From Satisfiability to Linear Algebra

Fangzhen Lin
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Technical Report

August 2013

1 Introduction

Satisfiability of boolean formulas (SAT) is an interesting problem for many reasons. It was the first problem proved to be NP-complete by Cook. Efficient SAT solvers have many applications. In fact, there is a huge literature on SAT, and its connections with other optimization problems have been explored. In this paper, we look at SAT using linear algebra, a basic and fundamental mathematics that have mature theory and efficient solvers like Matlab and Mathematica.

2 Definitions

We assume a vector $\mathbf{x}$ of n variables. A literal is a variable or the negation of a variable, and a clause is a disjunction of literals. In this paper, we sometime treat a clause as a set of literals in it, and without loss of generality, assume that a clause does not contain a variable and its negation at the same time.

Our basic idea is to associate a clause C with a linear combination of the following form:

$$\mathbf{cx}, \tag{1}$$

where $\mathbf{c}$ is a row vector of integers such that c_i is positive if $x_i \in C$, negative if $\neg x_i \in C$, and 0 otherwise, $1 \leq i \leq n$. In the following, we call (1) a linear combination for C, and the vector $\mathbf{c}$ in (1) a coefficient vector for C. If each element of $\mathbf{c}$ is either 0, 1, or -1, then $\mathbf{c}$ is called the standard coefficient vector of C, and $\mathbf{cx}$ the standard linear combination for C.[1]

If S is a set of clauses, then by transforming each clause in it to a linear combination for it, we get a set of linear combinations for it. A set of linear combinations can be

[1] We could use reals as coefficients. But this would not add any generality. For all practical purposes, we only need to consider rational numbers which can then be turned into integers by scaling the involved combinations.

conveniently represented in matrix notaion as a product: $\mathbf{Ax}$. Thus if $\mathbf{A}$ is a matrix such that there is a one-to-one correspondence between clauses in S and row vectors in $\mathbf{A}$: each row in it is a coefficient vector of a clause in S and each clause in S has a coefficient vector as a row in $\mathbf{A}$, then $\mathbf{Ax}$ is a set of linear combinations for S. In the following, we call such a matrix a coefficient matrix for S, and if each of its rows is the standard coefficient vector of a clause in S, then it is called the standard coefficient matrix of S.[2]

Notice that for a clause, there are infinitely many different linear combinations and coefficient vectors for it, and for a set of clauses, infinitely many different sets of lienar combinations and coefficeint matrices for it. However, each linear combination and each vector correspond to exactly one clause.

Example 1 Let S be the set of following clauses:

$$x_1 \vee \neg x_2 \vee x_3$$
$$x_1 \vee x_2$$
$$x_1 \vee \neg x_2 \vee \neg x_3$$

then the standard set of linear combinations for S consists of the following combinations:

$$x_1 - x_2 + x_3,$$
$$x_1 + x_2,$$
$$x_1 - x_2 - x_3.$$

In matrix notation, this is

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 0 \\ 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

The following is an alternative set of linear combinationsfor S:

$$3x_1 - 2x_2 + x_3,$$
$$x_1 + 5x_2,$$
$$x_1 - 2x_2 - 3x_3.$$

3 Reasoning with matrices

In the following, we use bold face for matrices including vectors. If $\mathbf{A}$ is a matrix, then we use A_{ij} to denote its (i, j) cell. If $\mathbf{x}$ is a row or a column vector, then we just use x_i to denote the its ith member. We will use $\mathbf{0}$ to denote a vector whose members are all zero. The dimension of $\mathbf{0}$ should be clear from the context.

Given a $m \times n$ matrix $\mathbf{A}$, we say that it can derive (entails) a row vector $\mathbf{c}$, written $\mathbf{A} \vdash \mathbf{c}$, if there is a row vector $\mathbf{w}$ such that

[2]Strictly speaking, we need to assume an ordering on clauses in S to make its standard matrix unique.

- $\mathbf{wA} = \mathbf{c}$, i.e. $\sum_{i=1}^{m} w_i A_{ij} = c_j$ for each $1 \leq j \leq n$.
- each w_i in $\mathbf{w}$ is greater or equal to 0.
- not all w_i are 0.[3]

It is easy to see that if $\mathbf{A} \vdash \mathbf{c}$, then $\mathbf{c}$ can obtained from $\mathbf{A}$ through the following basic row operations:

1. scale a row by a non-zero number;
2. add a row to another;
3. swap two rows.[4]

However, the converse is not true in general. In fact, it is easy to see that $\mathbf{A} \vdash \mathbf{c}$ iff $\mathbf{c}$ can be obtained from $\mathbf{A}$ when rows are only allowed to be scaled by a positive number, i.e. through the following row operations:

1. scale a row by a positive number;
2. add a row to another;
3. swap two rows.

In the following, given a matrix $\mathbf{A}$, we denote by $\mathbf{A^T}$ the transpose of $\mathbf{A}$, the matrix obtained from $\mathbf{A}$ by swapping rows and columns. The following proposition shows that the problem of checking $\mathbf{A} \vdash \mathbf{c}$ can be reduced to the problem of feasibility checking in linear programming, and thus can be decided in weak polynomial time.

Proposition 1 *For any $m \times n$ matrix $\mathbf{A}$ and any row vector $\mathbf{c}$, $\mathbf{A} \vdash \mathbf{c}$ iff the following linear system has a feasible solution:*

$$\begin{aligned} &\mathbf{A^T u} = \mathbf{c^T}, \\ &\sum_{i=1}^{m} u_i > 0, \\ &u_i \geq 0, \quad 1 \leq i \leq m, \end{aligned}$$

where $\mathbf{u}$ is a vector of m variables.

Proof: For any row vector $\mathbf{w}$, $\mathbf{wA} = \mathbf{c}$ iff $(\mathbf{wA})^T = \mathbf{c^T}$ iff $\mathbf{A^T w^T} = \mathbf{c^T}$. ■

The following result relates resolution to this notion of entailment.

Theorem 1 *Let S be a set of clauses, and $\mathbf{A}$ a coefficient matrix for it. If a clause C can be derived from S by resolution, then $\mathbf{A} \vdash \mathbf{c}$ for some coefficient vector $\mathbf{c}$ of C.*

[3] This condition is needed when $\mathbf{c} = \mathbf{0}$.

[4] This operation is not necessary unless one wants to transform a matrix to a certain normal form.

