The Resolution Algorithm

Introduction

In this lecture we introduce the Resolution algorithm for solving instances of the NP-complete CNF-SAT decision problem. Although the algorithm does not run in polynomial time for all instances, it does offer the advantages of

1. being correct and terminating on all instances of CNF-SAT,
2. offering a formal proof in the case that an instance of CNF-SAT is unsatisfiable,
3. being easy to implement, and
4. having a reputation for efficiency when applied to practical instances of CNF-SAT (as opposed to some provably-hard theoretical instances).

It should be noted that there has been much recent work performed in the development of randomized local-search heuristics for solving CNF-SAT instances. One such heuristic is Bart Selman's WalkSAT. Although these heuristics do not offer the above advantages of 1. and 2., they do possess advantages 3. and 4., and have been shown to outperform Resolution in several studies. The interested reader should consult Chapter 5 of the "Handbook of Constraint Programming", Elsevier Press, 2008.

Algorithm Description

The Resolution algorithm takes as input a set $\mathcal{C}$ of clauses and returns true iff $\mathcal{C}$ is satisfiable. It does this by performing a sequence of **resolution steps**, where each step consists of identfying two clauses $c_1, c_2 \in \mathcal{C}$ of the form $c_1 = P \vee x$, and $c_2 = Q \vee \overline{x}$, and then adding to $\mathcal{C}$ the **resolvent** clause $c = P \vee Q$, where P and Q are disjunctions of literals, and x is a variable, called the **resolved variable**. Moreover, the pair (c_1, c_2) is called a **resolution pair**, while $\text{res}(c_1, c_2)$ denotes the resolvent of the pair.

Proposition 1. Given a set of clauses $\mathcal{C}$, if clause c is the resolvent of clauses $c_1, c_2 \in \mathcal{C}$, then $\mathcal{C}$ is satisfiable iff $\mathcal{C} + c$ is satisfiable.

Proof of Proposition 1. Given $c_1, c_2 \in \mathcal{C}$, where $c_1 = P \vee x$, and $c_2 = Q \vee \overline{x}$, and $c = P \vee Q$ is a resolvent of c_1 and c_2, it is an exercise to show that

$$(P \vee x) \wedge (Q \vee \overline{x})$$

is logically equivalent to

$$(P \vee x) \wedge (Q \vee \overline{x}) \wedge (P \vee Q).$$

In other words, any assignment that satisfies the former formula will also satisfy the latter formula and vice versa. Therefore, $\mathcal{C}$ is satisfiable iff $\mathcal{C} + c$ is satisfiable. $\square$

Proposition 2. Given CNF-SAT instance $\mathcal{C}$ with $c_1, c_2 \in \mathcal{C}$ and $c_1 \subseteq c_2$, then $\mathcal{C}$ is satisfiable iff $\mathcal{C} - c_2$ is satisfiable.

Proof of Proposition 2. Suppose $\mathcal{C}$ is satisfiable. Then $\mathcal{C} - c_2$ is also satisfiable. Conversely, if $\mathcal{C} - c_2$ is satisfiable via assignment a, then a satisfies c_1, and hence must also satisfy c_2. Therefore, $\mathcal{C}$ is satisfiable. $\square$

Whenever there are two clauses $c_1, c_2 \in \mathcal{C}$ for which $c_1 \subseteq c_2$, then c_2 is said to be **subsumed** by c_1, and may be removed from $\mathcal{C}$ without affecting the (un)satisfiability of $\mathcal{C}$. Therefore, a resolvent c need only be added to $\mathcal{C}$ in case it is not subsumed by any clause that is already in $\mathcal{C}$. Moreover, upon adding resolvent c to $\mathcal{C}$, we may subtract from $\mathcal{C}$ any clause that is subsumed by c. Thus, we may always assume that $\mathcal{C}$ is **subsumption free**, meaning that it does not contain two clauses c_1 and c_2 for which $c_1 \subseteq c_2$. Henceforth, all CNF-SAT instances $\mathcal{C}$ are assumed to be subsumption free.

Ordering the resolution steps

Since a set of clauses $\mathcal{C}$ may possess more than one resolution pair, we define a linear ordering on such pairs that informs the Resolution algorithm on which pair to resolve next. To accomplish this we first define linear orderings for literals and clauses.

