suppose now that $\mathbf{c} > \mathbf{3}$ is an integer for which the statement of the theorem is valid. We consider cases **1**, **2** and **3**.

Case 1i): Let a 3-SAT $\mathcal{G}$, with c+1 clauses have a non-singleton literal x that does not belong to any K_{C+1}. ie. there is no solution using literal x.

Let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less one clause containing literal x. Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. If we apply Comparing to $\mathcal{W}'$, either we determine unsatisfiability in which case, x belongs to no solution, or a collection of equivalent S-sets $\mathcal{X}'$, is produced where the hypothesis holds. If $\mathcal{X}'$ contains edge-sequences with an endpoint associated to literal x, then x belongs to a K_C $\mathcal{K}$, for $\mathcal{G}'$, thus x would also belong to a K_{C+1} for $\mathcal{G}$, using x from the $(c+1)^{th}$ clause. This is true since there are edges between one occurrence of literal x_i say, used in $\mathcal{K}$ and all the other literals of $\mathcal{K}$, so there must also be edges between any x_j and all the other literals of $\mathcal{K}$ (except the literal in cell j that was replaced with x_j), due to

the refinement rules. Therefore, x does not appear in any edge-sequence or vertex-sequence for $\mathcal{G}'$ which implies x does not appear in any edge-sequence or vertex-sequence for $\mathcal{G}$ as well. This is so, since the same refinement would occur or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing a collection of equivalent S-sets $\mathcal{X}$.

Case 1ii): Let a 3-SAT $\mathcal{G}$, with c+1 clauses have a singleton literal x that does not belong to any K_{C+1}.

Let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less one clause of the form: (y, l_1, l_2), where literal y is a non edge-singleton wrt. $I_{x,y}$ of $\mathcal{G}$. Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. If we apply Comparing to $\mathcal{W}'$, either we determine unsatisfiability in which case, literal x belongs to no solution, or a collection of equivalent S-sets $\mathcal{X}'$, is produced where the hypothesis holds. Suppose there is an edge-sequence $I_{x,y}$ from $\mathcal{X}'$. If $\mathcal{X}'$ contains edge-sequence $I_{x,y}$, then x belongs to a K_C $\mathcal{K}$, for $\mathcal{G}'$, so x would also belong to a K_{C+1} for $\mathcal{G}$, using y from the $(c+1)^{th}$ clause. As in **1**i), there must be edges between any y, so in particular y from $(c+1)^{th}$ clause and the literals for $\mathcal{K}$, due to the refinement rules. It follows then, that $I_{x,y}$ does not appear in $\mathcal{X}'$ which implies it also does not appear in a collection of equivalent S-sets $\mathcal{X}$ for $\mathcal{G}$. Again, because the same refinement would occur or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing $\mathcal{X}$. If we construct other 3-SATs with the clauses of $\mathcal{G}$ less one clause with the property of containing at least one literal # that is a non edge-singleton wrt. $I_{x,\#}$ of $\mathcal{G}$, eventually all edge-sequences with one endpoint associated to x is of the form: $I_{x,\%}$ where % is an edge-singleton or just a singleton, and every position associated to a non edge-singleton has a zero entry, due to the actions of refinement rule 3. And, by lemma 4.2, if $I_{x,\%}$ is such that its 1 entries correspond to just edge-singletons or singletons, including x and %, then it belongs to at least one solution. Since x does not, then there is no $I_{x,\%}$ in $\mathcal{X}$, so it follows that $V_x = 0$.

Case 2i): Let a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$, with c+1 clauses have an edge-sequence $I_{x,y}$ that does not belong to any K_{C+1} where y is a non edge-singleton wrt. $I_{x,y}$.