Proof: Let $C_1, ..., C_k = C$ be a resolution derivation of C. We prove the lemma by induction on k. If $k = 1$, then $C_1 = C \in S$. The result follows as there is a row in $\mathbf{A}$ that is a coefficient vector for C.

Inductively, let the result holds for $k < h$. Consider $k = h$. If $C = C_h$ is in C, then the base case applies. Otherwise, for some $i, j < h$, C is the resolvant of C_i and C_j. Without loss of generality, suppose that for some variable y, $C_i = C_i' \cup \{y\}$, $C_j = C_j' \cup \{\neg y\}$, and $C = C_i' \cup C_j'$. This means that $C \cap \{y, \neg y\} = \emptyset$, and for any variable x, if $x \in C_i'$, then $\neg x \notin C_j'$, and conversely, if $x \in C_j'$ then $\neg x \notin C_i'$. This means that for any variable x, if it occurs in both C_i' and C_j', then it occurs either positively in both A and B or negatively in both C_i' and C_j'.

By the inductive assumption, there are two vectors $\mathbf{w^1}$ and $\mathbf{w^2}$ of non-negative integers, and two coefficient vectors $\mathbf{c^1}$ and $\mathbf{c^2}$ for C_i and C_j, respectively, such that

$$\mathbf{w^1 A} = \mathbf{c^1}, \;\; \mathbf{w^2 A} = \mathbf{c^2}.$$

Suppose a and $-b$ are the coefficients of y in $\mathbf{c^1}$ and $\mathbf{c^2}$, respectively. Then $a, b > 0$. Thus $b\mathbf{c^1} + a\mathbf{c^2}$ is a coefficient vector for C, and

$$b\mathbf{w^1 A} + a\mathbf{w^2 A} = (b\mathbf{w^1} + a\mathbf{w^2})\mathbf{A} = b\mathbf{c^1} + a\mathbf{c^2}.$$

Since both a and b are positive, $b\mathbf{w^1} + a\mathbf{w^2}$ is a vector of non-negative numbers, and contains at least one non-zero component. This proves the inductive case, thus the theorem. ■

Corollary 2 *If a set S of clauses is inconsistent, then for any of its coefficient matrix $\mathbf{A}$, $\mathbf{A} \vdash \mathbf{0}$.*

The converse of Theorem 1 is not true in general. It is instructive to see why by an example. Consider the following two clauses:

$$x_1 \vee x_2 \vee x_3,$$
$$x_1 \vee \neg x_2 \vee \neg x_3.$$

Their standard coefficient matrix is

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

Adding the first row to the second yields $(2, 0, 0)$, which corresponds to the clause x_1. However, the two clauses cannot derive x_1 by resolution. The reason is obvious: resolution eliminates one variable at a time, but adding two rows can eliminate more than one. For this example, the problem can be solved by using non-standard coefficient matrices. For instance, consider the following coefficient matrix for the two clauses:

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

It can be seen that this matrix does not entail any row of the form $(c, 0, 0)$, $c > 0$.

The question is if this can always be done: if a set S of clauses does not ential C logically, is there a coefficient matrix of S that does not entail any of the coefficient vectors of C? The answer is yes.

We first define a notion of reduction on matrices. Given a matrix $\mathbf{A}$ with at least two columns, its reduction is a new matrix generated when its last column is reduced to 0 and then eliminated. Formally, the reduction of $\mathbf{A}$, written $reduce(\mathbf{A})$, is the following matrix (as a set of rows):

- if $(a_1, a_2, ..., a_n, 0)$ is a row in $\mathbf{A}$, then $(a_1, a_2, ..., a_n)$ is a row in $reduce(\mathbf{A})$.
- if
$$r_1 : \ (a_1, ..., a_n, a)$$
and
$$r_2 : \ (a'_1, ..., a'_n, -b),$$
$a, b > 0$, are two rows of $\mathbf{A}$, then
$$r_{12} : \ 1/a(a_1, ..., a_n) + 1/b(a'_1, ..., a'_n)$$
is a row in $\mathbf{A}$. In essence, r_{12} is $r_1/a + r_2/b$: r_1 is scaled by $1/a$ and r_2 by $1/b$ so that the last element of $r_1/a + r_2/b$ becomes zero and can be eliminated. In the following, we call r_1 and r_2 the parent rows of r_{12}.

Notice that we could have chosen to eliminate the first column or any other column in the definition of reduction. We choose to eliminate the last column in order to make the proof of Theorem 3 easier to present.

Lemma 1 *For any* $\mathbf{A}$ *with at least two columns,* $\mathbf{A} \vdash \mathbf{0}$ *iff* $reduce(\mathbf{A}) \vdash \mathbf{0}$.

Proof: Without loss of generality, suppose $\mathbf{A}$ consists of the following rows:

$$\begin{aligned} r_i : &\quad (A_{i1}, ..., A_{in}, a_i), \ \ 1 \leq i \leq m^+, a_i > 0 \\ r_j : &\quad (A_{j1}, ..., A_{jn}, -a_j), \ \ m^+ + 1 \leq j \leq m^-, a_j > 0 \\ r_k : &\quad (A_{k1}, ..., A_{kn}, 0), \ \ m^- + 1 \leq k \leq m. \end{aligned}$$

Thus $reduce(\mathbf{A})$ consists of the following rows:

$$\begin{aligned} r_{ij} : &\quad 1/a_i(A_{i1}, ..., A_{in}) + 1/a_j(A_{j1}, ..., A_{jn}), \ \ 1 \leq i \leq m^+, m^+ + 1 \leq j \leq m^- \\ r_k : &\quad (A_{k1}, ..., A_{kn}), \ \ m^- + 1 \leq k \leq m. \end{aligned}$$

Suppose $reduce(\mathbf{A}) \vdash \mathbf{0}$. It follows that $\mathbf{A} \vdash \mathbf{0}$ as for each row r in $reduce(\mathbf{A})$, $\mathbf{A} \vdash (r, 0)$.