To define a linear ordering on literals, first assume that the clause variables come from the set $V = \{x_1, \ldots, x_n\}$, and let $\overline{V}$ denote the set of negations of variables that appear in V. Next, define the one-to-one function order : $V \cup \overline{V} \to I$ that assigns each literal a unique integer, where $\text{order}(x_i) = i$ and $\text{order}(\overline{x}_i) = -i$. Then literal l_1 precedes literal l_2 iff $\text{order}(l_1) < \text{order}(l_2)$.

Now let $c_1 \neq c_2$ be two clauses. Then we have $c_1 < c_2$ in case $c_1 \subset c_2$ and $c_2 < c_1$ in case $c_2 \subset c_1$. Hence, assume both $c_1 \not\subset c_2$ and $c_2 \not\subset c_1$ are true. Then $c_1 < c_2$ iff $\text{order}(l_1) < \text{order}(l_2)$, where, e.g., $l_1 \in c_1$ is the least literal that is not a member of c_2. As an example,

$$c_1 = (x_2, \overline{x}_3, x_5) < c_2 = (x_3, x_5, x_7),$$

since $\text{order}(x_2) = 2 < \text{order}(x_3) = 3$.

Finally, we may now define an ordering on resolution pairs based on the size (i.e. number of literals) of their respective resolvents. Indeed, letting $\text{size}(c)$ denote the number of distinct literals in c, we have $(c_1, c_2) < (c_3, c_4)$ iff either i) $\text{size}(\text{res}(c_1, c_2)) < \text{size}(\text{res}(c_3, c_4))$ or ii) $\text{size}(\text{res}(c_1, c_2)) = \text{size}(\text{res}(c_3, c_4))$ and $\text{res}(c_1, c_2) < \text{res}(c_3, c_4)$, or iii) $\text{res}(c_1, c_2) = \text{res}(c_3, c_4)$ and the resolving variable for (c_1, c_2) has lower index than that of (c_3, c_4).

Algorithm termination

There are two possibilities for how the Resolution algorithm terminates: via a refutation step, or via clause saturation.

In the first case, there is a step for which $c_1 = x$ and $c_2 = \overline{x}$, in which case P and Q are empty, and hence the resolvent $P \vee Q$ is the **empty clause**, and is denoted by `F`, since it is logically equivalent to `false`. Such a step is called a **refutation step**, and the algorithm is said to have performed a **refutation proof**. In this case $\mathcal{C}$ is unsatisfiable since, by Proposition 1, any assignment that satisfies $\mathcal{C}$ must also satisfy a resolvent (in this case `F`) derived from $\mathcal{C}$.

In the second case, for each possible pair of clauses $c_1 = P \vee x$, and $c_2 = Q \vee \overline{x}$, it is the case that $P \vee Q$ is subsumed by a member of $\mathcal{C}$, in which case no new resolvents (including `F`) can be added to $\mathcal{C}$. In this case $\mathcal{C}$ is said to be **saturated**. In the next section we prove that saturated instances must be satisfiable. Conversely, if $\mathcal{C}$ is satisfiable, then, by Proposition 1, it cannot derive the empty clause. Moreover, it must become saturated since there are at most 3^n different possible resolvents (why?).

Finding a satisfying assignment

Suppose subsumption-free CNF-SAT instance $\mathcal{C}$ is saturated. Then the following method can be used to determine a satisfying assignment for $\mathcal{C}$. While $\mathcal{C}$ is non-empty, choose a clause $c \in \mathcal{C}$ and a literal $l \in c$ and assign l to `true`. Remove c from $\mathcal{C}$ and any other clause from $\mathcal{C}$ that contains l. For any clause $c' \in \mathcal{C}$ that contains $\bar{l}$, remove $\bar{l}$ from c'. Note: when removing $\bar{l}$ from c', we may rest assured that c' does not become empty. Otherwise, it must be the case that $c' = (\bar{l})$, in which case c and c' could be resolved to form clause c_2 that subsumes c, which contradicts the assumption that $\mathcal{C}$ is both saturated and subsumption-free.

Claim: after i) removing clauses from $\mathcal{C}$ that contain l, ii) removing literal $\bar{l}$ from all remaining clauses, and iii) removing any clause $c_2 \in \mathcal{C}$ that is subsumed by another clause $c_1 \in \mathcal{C}$, the resulting set of clauses is saturated.