Assume that x and y do belong to (different) solutions, otherwise we are in Case 1. Let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less one clause

of the form: (y, l_1, l_2), where y is a non edge-singleton wrt. $I_{x,y}$ of $\mathcal{G}$. Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. If we apply Comparing to $\mathcal{W}'$, either we determine unsatisfiability in which case, $I_{x,y}$ belongs to no solution, or a collection of equivalent S-sets $\mathcal{X}'$, is produced where the hypothesis holds. Suppose there is an edge-sequence $I_{x,y}$ from $\mathcal{X}'$. If $\mathcal{X}'$ contains edge-sequence $I_{x,y}$ then y belongs to a K_C $\mathcal{K}$, for $\mathcal{G}'$, so $I_{x,y}$ would also belong to a K_{C+1} for $\mathcal{G}$, using y from the $(c{+}1)^{th}$ clause, just as we saw for Case 1i). So, it also follows that $I_{x,y}$ does not appear in $\mathcal{X}'$ which implies it also does not appear in a collection of equivalent S-sets $\mathcal{X}$ for $\mathcal{G}$. Because, the same refinement would occur or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing $\mathcal{X}$.

Case 2ii): Let a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$, with $c{+}1$ clauses have an edge-sequence $I_{x,y}$ that does not belong to any K_{C+1} and x and y are edge-singletons wrt. $I_{x,y}$.

Again, we assume that x and y do belong to different solutions, otherwise we are in Case 1. Let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less one clause not containing x or y, of the form: (z, l_1, l_2), where literal z is a non edge-singleton wrt. $I_{x,y}$. Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. If we apply Comparing to $\mathcal{W}'$, either we determine unsatisfiability in which case, an edge-sequence $I_{x,y}$ belongs to no solution, or a collection of equivalent S-sets $\mathcal{X}'$, is produced where the hypothesis holds. Suppose there is an edge-sequence $I_{x,y}$ from $\mathcal{X}'$. If $\mathcal{X}'$ contains edge-sequence $I_{x,y}$ with a 1 entry for z, then $I_{x,y}$ also belongs to a K_{C+1} for $\mathcal{G}$, using the literal z from the $(c{+}1)^{th}$ clause, contradicting that $I_{x,y} \notin$ any K_{C+1}. So if $I_{x,y}$ appears in $\mathcal{X}'$ it must have a zero entry for literal z, and it follows that the same refinement would occur or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing $\mathcal{X}$. If we construct other 3-SATs with the clauses of $\mathcal{G}$ less one clause not having x or y, with the property of containing at least one non edge-singleton wrt. $I_{x,y}$ for $\mathcal{G}$, eventually all that could remain having a 1 entry in $I_{x,y}$ are edge-singletons or singletons and every position associated to a non edge-singleton has a zero entry due to the actions of refinement rule 3. And, by lemma 4.2, if $I_{x,y}$ is such that its 1 entries correspond to only edge-singletons or singletons, then it belongs to at least one solution for $\mathcal{G}$, a contradiction. Therefore, $I_{x,y}$ does not exist in $\mathcal{X}$. ie. by Comparing $I_{x,y} = 0$.

Claim: Let a collection of S-sets $\mathcal{W}$, for a 3-SAT $\mathcal{G}$, with c+1 clauses have an edge-sequence $I_{x,y}$. If, by the Comparing process $I_{x,y}$ incurs zero entries for some literal z and for its negation $-z$, Comparing will produce $I_{x,y} = 0$.

Proof: Suppose that Comparing produced the edge-sequences $I_{x,y}$ with zero entries for literal z and its negation $-z$, where y is a non edge-singleton wrt. $I_{x,y}$. If we follow the same process as described in Case 2i), for this $\mathcal{G}$, and suppose we have that $I_{x,y}$ belongs to at least one K_{C+1} for $\mathcal{G}$, because $I_{x,y}$ belonged to a K_C for $\mathcal{G}'$, then a contradiction results. Since, it is assumed that by the Comparing process $I_{x,y}$ will incur zero entries for z and $-z$, then $I_{x,y}$ belongs to a K_{C+1} $\mathcal{M}$, not using z or $-z$. This of course implies that $I_{x,y}$ can also belong to a K_{C+1} using z or $-z$. Since the edges between literal z say, and the literals for some $K_C \subset \mathcal{M}$, must have been there from the outset for $\mathcal{G}$, then by lemma 4.1, they were not removed, contradicting that the 1 entries for z in $I_{x,y}$ of $\mathcal{G}$, were turned to zero entries (bit-changes), by Comparing. So, there could be no $I_{x,y}$ in $\mathcal{X}'$ for $\mathcal{G}'$, which implies that $I_{x,y} = 0$.