Now suppose $\mathbf{A} \vdash \mathbf{0}$: for some $\mathbf{w}$ of non-negative values, $\mathbf{wA} = \mathbf{0}$. Following up on the notation that we use for $\mathbf{A}$, let $\mathbf{w} = (w_1, ..., w_m)$. Thus

$$\sum_{i=1}^{m^+} w_i a_i = \sum_{j=m^+ +1}^{m^-} w_j a_j, \tag{2}$$

$$\sum_{i=1}^{m} w_i A_{ih} = 0, \ \ \text{for each } 1 \leq h \leq n. \tag{3}$$

We claim that there are non-negative numbers w_{ij} such that

$$w_i a_i = \sum_{j=m^+ +1}^{m^-} w_{ij}, \;\; 1 \leq i \leq m^+, \tag{4}$$

$$w_j a_j = \sum_{i=1}^{m^+} w_{ij}, \;\; m^+ + 1 \leq j \leq m^-. \tag{5}$$

Assuming this, then (3) implies

$$\sum_{i=1}^{m^+} A_{ih}/a_i \sum_{j=m^+ +1}^{m^-} w_{ij} + \sum_{j=m^+ +1}^{m^-} A_{jh}/a_j \sum_{i=1}^{m^+} w_{ij} + \sum_{k=m^- +1}^{m} w_k A_{kh} = 0$$

for each $1 \leq h \leq n$, which can be rewritten as

$$\sum_{i=1}^{m^+} \sum_{j=m^+ +1}^{m^-} w_{ij}(A_{ih}/a_i + A_{jh}/a_j) + \sum_{k=m^- +1}^{m} w_k A_{kh} = 0.$$

Now let $\mathbf{w}' = (w_{11}, ..., w_{m^+ m^-}, w_{m^- +1}, ..., w_m)$, then by the definition of $reduce(\mathbf{A})$, we have $\mathbf{w}' reduce(\mathbf{A}) = \mathbf{0}$. Furthermore, if $\mathbf{w}$ has at least one non-zero member, so does $\mathbf{w}'$. Thus $\mathbf{A} \vdash \mathbf{0}$ implies $reduce(\mathbf{A}) \vdash \mathbf{0}$, provided we have (4) and (5).

The proof of the existence of w_{ij} that satisfy (4) and (5) can be done by induction on m^+ and m^-, and is tedious. The idea is that we can line up $w_i a_i$ for $1 \leq i \leq m^+$ in one column and $w_j a_j$ for $m^+ + 1 \leq j \leq m^-$ in another. Then starting at the top, making the two columns of equal values by decomposing a larger number into a sum of smaller numbers when necessary. Here we illustrate by letting $m^+ = 2$ and $m^- = 4$. Equation (2) is

$$w_1 a_1 + w_2 a_2 = w_3 a_3 + w_4 a_4.$$

If any of coefficient w_i is 0, then let $w_{ij} = 0$ for all j if $i \leq m^+$, and $w_{ji} = 0$ for all j if $i > m^+$. Suppose $w_1 \neq 0$ and $w_1 a_1 \leq w_3 a_3$, then $w_2 a_2 \geq w_4 a_4$. So we decompose $w_3 a_3$ into $w_1 a_1 + (w_3 a_3 - w_1 a_1)$ and $w_2 a_2$ into $(w_3 a_3 - w_1 a_1) + w_4 a_4$, and we have

$$w_{13} = w_1 a_1, w_{14} = 0, w_{23} = w_3 a_3 - w_1 a_1, w_{24} = w_4 a_4.$$

■

In the following, let $reduce^0(\mathbf{A}) = \mathbf{A}$ and $reduce^k(\mathbf{A}) = reduce(reduce^{k-1}(\mathbf{A}))$.

Theorem 3 *A set S of clauses is consistent iff there is a coefficient matrix $\mathbf{A}$ for S such that $\mathbf{A} \vdash \mathbf{0}$ does not hold.*

Proof: The "if" part follows from Theorem 1. We show the "only if" part here. Let σ be an assignment that satisfies S. Given a matrix $\mathbf{X}$, we say that σ satisifes $\mathbf{X}$ if for every row in the matrix, σ satisfies the corresponding clause of the row. We'll construct $\mathbf{A}$ one column at a time. Now consider the following procedure:

1. let $\mathbf{Z}$ be the standard coefficiency matrix of S.
2. $\mathbf{X} = \mathbf{Z}$.
3. if $\mathbf{X}$ has just one column, then exit the procedure with $\mathbf{A} = \mathbf{Z}$.
4. if σ satisfies $reduce(\mathbf{X})$, then $\mathbf{X} = reduce(\mathbf{X})$, and go back to the third step.
5. let k be the number of columns in $\mathbf{X}$. For each member Z_{ij} in $\mathbf{Z}$ such that $j < k$, if $Z_{ij} > 0$ and $\sigma(x_j) = 1$, then increase Z_{ij} by 1, and if $Z_{ij} < 0$ and $\sigma(x_j) = 0$, then decrease Z_{ij} by 1.
6. go back to step 2.

The following are some important properties of this algorithm:

1. $\mathbf{X}$ is always satisfied by σ.
2. on termination, $\mathbf{X}$ has just one column.
3. if n is the number of variables and k the number of columns of $\mathbf{X}$, then $\mathbf{X} = reduce^{n-k}(\mathbf{Z})$.
4. the procedure always terminates.

The first three are easy to see. To show that the procedure always terminates, we need to show that the updates on $\mathbf{Z}$ in step 5 will eventually yield an $\mathbf{X}$ with k columns such that $reduce(\mathbf{X})$ is satisfied by σ. To see this, notice that the changes to $\mathbf{Z}$ in step 5 have no effects on the columns k and after. Furthermore, given that the changes only increase the coefficients of the literals in S that are true in σ, thus for any $i \leq n - k$, if σ satisfies $reduce^i(\mathbf{Z})$ before the changes, then it also satisfies $reduce^i(\mathbf{Z})$ after the change. This means that after the changes, the first time the algorithm enters step 5 again, the number of columns of $\mathbf{X}$ must be k or less. To see that eventually the number of columns of $\mathbf{X}$ will be less than k, consider any row r of $reduce(\mathbf{X})$ that is not satisfied by σ. Suppose the two parent rows of r in $\mathbf{X}$ are (r_1, a) and $(r_2, -b)$ for some $a, b > 0$. Suppose $\sigma(x_k) = 1$ (the case for $\sigma(x_k) = 0$ is symmetric), then σ must satisfy r_2 as it satisfies $(r_2, -b)$ (recall that $\mathbf{X}$ is always satisfied by σ). Suppose $\sigma(x_i) = 1$ and that the ith column of r_2 is positive (again the case for $\sigma(x_i) = 0$ is symmetric). Then there must be a row r' in $\mathbf{Z}$ such that r' is an ancestor of r_2, and the ith column of r' is positive as well. Thus step 5 will increase this coefficient by 1 at a time until the ith column of r becomes positive as well, and when this happens, r is satisfied by σ. This will happen to all rows in $reduce(\mathbf{X})$ that are not satisfied by σ, and thus will eventually make $reduce(\mathbf{X})$ being satisfied by σ.[5]

Since on termination $\mathbf{X}$ has just one column and is satisfiable, thus $\mathbf{X}$ must contain either all positive numbers or all negative numbers. In particular, it cannot contain any 0 (corresponding to the empty clause). Thus $\mathbf{X} \nvdash \mathbf{0}$. Since $\mathbf{X} = reduce^{n-1}(\mathbf{A})$, where n is the number of variables, thus by Lemma 1, $\mathbf{A} \nvdash \mathbf{0}$. ■

With this theorem, we can derive the following more general one:

[5] Admittedly this is not a very formal proof. But I hope the argument is convincing enough that a formal proof using induction is not necessary.