Proof of Claim. Suppose $\mathcal{C}$ is no longer saturated. Then there are two clauses $c_1, c_2 \in \mathcal{C}$ that can be resolved to produce the resolvent c that is not subsumed by any clause in $\mathcal{C}$. But prior to the removal of clauses and literals described in i-iii, c *was* subsumed by some clause c'. Moreover, if c' remains in $\mathcal{C}$, then the only possible change to c' is the removal of $\bar{l}$ from c'. But for any literal l, and two claues c_1 and c_2 with $c_1 \subseteq c_2$, we have both $c_1 - l \subseteq c_2$, and $c_1 - l \subseteq c_2 - l$. Thus, $c' - \bar{l}$ still subsumes c, and so it must be the case that c' was removed from $\mathcal{C}$. Moreover, in the case that c' contains l, since c' subsumes c, either c_1 or c_2 must have contained l, contradicting the fact that both clauses remain in $\mathcal{C}$. Finally, if c' was removed because of being subsumed by $c'' \in \mathcal{C}$, then c'' subsumes c by transitivity.

It follows from the above claim that either $\mathcal{C}$ will be empty (in which case any remaining unassigned variables may be arbitrarily assigned), or the process of selecting clause c, along with a satisfying literal l of c, can be successfully repeated until $\mathcal{C}$ becomes empty. Now suppose that the above process is repeated until the set of clauses becomes empty,. Then letting L denote the set of literals that were selected to be true at each stage, it follows that L is a consistent set, and hence the assignment a_L induced by L is well defined, and satisfies the original saturated set $\mathcal{C}$. To see this, consider $c \in \mathcal{C}$ from the original saturated set. If c was removed from $\mathcal{C}$ because $l \in c$ and l was set to `true` during one of the rounds, then $l \in L$ and a_L satisfies c. On the other hand, suppose c were removed because of being subsumed by c'. Then it is an exercise to show that there must be a clause d for which a_L satisfies d, and for which $d \subseteq c$. Thus, a_L satisfies c.

Example 1. Use the above claim to construct a tree that enumerates all satisfying assignments for the saturated set of clauses

$$\{(x_1, \overline{x}_2), (x_1, x_3), (\overline{x}_2, x_3)\}.$$

Resolution Examples

The following examples of the Resolution algorithm will use the following method for determining the next resolution step. Begin by placing all clauses in a list L, and perform the following until either L is saturated or a refutation is found. For each clause $c_1 \in L$, find all clauses $c_2 \in L$ that come after c_1, and for which c_1 can be resolved with c_2, but has yet to be resolved with c_2. Append the resolvent c of c_1 and c_2 to L provided it is not subsumed by an existing clause in L. Remove all clauses from L that are subsumed by c. Repeat until L has been cycled through at least once without any new clauses being added.

Example 2. Use the Resolution algorithm to determine if the following set of clauses is satisfiable.

$$\mathcal{C} = \{(x_2), (\overline{x}_2, x_3), (\overline{x}_2, \overline{x}_1), (x_1, \overline{x}_2, \overline{x}_3)\}.$$

Example 3. Use the Resolution algorithm to determine if the following set of clauses is satisfiable.

$$\mathcal{C} = \{(a, b), (c, \overline{e}), (\overline{b}, c), (\overline{d}, \overline{a}), (\overline{a}, \overline{b}), (\overline{c}, \overline{d}), (d, e), (\overline{c}, b)\}.$$

An Exponential Lower Bound for the Running Time of Resolution

In this section we prove that, for every n, there exists an unsatifiable set of clauses $\mathcal{C}_n$ for which the Resolution algorithm requires c^n steps before a refutation step can be found, where $c > 1$ is a constant. This proves that the Resolution algorithm has worst-case exponential running time.

The following definition will prove useful in establishing the lower-bound result. Let $\mathcal{C}$ be an unstatisfiable set of clauses, and let $R(\mathcal{C})$ denote the set of resolvents (incuding $\mathcal{C}$ itself) that are produced by the Resolution algorithm on input $\mathcal{C}$. Then the **refutation proof graph with respect to** $\mathcal{C}$ is the directed graph whose vertex set is $R(\mathcal{C})$, and whose edge set is the set of edges (c_1, c_2), where $c_1, c_2 \in R(\mathcal{C})$, and c_1 was resolved with some other clause c to produce resolvent c_2. Note: since G is directed c_1 is called the **parent** of c_2, and c_2 is the **child** of c_1. It thus follows that, for every clause $c \in R(\mathcal{C})$, either $c \in \mathcal{C}$ and c has no parents, or c is a resolvent, and has two parents.

Without loss of generality, we henceforth assume that, for every vertex v of a proof graph, there is a path from v to the empty clause. In other words, we may remove any part of the graph that does not contribute to the derivation of empty clause `F`.