Suppose now that Comparing produced the edge-sequences $I_{x,y}$ with zero entries for literal z and its negation $-z$, where x and y are edge-singletons in $I_{x,y}$. If we follow the same process as described in Case 2ii), for this $\mathcal{G}$, and suppose we have that $I_{x,y}$ belongs to at least one K_{C+1} for $\mathcal{G}$, then the same contradiction results. Therefore, Comparing will produce $I_{x,y} = 0$.

Case 3: Suppose an edge-sequence $I_{x,y}$ from a collection of equivalent S-sets $\mathcal{X}$, for a 3-SAT $\mathcal{G}$, with c+1 clauses, belongs to at least one K_{C+1}.

First, consider any non edge-singleton literal s that's not x or y, in $I_{x,y}$. Let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less one clause of the form: (s, l_1, l_2). Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. If we apply Comparing to $\mathcal{W}'$, a collection of equivalent S-sets $\mathcal{X}'$, is produced where the hypothesis holds. We know that $\mathcal{X}'$ with $I_{x,y}$ will be produced because $\mathcal{X}$ for $\mathcal{G}$, was produced with the same collection of clauses plus (s, l_1, l_2), where $I_{x,y}$ exists in $\mathcal{X}$. So, if $\mathcal{X}'$ contains edge-sequence $I_{x,y}$ with a 1 entry for s, then $I_{x,y}$ also belongs to a K_{C+1} for $\mathcal{G}$, using the literal s from the $(c+1)^{th}$ clause. If to the contrary, $I_{x,y}$ appeared in $\mathcal{X}'$ with a zero entry for literal s, it would follow that the same refinement would occur, or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing $\mathcal{X}$, where literal s would instead, have a zero entry.

Next, consider the cases where both x and y are non edge-singletons each occurring at least twice in I_{x_i,y_j} or only x is a non edge-singleton which occurs at least three times in I_{x_i,y_j}, where we follow the same process as described in Case 1i). For example, suppose x and y are non edge-singletons each occurring twice in I_{x_i,y_j}. Then, there exists two clauses C_m and C_n containing x and y, respectively. Note that m, n, i and j are distinct cells. We shall construct $\mathcal{G}' = \mathcal{G} - \{C_m\}$ and $\mathcal{G}'' = \mathcal{G} - \{C_n\}$ and follow the process as outlined in 1i). For $\mathcal{G}' = \mathcal{G} - \{C_m\}$, we are able to determine the entries for all cells except cells m and i, when examining I_{x_i,y_j} and I_{x_i,y_n}. And, for $\mathcal{G}'' = \mathcal{G} - \{C_n\}$, we are able to determine the entries for all cells except cells j and n, when examining I_{x_i,y_j} and I_{x_m,y_j}. However, all together, every cell had its entries determined at size c, where the hypothesis holds, which implies the entries are also known at size c+1, for these edge-sequences.

Now, suppose that x is a non edge-singleton which occurs three times in I_{x_i,y_j} and y is an edge-singleton wrt. I_{x_i,y_j}. Then, there exists three distinct cells C_m, C_n and C_i containing x. We shall construct $\mathcal{G}' = \mathcal{G} - \{C_m\}$ and $\mathcal{G}'' = \mathcal{G} - \{C_i\}$ then, follow the process as outlined in 1i). For $\mathcal{G}' = \mathcal{G} - \{C_m\}$, we are able to determine the entries for all cells except cell m, when examining I_{x_i,y_j} and I_{x_n,y_j}. And, for $\mathcal{G}'' = \mathcal{G} - \{C_i\}$, we are able to determine the entries for all cells except cell i, when examining I_{x_n,y_j} and I_{x_m,y_j}. Again, all together, every cell had its entries determined at size c, where the hypothesis holds, so it follows that the entries are also known at size c+1 for these edge-sequences.