Corollary 4 *Let S be a set of clauses, and C a clause. If S does not entail C, then there is a coefficient matrix $\mathbf{A}$ for S such that for any coefficient vector $\mathbf{c}$ for C, $\mathbf{A} \vdash \mathbf{c}$ does not hold.*

Proof: Suppose S does not entail C, then $S' = S \cup \{\{\hat{l}\} \mid l \in C\}$, where $\hat{l}$ is the complement of l, is a consistent set of clauses. Thus by Theorem 3, there is a matrix $\mathbf{B}$ for S' such that $S' \vdash \mathbf{0}$ does not hold. Now let $\mathbf{A}$ be the matrix obtained from $\mathbf{A}$ by eliminating the rows that are not in S. Then $\mathbf{A}$ is an matrix for S. For any coefficient vector $\mathbf{c}$ for C, $\mathbf{A} \vdash \mathbf{c}$ cannot hold, for otherwise, $\mathbf{B} \vdash \mathbf{c}$ holds, and thus $\mathbf{B} \vdash \mathbf{0}$ as $\mathbf{B}$ contains vectors for $\{\hat{l}\}$ for each $l \in C$. ■

If S is consistent, and we have found a coefficient matrix $\mathbf{A}$ sucht that $\mathbf{A} \nvdash \mathbf{0}$, we can then use the following result to find a satisfying assignment in weak polynomial time.

Proposition 2 *If $\mathbf{A} \nvdash \mathbf{0}$, the for each i, either $\mathbf{A}(i) \nvdash \mathbf{0}$ or $\mathbf{A}(-i) \nvdash \mathbf{0}$, where $\mathbf{A}(i)$ ($\mathbf{A}(-i)$) is the matrix obtained from $\mathbf{A}$ by first deleting all rows whose ith elements are positive (negative), and then delete the ith column.*

Theorem 5 *Let S be a set of clauses. The following statements are equivalent*

1. *S is consistent;*
2. *for some coefficient matrix $\mathbf{A}$ of S, $\mathbf{A} \nvdash \mathbf{0}$;*
3. *for some coefficient matrix $\mathbf{A}$ of S, there does not exist a non-negative not all zero column vector $\mathbf{y}$ such that $\mathbf{A^T y} = \mathbf{0}$;*
4. *for some coefficient matrix $\mathbf{A}$ of S, and some column vector $\mathbf{z}$, $\mathbf{Az} > \mathbf{0}$.*

Proof: We only need to prove the equivalence of the last two. Notice that there does not exist a non-negative not all zero column vector $\mathbf{y}$ such that $\mathbf{A^T y} = \mathbf{0}$ means that the following linear system is not feasible:

$$\begin{aligned} &\mathbf{A^T y} = \mathbf{0}, \\ &y_1 + \cdots + y_m > 0, \\ &y_i \geq 0, \;\; 1 \leq i \leq m \end{aligned}$$

where m is the dimension of $\mathbf{y}$, which is the same as the number of columns of $\mathbf{A}$. Now let $\mathbf{B}$ be the result of adding an all 1 column to $\mathbf{A}$: $\mathbf{B} = (1 \; \mathbf{A})$, and consider the following system:

$$\begin{aligned} &\mathbf{B^T y} = \mathbf{I_1}, \\ &y_i \geq 0, \;\; 1 \leq i \leq m \end{aligned}$$

were I_1 is the vector whose first element is 1 and the rest is 0. Clearly, if there is a solution to the one about $\mathbf{B}$, then there is one to the one about $\mathbf{A}$. Now suppose there is a solution $\mathbf{y}$ to the system about $\mathbf{A}$. Let $a = 1/\sum_i y_i$, and $y'_i = y_i/a$. Then $\mathbf{y}'$ is a solution to the system about $\mathbf{B}$. This means that the two systems are equivalent.

By Farkas' lemma, the system about $\mathbf{B}$ does not have a feasible solution iff the following system has a feasible solution:

$$\mathbf{Bv} \geq 0,$$
$$v_1 < 0.$$

This is equivalent to the following system has a feasible solution:

$$\mathbf{Az} > 0.$$

■

More generally, we have

Proposition 3 *Let S be a set of clauses, C a unary clause, and $\mathbf{c}$ the standard coefficient vector of C. The following statements are equivalent*

1. *$S \not\models C$;*
2. *for some coefficient matrix $\mathbf{A}$ of S, $\mathbf{A} \not\vdash \mathbf{c}$;*
3. *for some coefficient matrix $\mathbf{A}$ of S, there does not exist a non-negative column vector $\mathbf{y}$ such that $\mathbf{A^T y} = \mathbf{c^T}$;*
4. *for some coefficient matrix $\mathbf{A}$ of S, and some column vector $\mathbf{z}$, $\mathbf{Az} \geq \mathbf{0}$ and $\mathbf{cz} < 0$.*

Proof: Notice that for a unary clause C, $\mathbf{A}$ can derive a coefficient vector for C iff it can derive the standard coefficient vector of C.

The last two statements are equivalent by Farkas' lemma. ■

4 Some classes of satisfiable sets of clauses

Theorem 3 yields infinite classes of satisfiable sets of clauses and furthermore by Proposition 1, the membership of these classes can be checked in polynomial time. The following is one example:

$$\{S \mid \text{ the standard matrix of } S \text{ does not entail } \mathbf{0}\}.$$

One particularly interesting class is about sets of clauses that have the same or less number of clauses than that of variables.

Recall that the rank of a matrix is the maximal number of independent rows of the matrix. An $m \times n$ matrix is said to have full rank if its rank is m. Clearly, this is possible only if $m \leq n$, and when $m = n$, the matrix has a reverse.

By Theorem 3, if a set of clauses has a full rank coefficient matrix, then it is satisfieable. We now show that we can do better here: in this case, a satisfiable assignment can be found in polynomial time.