Example 4. Show the proof graph for the unsatisfiable set of clauses from Example 2.

To prove the exponential lower bound for the running time of Resolution, we prove that, for each $n \geq 1$, there is an unsatisfiable set of clauses $\mathcal{C}_n$ for which the size of any proof graph with respect to $\mathcal{C}_n$ must exceed c^n, where $c > 1$ is a constant to be determined later.

The set of clauses $\mathcal{C}_n$ will model the following two statements involving the Pigeonhole Principle.

1. Each of $n + 1$ pigeons is placed in exactly one of n holes.
2. No two pigeons share the same hole.

These statements taken together are false, and hence $\mathcal{C}_n$ is unsatisfiable. Now let x_{ij} be the Boolean variable that evaluates to true iff hole i contains pigeon j, $1 \leq i \leq n$, $1 \leq j \leq n+1$. Then Statement 1. can be modeled with the **type-1** clauses

$$x_{1j} \vee \cdots \vee x_{nj},$$

$1 \leq j \leq n + 1$, while Statement 2 can be modeled with the **type-2** clauses

$$(\overline{x}_{i1}, \overline{x}_{i2}), \ldots, (\overline{x}_{i1}, \overline{x}_{i(n+1)}), (\overline{x}_{i2}, \overline{x}_{i3}), \ldots, (\overline{x}_{i2}, \overline{x}_{i(n+1)}), \ldots (\overline{x}_{in}, \overline{x}_{i(n+1)}),$$

$1 \leq i \leq n$.

Given the variables being used, it seems appropriate to use matrices to represent both assignments and clauses. For example, any assignment over the variables of $\mathcal{C}_n$ may be represented with an $n \times (n+1)$ matrix a for which a_{ij} is the value assigned to variable x_{ij}. Similarly, any clause over the variables of $\mathcal{C}_n$ can be represented with an $n \times (n + 1)$ matrix c for which

$$c_{ij} = \begin{cases} \oplus & \text{if } x_{ij} \in c \\ \ominus & \text{if } \overline{x}_{ij} \in c \\ \text{blank} & \text{otherwise} \end{cases}$$

Example 5. For $\mathcal{C}_3$, provide a matrix representation of i) the type-1 clause that asserts that Pigeon 1 is placed in some hole, and ii) the assignment that assisgns Pigeon 1 to holes 1 and 3, Pigeon 2 to hole 2, and no hole to Pigeon 3.

Assignment a is said to be **critical** iff a assigns each of n pigeons to n different holes, and assigns no hole to one of the pigeons. Thus, matrix a has n columns that have exactly one 1, and one **zero column** with no 1's.

Example 6. Give an example of a critical assignment for $\mathcal{C}_4$ in which column 3 serves as the zero column.

Moreover, Exercise 6 implies that, for the proof graph G_n of $\mathcal{C}_n$, and for any assignment a over $\text{var}(\mathcal{C}_n)$, there is a path in G_n from one of the clauses of $\mathcal{C}_n$ to the empty clause, for which each clause in the path is unsatisfied by a. In particular, if a be a critical assignment in which Pigeon j is not assigned to a hole, then the path P_a associated with a must begin with the type-1 clause

$$x_{1j} \vee \cdots \vee x_{nj}.$$

This clause (as a matrix) has n $\oplus$'s in column j. Furthermore, since P_a ends at the empty clause, which has zero $\oplus$'s in column j, and a single resolution step will produce a resolvent with at most one fewer $\oplus$ than one of its parents, it follows that there must be a clause in P_a that has exactly $n/2$ $\oplus$'s in column j.

Next, recall that, for any directed acyclic graph $G = (V, E)$, the vertices of G can be **topologically ordered** in such a way that, for every $(u, v) \in E$, u comes before v in the order. Assume such an ordering for G_n. Now, for critical assignment a whose zero column is j, let c_a be the first clause in the ordering that has exactly $n/2$ $\oplus$'s in column j, and that is not satisfied by a. By the previous paragraph, c_a must exist. Why does c_a have no $\ominus$ in column j?

Now let s be a partial assignment that assigns values to exactly $n/8$ of the $(n+1)$ columns, where each column is assigned exactly one 1 and $(n-1)$ 0's, and the $n/8$ 1's occur in different rows. In other words, s assigns exactly $n/8$ pigeons to $n/8$ distinct holes. Then define c^s as being the first clause c_a in the topological ordering for which critical assignment a is an extension of s. For each such s, c^s is called the **complex clause** associated with s. Notice that, for $c^s = c_a$, critical assignment a has exactly $n - n/2 - n/8 = 3n/8$ 1's that were not assigned by s and are not in the same row as any of the $n/2$ $\oplus$'s that occur in column j (the zero column of a) of c_a. Any column of c_a where one of the $3n/8$ 1's is located is called a **good column** of c_a.