The $I_{x,x}$ edge-sequences: Observe that the edge-sequences whose endpoints correspond to the same literal such as $I_{x,x}$, are never removed by Comparing, if x belongs to at least one solution. Moreover, each instance of x from any two clauses, x_i and x_j, will have the same vertex-sequences outside cells i and j. Therefore, if a literal z, with a 1 entry in $I_{x,x}$ belongs to at least one solution with x, then the edge-sequence $I_{x,z}$ must exist, which is a Case 2 scenario (assuming both x and z belong to solutions). Conversely, if Comparing produces $I_{x,z} = 0$, then there will be a zero entry corresponding to z in $I_{x,x}$ due to the actions of refinement rule 3.

For the remainder of Case 3, assume that literals x, y and z (all non-*pure* literals), are edge-singletons wrt. $I_{x,y}$, the edge-sequence under consideration. Or, that just y is a non edge-singleton occurring exactly twice in I_{x_i,y_j}.

ie. there exists a y_k where $j \neq k$. And, let literal z belong to at least one solution for $\mathcal{G}$.

The edge-pure literal: From the last claim in case **2**, the implication is that an edge-pure literal z can not be removed by the Comparing process, if the edge-sequence $I_{x,y}$ belongs to a solution. That is, if the Comparing process produced bit-changes for $-z$ in $I_{x,y}$ followed by producing bit-changes for z in $I_{x,y}$ then Comparing will eventually produce $I_{x,y} = 0$. This further implies that if literals x, y and w together, belong to a solution (w has 1 entries in $I_{x,y}$), then x, y , w and z together, belong to a solution as well. Of course, if a literal is *pure* because its negation appeared in no clause for the given 3-SAT, then it too can belong to a solution with x, y and w together. So, either Comparing produces edge-pure literals in edge-sequences, or by construction of the edge-sequences, or it is the case that the negations for some literal occur only in the cells containing x and y where both are singletons or edge-singletons. Note that we could have established the theorem statement for edge-sequences with edge-pure literals as a separate case.

Finally, let a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$, with c+1 clauses have an edge-sequence $I_{x,y}$ such that every solution using x and y together, also uses $-z$. Recall that x, y and z are non-*pure* literals and edge-singletons wrt. $I_{x,y}$.

Now, let $\mathcal{G}'$ be the 3-SAT formed with the clauses of $\mathcal{G}$ less clause $C_i = (-z, a, b)$. Let $\mathcal{W}'$ be the collection of S-sets for $\mathcal{G}'$. Next, we impose a bit-change to each $-b$ of $I_{z,-a}$, each $-a$ of $I_{z,-b}$ and to each (possible), z of $I_{-a,-b}$. For this final case, after the Comparing process is applied to $\mathcal{W}'$, a collection of equivalent S-sets $\mathcal{X}'$, is produced because a collection of equivalent S-sets $\mathcal{X}$, was produced. It also follows that $I_{x,y}$ from $\mathcal{X}'$ exists. Suppose that there is a 1 entry associated to z in $I_{x,y}$ from $\mathcal{X}'$. So, there is no solution using literals z, $-a$ and $-b$ together, because we imposed bit-changes eliminating such a solution for $\mathcal{G}'$. ie. $\nexists$ a K_3 using z, $-a$ and $-b$ together. Then, if there is a solution for $\mathcal{G}'$ using literals x, y and z together, it implies that either a or b can be part of that solution, again because $-a$ and $-b$ together with z can not. However, this contradicts the assumption that there is no solution using x, y and z together, for $\mathcal{G}$, since one of a or b can be used from C_i producing a K_{C+1} (by lemma 4.1), that does use x, y and z together. Therefore, it must be the case that imposing bit-changes to $-b$, $-a$ and z of $I_{z,-a}$, $I_{z,-b}$ and $I_{-a,-b}$ respectively, forces a bit-change to z in $I_{x,y}$ via the Comparing of $\mathcal{W}'$.