In the following, if S is a set of clauses, and x a variable, then we use $S(x = 1)$ to denote the set of clauses obtained from S by deleting all clauses that mention x positively, and deleting $\neg x$ from all other clauses. Similarly, $S(x = 0)$ denotes the set of clauses obtained from S by deleting all clauses that mention x negatively, and deleting x from all other clauses.

We can do similar operations with matrix. Let $\mathbf{x}$ be the tuple of variables in S and $\mathbf{A}$ a coefficient matrix. For each i, $\mathbf{A}(x_i = 1)$ ($\mathbf{A}(x_i = 0)$) is the matrix obtained from $\mathbf{A}$ by deleting all rows that have positive (negative) values in their ith column and then deleting the ith column. Clearly, if $\mathbf{A}$ is a matrix for S, then $\mathbf{A}(x_i = 1)$ ($\mathbf{A}(x_i = 0)$) is a matrix for $S(x_i = 1)$ ($S(x_i = 0)$).

Theorem 6 *Let S be a set of clauses and $\mathbf{A}$ any matrix for it. If $\mathbf{A}$ has full rank, then S is consistent. Furthermore, for any variable x_i, either the matrix $\mathbf{A}(x_i = 1)$ or the matrix $\mathbf{A}(x_i = 0)$ has full rank.*

Proof: If S is not consistent, then by Lemma 1, one of the rows of $\mathbf{A}$ can be reduced to 0, a contradition with the assumption that $\mathbf{A}$ has full rank.

If none of $\mathbf{A}(x_i = 1)$ and $\mathbf{A}(x_i = 0)$ has full rank, then there is a row in $\mathbf{A}(x_i = 1)$ and one in $\mathbf{A}(x_i = 0)$ that can be transformed to 0. Thus the same transformation when applied to $\mathbf{A}$ must yield two rows that are all zero except the ith column. Furthermore, the ith column of one row must be positive (the one corresponding to $\mathbf{A}(x_i = 0)$), and the other must be negative. This means that an all zero row can be obtained from $\mathbf{A}$, a contradiction. ■

Thus if we can find a full rank coefficient matrix $\mathbf{A}$ for S, then we can use the following algorithm to find a satisfying assignment: pick a variable x, compute $\mathbf{A}(x = 1)$ and $\mathbf{A}(x = 0)$, choose the one that has full rank, make the corresponding assignment for x, and then continue this way recursively.

Notice that the problem of checking if a set of clauses with equal number of variables and clauses is satisfiable is still NP-complete. This is because given any set S of clauses, if the number of variables is smaller than the number of clauses in S, then we can introduce new variables $z_1, ..., z_k$ and add to S a single clause $z_1 \vee \cdots \vee z_k$. It is easy to see that the original set of clauses is satisfiable iff the new set of clauses is satisfiable.

Theorem 6 transfers the problem of satisfiability to that of finding a matrix with full rank, which in general must be NP-hard as well. There is one case, however, we can construct a matrix with full rank explicitly.

Theorem 7 *Let S be a set of m clauses with n variables. If $m \leq n$, and there is an ordering $\mathbf{x}$ of variables in S, and an ordering of clauses in S: $S = \{C_1, ..., C_n\}$ such that x_i occurs in the ith clause in S for any $1 \leq i \leq m$, then S is satisfiable, and there is a polynomial time algorithm to compute a full rank matrix $\mathbf{A}$ for S.*

Proof: We construct $\mathbf{A}$ one column at a time, from left to right. The first column are coefficients for x_1, and are set to be their standard values. Suppose all columns before k have been constructed. For column k, let A_{hk} be the standard value for any $h \geq k$, and for $h < k$, any legal value such that Gauss-Jordan procedure of converting

a matrix to an upper triangular one will not make the (k, k) element 0. This can be done inductively starting with $h = 1$.

It can be verified that the matrix so constructed is a matrix for S and has full rank. ■

Example 2 Consider the following set of clauses

$$x_1 + x_2 + x_3 - x_4,$$
$$-x_1 - x_2 + x_3,$$
$$x_1 + x_2 - x_3 - x_4,$$
$$-x_1 + x_2 - x_3 + x_4.$$

The first column of $\mathbf{A}$ is

$$\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

For the second column, A_{12} needs to be a positive value such that when the first row is added to the second, the $(2, 2)$ cell does not become 0. This means that it can be any positive number other than 1. Let's make it 2. We thus have

$$\begin{pmatrix} 1 & 2 \\ -1 & -1 \\ 1 & 1 \\ -1 & 1 \end{pmatrix}$$

For the 3rd column, A_{13} needs to be a value such that when the 3rd row is subtracted from the first row, the $(3, 3)$ cell does not become 0. This can never be the case for any positive value, so we can use the standard value for A_{13}. And A_{23} needs to be a value such that the $(3, 3)$ element does not become 0 after each of the following operations:

- multiply the first one by -1, and then add it to the third row;
- multiply the first one by -1, then add it to the third row, add the first row to the second row, and then add the second row to the thrid row.

This would mean that we cannot use the standard value for it. So we make it 2 and have

$$\begin{pmatrix} 1 & 2 & 1 \\ -1 & -1 & 2 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \end{pmatrix}$$

With similar calculations, we can have the following matrix which is full rank:

$$\begin{pmatrix} 1 & 2 & 1 & -2 \\ -1 & -1 & 2 & 0 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{pmatrix}$$

It can be transformed to the following upper triangular matrix using the Gauss-Jordan procedure:

$$\begin{pmatrix} 1 & 2 & 1 & -2 \\ 0 & 1 & 3 & -2 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -4 \end{pmatrix}$$

5 Reasoning with linear combinations of clauses: constraint propagation

Theorem 3 reduces the problem of satisfiability to that of finding a coefficient matrix that does not entail $\mathbf{0}$. We now consider how to keep track of a derivation to reason about the the maximal and minimal values of a derived clause.

To do this, we introduce names for linear combinations. Specifically, if S is a set of clauses and $\mathbf{A}$ a coefficient matrix for it, then we introduce a name for the linear combination of each row, so we have the following set of equations:

$$\mathbf{Ax} = \mathbf{r}$$

where $\mathbf{x}$ is the vector of variables in S, and $\mathbf{r}$ a vector of names (variables), one for each clause[6]. So if $\mathbf{c^i}$ is the ith row of $\mathbf{A}$ representing the ith clause C_i in S, then r_i is a name for the clause and stands for the linear combination $\mathbf{c^i x}$. In the following, we call C_i the corresponding clause, and $\mathbf{c^i}$ the corresponding coefficient vector of r_i. The variables in C_i are also called variables in r_i.