Example 7. For $n = 8$, let s be the partial assignment that assigns column 1 the values (10000000) from top to bottom. Give an example of a critical-assignment extension a of s with zero column 9, along with a possible clause c^s for which $c^s = c_a$.

Claim: every good column of $c^s = c_a$ has either (exactly) one $\ominus$ or at least $n/2$ $\oplus$'s.

Proof of claim. Let s, a, and $c^s = c_a$ be given. First notice that no column of c_a can have two or more $\ominus$'s (why?). Now suppose that good column k of c_a has zero $\ominus$'s and fewer than $n/2$ $\oplus$'s. Let a^* be the critical assignment obtained from a by changing the 1 in column k to 0, and changing the 0 in column j (the zero column of a) to a 1, where the changed 0 in column j is in the same row as the changed 1 in column k. Notice that a^* does not satisfy c_a. This is true since i) a does not satisfy c_a, ii) a^* has all 0's in column k, but c_a has no $\ominus$'s in column k, and iii) the 0 that was changed to a 1 in column j is not in the same position as one of the $n/2$ $\oplus$'s of c_a, since the row where the change occurred was one of the $3n/8$ rows that neither coincide with a 1 of s nor with an $\oplus$ in column j.

Since a^* does not satisfy c_a, we can trace a path back from c_a to one of the original (type-1) clauses, such that every clause on the path is not satisfied by a^*. Moreover, the original type-1 clause of this path has n $\oplus$'s in column k, and, since c_a has fewer than $n/2$ $\oplus$'s in column k, there must be some clause c along the path that has exactly $n/2$ $\oplus$'s in column k. Hence, c has the same form as c_{a^*} which implies that $c_{a^*} \leq c < c_a$ in the topological ordering. But since a^* is also an extension of s, and c^s is defined as the *first* c_a for which a is an extension of s, c_{a^*} should have been chosen for c^s instead of c_a, which is a contradiction. Therefore, c_a must have either exactly one $\ominus$ or at least $n/2$ $\oplus$'s in each of its good columns. $\square$

Now let $c_1, \ldots, c_t$ denote the complex clauses that occur in G_n. Each c_r can be adjusted to form an **altered complex clause** c'_r. This is done by taking any column of c_r that has an $\ominus$ and replacing it with a column that removes the $\ominus$ and has $n-1$ $\oplus$'s, with no $\oplus$ in the former position of the $\ominus$. Now let s be a partial assignment that does not satisfy c_r. Suppose that one of the $n/8$ columns of s corresponds with a column of c_r that was changed to form c'_r. Since the unique 1 assigned by s in this column occurs at the former location of $\ominus$, it follows that s does not satisfy c'_r via this column, and hence s does not satisfy c'_r.

Let $A = \{c'_1, \ldots, c'_t\}$ denote the altered set of complex clauses. It follows by the above claim that each clause in A has at least $(3n/8 + 1) \cdot (n/2)$ $\oplus$'s. Let A_{ij} denote the set of altered complex clauses that have an $\oplus$ in row i and column j, $1 \leq i \leq n$, $1 \leq j \leq n + 1$.

We now describe a greedy algorithm for constructing a partial assignment s. The algorithm greedily attempts to construct s so that is satisfies as many of the t clauses of A as possible. The algorithm works in $n/8$ rounds, where in each round a position is selected for where the next 1 will be assigned by s. In round 1, the position is selected as (i, j), where $|A_{ij}|$ is a maximum. In other words, it assigns a 1 at the position where the most number of $\oplus$'s are occurring with respect to clauses in $A^1 = A$. It then sets $A^2 = A^1 - A_{ij}$.

Now, if an adversary, call him `min`, is attempting to spoil the ambitions of the algorithm, he would distribute the $t(3n/8 + 1) \cdot (n/2)$ $\oplus$'s uniformly about each of the $n(n + 1)$ positions. In this way assigning a 1 to a single position would be guaranteed to satisfy at least

$$\frac{t(3n/8 + 1)(n/2)}{n(n + 1)} \geq [\frac{t(3n/8)(n/2)}{n(n + 1)} = \frac{3t}{16}$$

clauses, or $\frac{3}{16}$ of the entire set.