Observe that if a and b were *pure* literals, no bit-changes would be imposed on $\mathcal{W}'$, since no negations for a and b exist. And note as in the other cases, it follows that the same refinement would occur or be subsumed by another refinement when Comparing is applied to a collection of S-sets $\mathcal{W}$, for $\mathcal{G}$ producing $\mathcal{X}$. Note well that a $\mathcal{G}'$ could have been formed where the $(c{+}1)^{th}$ clause is any clause not containing either endpoint for $I_{x,y}$ or the clause containing literal z, with the appropriate bit-changes applied to its $\mathcal{W}'$, again, forcing a bit-change to z in $I_{x,y}$ through Comparing. Additionally, if $I_{x,y}$ had an edge-pure literal r, then continuing the Comparing process with the cell containing the edge-pure literal, removed from all edge-sequences, would properly determine which positions in the remaining cells have 1 entries for edge-sequence $I_{x,y}$. With respect to $I_{x,y}$ this is equivalent to forming a $\mathcal{G}'$ with the $(c{+}1)^{th}$ clause containing literal r, and where all positions associated to $-r$ in $I_{x,y}$ of $\mathcal{W}'$, have bit-changes imposed, before Comparing commences. ie. $I_{x,y}$ will have the same 1 entries when equivalency of the initial S-sets $\mathcal{W}'$, is reached. The 1 entries in the cell/s containing r will also correspond. For two or more cells containing r, in turn, each of these cells can be the one removed (and selected as the $(c{+}1)^{th}$ clause). If there is only one occurrence of r_k in $I_{x,y}$, then cell k must have zero entries for the other two positions not associated to r. Otherwise, a solution existed with x and y not using r, implying a solution with x, y and $-r$ does exist, a contradiction. This can be established by not selecting clause C_k to be the $(c{+}1)^{th}$ clause.

We conclude that the inductive statement is valid at $c{+}1$ which completes the proof of the induction step, and so the result follows.

■

By theorem 4.1, if a literal x, does not belong to any K_C for a 3-SAT $\mathcal{G}$, with c clauses (after pre-processing), then x has zero entries for every occurrence, in every edge-sequence and its vertex-sequence V_x equals zero. It follows that if every literal for some clause C_i does not belong to any K_C then cell C_i has all zero entries, and the vertex-sequences associated to clause C_i are equal to zero. Therefore, $\mathcal{G}$ is unsatisfiable, which is always discovered during the first round. Otherwise, the collection of S-sets $\mathcal{X}$, are equivalent, where it is known that $\mathcal{G}$ is satisfiable at the completion of round 1, without necessarily having a known solution. There are 3-SATs where a solution is produced at the completion of round 1, and others during pre-processing, like the example.

Corollary 4.1. *The algorithm applied to any* 3*-SAT* $\mathcal{G}$*, determines satisfiability in polynomial time.*

Proof. The scenario which stops the Comparing process is either: no refinement occurring for an entire **run**, or the vertex-sequences for literals from a clause, are zero. Theorem 4.1 establishes that when no refinement occurs for a **run**, what remains are equivalent S-sets, which implies a solution exists. Otherwise, if during the first **round**, the literals for a clause C_i have vertex-sequences that equal zero, the **run** is stopped. ie. The literals of C_i do not belong to any solution, and $\mathcal{G}$ is reported as unsatisfiable. To make the complexity analysis straightforward for determining satisfiability, we take into account no efficiencies. Further, the worst case scenario assumes redundancies that no 3-SAT would have through processing.

Let n be the sum of the sizes for all c clauses, that remained after preprocessing, for a given 3-SAT $\mathcal{G}$. Then, $c \leq n \leq 3c$. We shall assume that it always took an entire **run** to produce just one bit-change. We shall further assume that to determine satisfiability, it was required that every position in every edge-sequence incurred a bit-change, which of course, would never occur.

The number of edges is $\leq \binom{n}{2}$. Thus, the total number of positions:

$$\binom{n}{2}n = \left(\frac{(n)(n-1)}{2}\right)n = \left(\frac{n^2-n}{2}\right)n < n^3$$

.