Given a name r that stands for the coefficient vector $\mathbf{c}$, and a variable assignment σ on the variables in $\mathbf{x}$, the value of r, written $\sigma(r)$, is

$$\sigma(r) = \mathbf{c}\sigma(\mathbf{x}),$$

where $\sigma(\mathbf{x}) = (\sigma(x_1), ..., \sigma(x_n))$. More generally, given a tuple of names $\mathbf{r} = (r_1, ..., r_m)$, the value of a variable assignment σ on a linear combination of names $\beta = \mathbf{wr}$, written $\sigma(\beta)$, is

$$\sigma(\beta) = \sigma(\mathbf{wr}) = \mathbf{w}\sigma(\mathbf{r}),$$

where $\sigma(\mathbf{r}) = (\sigma(r_1), ..., \sigma(r_m))$.

If each r_i stands for the coefficient vector $\mathbf{c^i}$, $1 \leq i \leq m$, then the linear combination $\mathbf{wr}$ stands for the vector $\mathbf{wA}$, where $\mathbf{A}$ is the matrix whose ith row vector is $\mathbf{c^i}$. Thus $\mathbf{A}$ entails $\mathbf{c}$ iff there is a positive linear combination of rows in $\mathbf{A}$ that stands for $\mathbf{c}$. In particular, $\mathbf{A} \vdash \mathbf{0}$ if there is a positive linear combination of names that stands for the empty clause, where a linear combination is positive if all the coefficients are nonnegative and at least one of them is positive.

Definition 1 *The minimal and maximal values of a linear combination β of names are defined as follows. We say that a name occurs in β if the coefficient of the name is*

[6]We call the elements in $\mathbf{r}$ names instead of variables to differentiate them from variables in S.

non-zero in β. The minimal (maximal) value of β, written $min(\beta)$ ($max(\beta)$), is the minimal (maximal) value in the set

$$\{\sigma(\beta) \mid \sigma \textit{ satisfies all clauses denoted by the names that occur in } \beta\}.$$

Similarly, the minimal and maximal values of a linear combination $\alpha = \mathbf{cx}$ of variables are defined as follows: the minimal (maximal) value of α, written $min(\alpha)$ ($max(\alpha)$), is the minimal (maximal) value in the set

$$\{\sigma(\alpha) \mid \sigma \textit{ satisfies the clause that corresponds to } \alpha\}.$$

The following proposition is easy to prove and shows the usefulness of keeping track of minimal and maximal values.

Proposition 4 *Let S be a set of clauses and $\mathbf{A}$ a matrix for it. Let $\mathbf{r}$ be a vector of names, one for each row of $\mathbf{A}$. Suppose β is a linear combination of names and stands for the coefficient vector $\mathbf{c}$. Suppose that the clause of $\mathbf{c}$ is C, $\alpha = \mathbf{cx}$. Then*

1. *if $min(\beta) \geq min(\alpha)$, then S logically entails C;*
2. *if $max(\beta) < min(\alpha)$, then S logically entails $\neg C$;*
3. *if either $min(\beta) > max(\alpha)$ or $max(\beta) < min(\alpha)$, then S is inconsistent.*

Proof: We prove the first case. The others are similar. Suppose S does not entail C, then there is an assignment σ that satisfies S and $\neg C$. By definition, $min(\beta) \leq \sigma(\beta)$. Since β stands for α. Thus $\sigma(\beta) = \sigma(\alpha)$. But σ does not satisfy C, so $\sigma(\alpha) < min(\alpha)$. Thus $min(\beta) < min(\alpha)$. ■

The following proposition summaries some simple rules that are useful for computing the minimal and maximal value of a linear combination of names.

Proposition 5 *In the following, β and β_i, $i = 1, 2$, are linear combinations of names, and S, S_i, $i = 1, 2$, are sets of clauses.*

1. *If r is a name for $\mathbf{c}$ which is a coefficient vector for a clause C, then $min(r)$ is the minimal value $\mathbf{c}\sigma(\mathbf{x})$ among those σ that assigns exactly one of the literals in C true, and $max(r)$ is $\mathbf{c}\sigma(\mathbf{x})$ for the σ that assigns 1 to x_i if it is in C and 0 otherwise.*
2. *If c is positive, then $min(c\beta) = c \times min(\beta)$, and $max(c\beta) = c \times max(\beta)$.*
3. *If c is negative, then $min(c\beta) = c \times max(\beta)$, and $max(c\beta) = c \times min(\beta)$.*
4. *$min(\beta_1 + \beta_2) \geq min(\beta_1) + min(\beta_2)$, and $max(\beta_1 + \beta_2) \leq max(\beta_1) + max(\beta_2)$.*
5. *Let S_i be the set of clauses that occur in β_i, $i = 1, 2$. If S_1 and S_2 do not have common variables, then $min(\beta_1 + \beta_2) = min(\beta_1) + min(\beta_2)$, and $max(\beta_1 + \beta_2) = max(\beta_1) + min(\beta_2)$.*

In general, computing $min(\beta)$ and $max(\beta)$ is hard. Our following algorithm needs exponential space in the worst case. It computes the set of "minimal assignments" ("maximal assignments") of a linear combination inductively.

Given a clause C, and two variable assignments σ_1 and σ_2, we write $\sigma_1 \leq_C \sigma_2$ if for any i, if $x_i \in C$, then $\sigma_2(x_i) = 1$ whenever $\sigma_1(x_i) = 1$, else if $\neg x_i \in C$, then $\sigma_2(x_i) = 0$ whenever $\sigma_1(x_i) = 0$. The intuition is that if $\sigma_1 \leq_C \sigma_2$, then $\sigma_1(\mathbf{cx}) \leq \sigma_2(\mathbf{cx})$ for any coefficient vector $\mathbf{c}$ for C.

Let β be a linear combination of names. Suppose that β stands for clause C. We say that a variable assignment σ is a minimal assignment of β if it satisfies all clauses denoted by the names in β and there is no assignment σ' that also satisfies all clauses denoted by the names in β but $\sigma' <_C \sigma$. Symmetrically, we can define maximal assignment as well. Since a linear combination β of names stands for a unique clause C, in the following, by $\sigma_1 \leq_\beta \sigma_2$ we mean $\sigma_1 \leq_C \sigma_2$.