Now suppose the algorithm has been described up to round $k - 1$ for some $k \geq 2$, and that $A^k \subseteq A$ has been defined. Let A^k_{ij} denote those clauses of A that have an $\oplus$ in row i and column j, $1 \leq i \leq n$, $1 \leq j \leq n + 1$, and have not been satisfied by s in any of the previous rounds. Then in round k the algorithm selects position (i, j) for which i) i is different from any row selected in a previous round, ii) j is different from any column selected in a previous round, and iii) $|A^k_{ij}|$ is a maximum among all such row-column combinations.

Similar to round 1, in round $k + 1$, if `min` wants to spoil the ambitions of the algorithm, he would distribute the $|A^{k+1}|(3n/8+1-k)\cdot(n/2-k)$ $\oplus$'s uniformly about each of the remaining $(n-k)(n+1-k)$ positions. In this way assigning a 1 to a single position would be guaranteed to satisfy a fraction of

$$\frac{(3n/8 + 1 - k)(n/2 - k)}{(n - k)(n + 1 - k))} \geq \frac{(3n/8 - k)(n/2 - k)}{(n - k)^2}.$$

Moreover, we can obtain a lower bound for this fraction by setting $k = n/8$ to get

$$\frac{(3n/8 - n/8)(n/2 - n/8)}{(n - n/8)^2} = \frac{6}{49}.$$

Thus, the number of clauses from A that remain unsatisfied after round k is no more than $(\frac{43}{49})^k|A|$. In other words,

$$|A^k| \leq t(\frac{43}{49})^k,$$

which, since $|A^k|$ is an integer, implies $|A^{n/8}| = 0$ provided

$$t(\frac{43}{49})^{n/8} < 1 \Rightarrow t < (\frac{49}{43})^{n/8} = c^n \approx (1.0646)^n.$$

Now suppose that t is in fact less than $(1.0646)^n$. Then the above algorithm implies that there is a partial assignment s that satisfies all complex clauses of G_n. But, by definition, $c^s = c_a$ is a complex clause of G_n, where a is an extension of s that does not satisfy c_a. Thus, s does not satifsy $c^s = c_a$ which is a contradiction. Therefore, it must be the case that $t \geq c^n$, where $c \approx 1.0646$. Therefore, the proof graph G_n has a size that grows exponentially in n, and the Resolution algorithm has worst-case exponential running time.

References

U. Schoning, R. Pruim, “Gems of Theoretical Computer Science”, Chapter 6
Springer, 1998

Exercises

1. We say that a clause c over a set of n variables is **simple** iff i) every literal appears at most once in c, and ii) if $l \in c$, then $\bar{l} \notin c$. Provide a counting argument to establish that there are exactly 3^n different simple clauses over n variables.

2. Use the method shown in Example 1 to determine all satisfying assignments for the saturated set of clauses
$$\{(x_2), (\overline{x}_1, x_3), (x_3, x_4), (\overline{x}_1, x_4)\}.$$

3. Perform the Resolution algorithm on the following set of satisfiable clauses
$$\{(\overline{x}_2, \overline{x}_3), (x_2, \overline{x}_4), (x_1, \overline{x}_3), (x_2, x_3), (x_1, x_4), (\overline{x}_1, x_4), (x_1, \overline{x}_2)\}.$$
Provide the final set of saturated clauses and use it to determine a satisfying assignment.

4. Repeat the previous exercise, but with the added clause of $(\overline{x}_2, x_3)$. Draw the refutation proof graph.

5. Let G be the proof graph for some set $\mathcal{C}$ of unstatisfiable clauses. Let a be any assignment over the clause variables. Prove that if a does not satisfy some clause c in $R(\mathcal{C})$, and if c has parents, then a also does not satisfy one of its parents.

6. Let G be the proof graph for some set $\mathcal{C}$ of unstatisfiable clauses. Let a be any assignment over the clause variables. Prove that there is a path in G from one of the clauses of $\mathcal{C}$ to the empty clause, for which each clause in the path is unsatisfied by a. Hint: use the previous exercise and work backwards from the empty clause.

7. How many critical assignments are there for Pigeon-Hole problem $\mathcal{C}_n$?

8. Given critical assignment a whose zero column is i, explain why c_a cannot have a $\ominus$ in column i.

9. Given partial asignment s and critical assignment a for which $c^s = c_a$, prove that no column of c_a can have two or more $\ominus$'s.