Recall, that the number of S-sets $= \binom{c}{2}$. And a **run** would be:

$$\binom{\binom{c}{2}}{2} = \frac{\binom{c}{2}^2 - \binom{c}{2}}{2} = \frac{\left(\frac{c^2-c}{2}\right)^2 - \left(\frac{c^2-c}{2}\right)}{2} < c^4 \leq n^4$$

Now, we need to determine the complexity for a comparison between two S-sets. First, we point out that if only one bit-change was supposed to have occurred for each **run**, implies that for all Compares of two S-sets, $S_{i,j}$ and

$S_{k,l}$ just $S_{i,j} \overset{1}{\underset{2}{\rightleftharpoons}} S_{k,l}$ occurred. ie. every edge in $S_{i,j}$ was **determined** by the edges of $S_{k,l}$ and every edge in $S_{k,l}$ was **determined** by the edges of $S_{i,j}$ and no more was done between those two S-sets. Of course, one Comparing did have a bit-change occur. In this case, at most, one more determination of every edge in $S_{i,j}$ was done with the edges of $S_{k,l}$ again, because the bit-change could have occurred during: $S_{i,j} \overset{2}{\leftharpoonup} S_{k,l}$. The worst case for Comparing in one direction: $S_{i,j} \rightharpoonup S_{k,l}$ would be that every edge-sequence in $S_{k,l}$ have 1 entries for both endpoints of every edge in $S_{i,j}$. If we assume $2n$ steps are required to compare position by position, two sequences of length n, then all of the intersections and unions needed to process one edge is $36(n)$. And, if all 9 edges are the same scenario, then it takes $324(n)$ steps for one direction. Assuming we have the same scenario comparing in the other direction, it would be a total of $648(n)$ steps. Let m be the constant for the work $m(n)$ to do LCR compliancy. LCR is clearly linear, since the work is a look-up and bit-changes for the negations of loner cell literals. K-rule compliancy is known during the $2n$ steps to compare positions, as is the discovery of new loner cells. Now, let M be the constant $648 + m$, where the 648, is for the $648(n)$ steps for all Compares less one which might have used an additional $324(n)$ steps. Then, the work for comparing two S-sets with no bit-change is $M(n)$. The additional work for: $S_{i,j} \overset{3}{\rightharpoonup} S_{k,l}$ would be for an entire **run**. However, we could choose to make M equal to $648 + 324 + m$, which says every Comparing of two S-sets, did $S_{i,j} \overset{3}{\rightharpoonup} S_{k,l}$. It is important to note that the algorithm would produce the same result if only $S_{i,j} \overset{1}{\underset{2}{\rightleftharpoons}} S_{k,l}$ was done regardless if bit-changes occurred. The only computational difference is that it may cause more **runs** to occur, because no follow up Compares were done in the current **run**. However, if another bit-change would have occurred say doing: $S_{i,j} \overset{3}{\rightharpoonup} S_{k,l}$ then that scenario awaits to be done in the next **run**, as explained in section 3. But with respect to worst case, this is irrelevant, because we are assuming that the maximum number of **runs** possible, were done.

Finally, taking the product for: the work of comparing two S-sets (Mn), with the number of S-sets compared per **run** (n^4), and the most **runs** possible (n^3), is the worst case. This gives a complexity of $\mathcal{O}(Mn^8)$, in Big O notation.

■

It's straightforward to show that a solution can be constructed with complexity $\mathcal{O}(Mn^9)$. Just take any edge-sequence $I_{x,y}$ at the end of round 1, remove the cells containing an x, y and a third literal z, who had a 1 entry in $I_{x,y}$. Then, make all positions for $-z$ a zero. Next, reconstruct a new 3-SAT with clauses matching the 1 entries in the remaining cells in $I_{x,y}$ and apply the algorithm again. There will be equivalent S-sets where again we can select an edge-sequence and repeat the above. This can only be done at most $\frac{c}{3}$ times. Thus, we have at most $(n^8)(\frac{c}{3}) < n^9$. However, it's been established that an algorithm can construct a solution, also with complexity $\mathcal{O}(Mn^8)$.

The most important empirical observation with respect to efficiency, is after processing many 3-SATs considered to be hard instances, our original version which had no efficiencies including the actions for the refinement rules, never exceeded n^7 to construct a complete solution, where n is the number of clauses! Thus, we expect any version with one or more non-trivial efficiencies added, to do better than the bound observed empirically. That is why we expect that efficient versions will be practical with respect to today's hardware. Still, it would be reasonable to expect that on average the bound should be n^6. However, there are numerous schemes that allow processing of a round to be much less work, than the work done by the algorithm presented. Of course, it had to be established first, that a general method exists, namely, the Comparing of the S-sets until equivalency is reached.