Clearly if σ makes β minimal: it is an assignment that satisfies all clauses denoted by names in β such that $\sigma(\beta) = min(\beta)$, then σ is a minimal assignment of β. But the converse is not true in general. A minimal assignment may not always yield the minimal value. However, the number of minimal assignments is typically much smaller than the number of all possible assignments.

Our strategy for computing $min(\beta)$ and $max(\beta)$ is to keep a record of the minimal assignments. We now show how we can update the set of minimal assignments as we build up a linear combinations of names inductively. For the base case, we have

Proposition 6 *If r is a name denoting a clause C, then the minimal assignments of r are those that make exactly one of the literals in C true, and the maximal assignments of r are those that make all of the literals in C true.*

Technically, this means that there are many minimal assignments for a name as variables that are not in the clause can take any values. However, for a clause with k literals, there only k significantly different minimal assignments, those that differ on the variables in the clause. Thus, in the following, if r is, say $\{x_1 + x_2\}$, we'll just say that r has two minimal assignments $\{x_1, \neg x_2\}$ and $\{\neg x_1, x_2\}$, no matter how many other variables there are in other clauses. The following are some simple composition rules.

Proposition 7 *Let β, β_i, $i = 1, 2$, be linear combinations of names.*

- *If c is a positive number, then $c\beta$ and β have the same minimal and maximal assignments.*
- *If c is a negative number, then the minimal (maximal) assignments of $c\beta$ are the maximal (minimal) assignments of β.*
- *Let*

  $$X_i = \{x \mid x \text{ occurs in a clause denoted by a name in } \beta_i\}.$$

  If $X_1 \cap X_2 = \emptyset$, then an assignment is a minimal (maximal) assignment of $\beta_1 + \beta_2$ iff it is a minimal (maximal) assignment of both β_1 and β_2.

For example, given the following two clauses represented by their standard linear combinations and their names:

$$\begin{aligned} r_1: &\quad x_1 + x_2 \\ r_2: &\quad x_3 + x_4 \end{aligned}$$

The set of minimal assignments for $r_1 + r_2$ is

$$\{\{x_1, \neg x_2, x_3, \neg x_4\}, \{x_1, \neg x_2, \neg x_3, x_4\}, \{\neg x_1, x_2, x_3, \neg x_4\}, \{\neg x_1, x_2, \neg x_3, x_4\}\}.$$

In general, the minimal assignments of $\beta_1 + \beta_2$ are computed according to the following proposition.

Proposition 8 *Let $\beta_i = \mathbf{w^i r}$. Suppose that $\beta_1 + \beta_2$, which is $(\mathbf{w^1} + \mathbf{w^2})\mathbf{r}$, does not eliminate any names, that is, for each i, $w_i^1 + w_i^2 = 0$ iff $w_i^1 = 0$ and $w_i^2 = 0$. We have that if σ is a minimal assignment of $\beta_1 + \beta_2$, then there are two minimal assignments σ_1 and $sigma_2$ for β_1 and β_2, respectively, such that*

- *$\sigma_1 \leq_{\beta_1} \sigma$ and $\sigma_2 \leq_{\beta_2} \sigma$, and*
- *there does not exist another assignment σ' such that σ' satisfies all clauses denoted by the names in $\beta_1 + \beta_2$, $\sigma' <_\beta \sigma$, $\sigma_1 \leq_{\beta_1} \sigma'$, and $\sigma_2 \leq_{\beta_2} \sigma'$.*

Thus if Min_i is the set of minimal assignment of β_i, $i = 1, 2$, then the following procedure will return the set of minimal assignments of $\beta_1 + \beta_2$:

1. $\Sigma = \emptyset$.
2. for each $\sigma_i \in Min_i$, $i = 1, 2$, add to Σ the following variable assignment σ: $\sigma_1 \leq_{\beta_1} \sigma$ and $\sigma_2 \leq_{\beta_2} \sigma$, and there does not exist another assignment σ' such that σ' satisfies all clauses denoted by the names in $\beta_1 + \beta_2$, $\sigma' <_\beta \sigma$, $\sigma_1 \leq_{\beta_1} \sigma'$, and $\sigma_2 \leq_{\beta_2} \sigma'$. This is like computing a least upper bound of two elements in a lattice.
3. for each $\sigma \in \Sigma$, if there is another $\sigma' \in \Sigma$ such that $\sigma' <_\beta \sigma$, then delete σ from Σ.
4. return Σ.

Notice that here we check for minimality in the end. In actual implementation, this can be done dyamically as new σ is added to Σ. Notice a similar algorithm works for computing maximal assignments.

Example 3 Consider

$$\begin{aligned} r_1: &\quad x_1 + x_2, \\ r_2: &\quad -x_3 - x_2. \end{aligned}$$

The set of minimal assignments for r_1 is $\{\{x_1, \neg x_2\}, \{\neg x_1, x_2\}\}$, and for r_2 is $\{\{x_2, \neg x_3\}, \{\neg x_2, x_3\}\}$. So the above algorithm produces the following assignments for $r_1 + r_2$, which stands for the linear combination $x_1 - x_3$ and clause $x_1 \vee \neg x_3$:

- $\{x_1, \neg x_2\}$ and $\{x_2, \neg x_3\}$ yield $\sigma_1 = \{x_1, \neg x_3\}$.
- $\{x_1, \neg x_2\}$ and $\{\neg x_2, x_3\}$ yield $\sigma_2 = \{x_1, x_3\}$.
- $\{\neg x_1, x_2\}$ and $\{x_2, \neg x_3\}$ yield $\sigma_3 = \{\neg x_1, \neg x_3\}$.
- $\{\neg x_1, x_2\}$ and $\{\neg x_2, x_3\}$ yield $\sigma_2 = \{x_1, x_3\}$ and $\sigma_3 = \{\neg x_1, \neg x_3\}$ as $\{\neg x_1, x_3\}$ does not satisfy $\{r_1, r_2\}$.

Since $\sigma_3 <_{r_1+r_2} \sigma_1$, thus the minimal assignments for $r_1 + r_2$ are σ_2 and σ_3, and $min(r_1 + r_2) = 0$.

Example 4 As a complete example, consider the proof of pigeonhole principle which states that $n + 1$ items cannot fit into n pigeonholes. It is well-known that when represented as a set of clauses, a proof by resolution requires exponential steps.