5 Final comments

Most of our research was devoted to finding efficiencies for both general purpose solving and for specific classes of SATs. We have found dozens so far, and it's clear many more can be developed.

We observed empirically, that with respect to constructing a solution, the combined work done in subsequent rounds was only a fraction of the work done in round 1. Note that this observation is not why the worst case bound remains unchanged for constructing a solution.

One powerful efficiency that was developed, constructs a solution while dispensing with completing rounds altogether, so in particular round 1.

The algorithm described herein does not make use of all the information that is determined while Comparing S-sets, that if used, would only increase the space complexity by adding at most n^3 for lookup tables, while reducing the number of **runs** and dramatically reducing the number of edge-sequence intersections performed. We are assuming that always doing a lookup is less work than performing every intersection when n is sufficiently large.

Additionally, well known efficiency schemes could take on new relevance with respect to processing edge-sequences, by providing analysis at relatively no cost. They will also become non-trivial efficiency schemes for many classes of 3-SATs.

There is a natural extension of the algorithm described in this paper for SAT in general. This can be exploited for efficiency purposes in cases where possibly the conversion to 3-SAT reaches a prohibitive number of objects.

We have developed a way for parallel implementation that requires no communication between the nodes that is not the master node, and at that, it only requires passing information to update the sequences, at specific times.

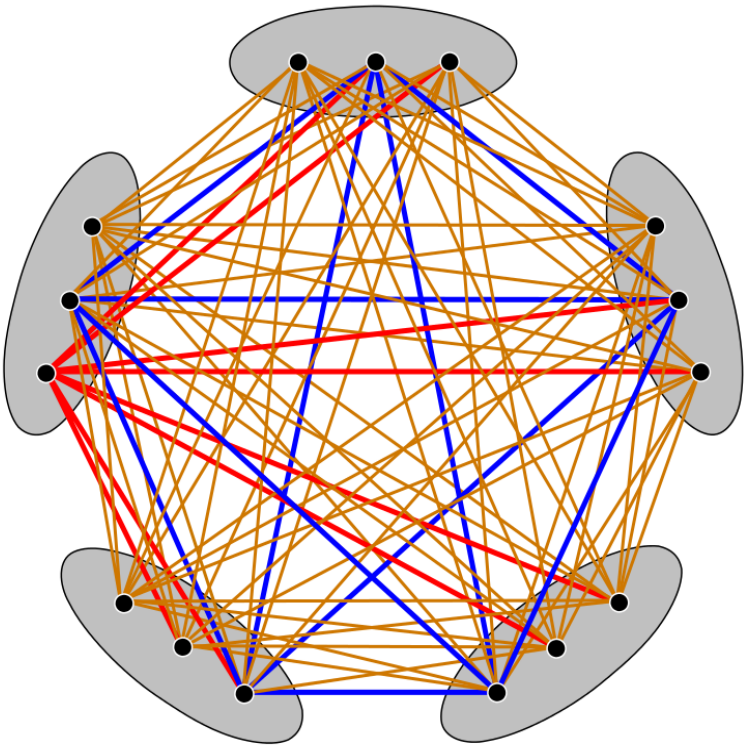
A lot of time was dedicated in the last few years for pre-processing analysis, to determine the best initial choices for Comparing S-sets. We experimented by choosing certain S-sets to Compare first, which in many cases had a dramatic effect on efficiency, and it was these observations that inspired one of our most powerful efficiency schemes.

Lastly, note that code with a few of the non-trivial efficiencies employed, has a complexity less than $\Omega(n^4)$ for 3-SATs of semi-primes.

Example re-visited

The 3-SAT below is Figure 1 with all its edges shown, prior to any application of LCR and K-rule to the corresponding edge-sequences. The blue edges correspond to the brown edges for the K_5 depicted in Figure 1. The edges in red (which will fail to be LCR and K-rule compliant), are those

between the vertex associated to literal a and a vertex associated to some other literal that is not literal $-a$.



References

[1] Cook, Stephen
The complexity of theorem proving procedures.
Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151-158, 1971.

[2] Karp, Richard M.
Reducibility Among Combinatorial Problems.
Complexity of Computer Computations. New York: Plenum.
pp. 85-103, 1972.