Let x_{ij} stands for putting item i into pigeonhole j. The clauses, represented by their standard linear combinations, are as follows:

$$r_i : \quad x_{i1} + x_{i2} + \cdots + x_{in}, \quad \text{for } 1 \leq i \leq n+1 \tag{6}$$

$$r_{ijk} : \quad \neg x_{ik} + \neg x_{jk}, \quad \text{for } 1 \leq i < j \leq (n+1) \text{ and } 1 \leq k \leq n \tag{7}$$

This set of clauses is inconsistent, and as we mentioned, the resolution proof of this fact requires exponential steps.

It is easy to see that

$$\sum_{1 \leq i \leq n+1} nr_i + \sum_{1 \leq k \leq n} \sum_{1 \leq i \leq n} \sum_{i < j \leq n+1} r_{ijk} = 0. \tag{8}$$

We'll show that the minimal value of the linear combination in the left side is greater than 0, thus a contradiction. We'll show this using Proposition 5 and the above algorithm for inductively computing minimal assignments. Our proof takes only polynomial number of steps.

Compute first the minimal assignments and minimal values of the new variables:

- $min(r_i) = 1$,
- an assignment of r_i is minimal iff exactly one of the variables in it is true,
- $min(r_{ijk}) = -1$,
- an assignment of r_{ijk} is minimal iff exactly one of the variables in it is true.

To compute the minimal value of $\sum_{1 \leq i \leq n+1} nr_i$, observe that it stands for the following linear combination of variables:

$$\sum_{1 \leq i \leq n+1} \sum_{1 \leq j \leq n} nx_{ij}.$$

Since the variables in r_i and r_j are distinct for any $i \neq j$, thus by Proposition 5, $min(\sum_{1 \leq i \leq n+1} nr_i) = n(n+1)$.

We now compute the minimal value of $\sum_{1\leq k\leq n}\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}$ inductively using the above algorithm. Consider $r_{121}+r_{131}$, which stands for the linear combination $-2x_{11}-x_{21}-x_{31}$. The minimal assignments for r_{121} are $\{x_{11},\neg x_{21}\}$ and $\{\neg x_{11},x_{21}\}$, and for r_{131} are $\{x_{11},\neg x_{31}\}$ and $\{\neg x_{11},x_{31}\}$. Using the algorithm above:

- $\{x_{11},\neg x_{21}\}$ and $\{x_{11},\neg x_{31}\}$ yield $\sigma_1=\{x_{11},\neg x_{21},\neg x_{31}\}$.
- $\{x_{11},\neg x_{21}\}$ and $\{\neg x_{11},x_{31}\}$ yield $\sigma_2=\{\neg x_{11},\neg x_{21},x_{31}\}$.
- $\{\neg x_{11},x_{21}\}$ and $\{x_{11},\neg x_{31}\}$ yield $\sigma_3=\{\neg x_{11},x_{21},\neg x_{31}\}$.
- $\{\neg x_{11},x_{21}\}$ and $\{\neg x_{11},x_{31}\}$ yield $\sigma_4=\{\neg x_{11},x_{21},x_{31}\}$.

But $\sigma_4<_\beta\sigma_2$ and $\sigma_4<_\beta\sigma_3$, where β is the linear combination $r_{121}+r_{131}$. Thus the minimal assignments for $r_{121}+r_{131}$ are σ_1 and σ_4. Continuing this way, we can compute the minimal assignments for $\sum_{i<j\leq n+1} r_{ijk}$, which stands for $\sum_{i<j\leq n+1}-x_{ik}-x_{jk}$, and they are

1. those that assign 1 to x_{ik}, and 0 to x_{jk} for any $i<j\leq n+1$, and
2. those that assign 0 to x_{ik}, and 1 to x_{jk} for any $1<j\leq n+1$.

Now consider

$$sum_{1<j\leq n+1}r_{1jk}+\sum_{2<j\leq n+1} r_{2jk}, \tag{9}$$

which stands for

$$\sum_{1<j\leq n+1}-x_{1k}-x_{jk}+\sum_{2<j\leq n+1}-x_{2k}-x_{jk}.$$

Again using the above algorithm:

- $\{x_{1k},\neg x_{2k},...,\neg x_{(n+1)k}\}$ and $\{x_{2k},\neg x_{3k},...,\neg x_{(n+1)k}\}$ yield
$$\sigma_1=\{x_{1k},\neg x_{2k},...,\neg x_{(n+1)k}\}.$$
- $\{x_{1k},\neg x_{2k},...,\neg x_{(n+1)k}\}$ and $\{\neg x_{2k},x_{3k},...,x_{(n+1)k}\}$ yield
$$\sigma_1=\{x_{1k},\neg x_{2k},...,\neg x_{(n+1)k}\},\text{ and }\sigma_2=\{\neg x_{1k},\neg x_{2k},x_{3k},...,x_{(n+1)k}\}.$$
- $\{\neg x_{1k},x_{2k},...,x_{(n+1)k}\}$ and $\{x_{2k},\neg x_{3k},...,\neg x_{(n+1)k}\}$ yield
$$\sigma_3=\{\neg x_{1k},x_{2k},\neg x_{3k}...,\neg x_{(n+1)k}\}.$$
- $\{\neg x_{1k},x_{2k},...,x_{(n+1)k}\}$ and $\{\neg x_{2k},x_{3k},...,x_{(n+1)k}\}$ yield
$$\sigma_4=\{\neg x_{1k},\neg x_{2k},x_{3k}...,x_{(n+1)k}\}.$$

But $\sigma_3 <_\beta \sigma_4$, where β is the clause of the linear combination. Thus the minimal assignments for (9) are σ_1, σ_2, and σ_3 above. Again continuing this, we can compute the minimal assignments of $\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}$, and they are the ones that assign exactly one of the variables in $\{x_{ik}|1 \leq i \leq n+1\}$ true. Thus $min(\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}) = -n$. If $k_1 \neq k_2$, then $\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} -x_{ik_1} - x_{jk_1}$ and $\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} -x_{ik_2} - x_{jk_2}$ do not share variables, thus

$$min(\sum_{1\leq k\leq n}\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}) = -n^2.$$

So

$$min(\sum_{1\leq i\leq n+1} nr_i + \sum_{1\leq k\leq n}\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}) \geq n(n+1) - n^2 > 0.$$

But as (8) shows

$$\sum_{1\leq i\leq n+1} nr_i + \sum_{1\leq k\leq n}\sum_{1\leq i\leq n}\sum_{i<j\leq n+1} r_{ijk}$$

stands for 0, the empty clause. This means that the set of clauses (6) and (7) is inconsistent. Notice that this proof takes only polynomial number of steps in n.