NP = P

Maurice Gittens <maurice@gittens.nl>

June 20, 2025

Abstract

A novel equivalence relation for boolean satisfiability formulae is leveraged to outline a straight forward polynomial time algorithm for the boolean satisfiability problem. This algorithm recursively partitions boolean satisfiability formulae into nine partitions without backtracking. The master theorem is used to prove that the operations involved are polynomial time. The existence of this algorithm serves as proof $NP = P$.

1 Representing formulae in CNF as Matrices

Let $F = c_0 \cdot c_1 ... \cdot c_{|k|}$ represent the set of boolean satisfiability formulae in conjunctive normal form with variable set $V = v_0, v_1, ...v_m$ and $C = c_0, c_1, ...c_n$. Every c_i is a disjunction of literals. All boolean satisfiability formulae F in CNF can be represented as matrix as shown below.

$$F = \boxed{\begin{array}{cccc} l_{0,0} & l_{0,1} & \cdots & l_{0,|C|-1} \\ l_{1,0} & l_{1,1} & \cdots & l_{1,|C|-1} \\ \vdots & \vdots & \vdots & \vdots \\ l_{|V|-1,0} & l_{|V|-1,1} & \cdots & l_{|V|-1,|C|-1} \end{array}}$$

where:

- $F[v_i, c_i] \in \{-1, 0, 1\}$ and $F[v_i, c_i] = -1$ when $\tilde{v_i}$ in clause c_i, $F[v_i, c_i] = 1$ when v in c_i . $F[v_i, c_i] = 0$ otherwise.
- $|V|$ represents the number of rows in F
- $|C|$ represents the number of columns in F

As an example the matrix matrix for $F = c_0 \cdot c_1, c_2, c_3 = (x+y) \cdot (x+\tilde{y}) \cdot (\tilde{x}+\tilde{z}) \cdot (\tilde{y}+z)$.,over variables x, y, z corresponding with rows $0, 1, 2$ is presented..

$$\text{F=}\boxed{\begin{array}{cccc} 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \\ 0 & 0 & -1 & 1 \end{array}}$$

2 Partitioning boolean satisfiability formulae

To solve the satisfiability problem in polynomial time the search space needs to be efficiently partitioned multi dimensionally. This section discusses an approach. Boolean satisfiability formula F in CNF represented as a matrix can be partitioned using a pair of variables called a partitioning key $k = (v_0, v_1)$ defined as a set of two variables with v_1, v_2 ,$v_1 \neq v_2$.

An example

Consider a specific example $F = (v_0 + v_1 + \tilde{v_4}) \cdot (v_1 + \tilde{v_2}) \cdot (v_0) \cdot (\tilde{v_1} + v_2 + \tilde{v_3} + \tilde{v_4}) \cdot (\tilde{v_0} + \tilde{v_2} + v_4) \cdot (v_3) \cdot (\tilde{v_1} + \tilde{v_4}) \cdot (\tilde{v_0} + v_1 + \tilde{v_2}) \cdot (\tilde{v_2} + \tilde{v_3} + v_4) \cdot (v_0 + v_3)$ with partitioning key $k = (0,4)$. The matrix representation of

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & -1 & 1 & 0 & -1 & 0 & 1 & 0 \end{bmatrix}.$$

Partitioning F by k yields the partitions f_0, f_1, f_2, f_3 defined as

$$f_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

corresponds with all columns x of F for which $F[0,x] = 0$ and $F[4,x] = 0$. Alternatively $f_0 = (v_1 + \tilde{v_2}) \cdot (v_3)$

$$f_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

corresponds with all columns x of F for which $F[0,x] \neq 0$ and $F[4,x] = 0$. All $F[0,x]$ are set to 0. Alternatively $f_1 = (v_1 + \tilde{v_2}) \cdot (v_3)$

$$f_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

corresponds with all columns x of F for which $F[0,x] = 0$ and $F[4,x] \neq 0$. All $F[4,x]$ are set to 0. Alternatively $f_2 = (\tilde{v_1} + v_2 + \tilde{v_3}) \cdot (\tilde{v_1}) \cdot (\tilde{v_2} + \tilde{v_3})$

$$f_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

corresponds with all columns x of F for which $F[0,x] \neq 0$ and $F[4,x] \neq 0$. All $F[0,x]$ and $F[4,x]$ are set to 0. Alternatively $f_3 = (v_1) \cdot (\tilde{v_2})$

Do take note that, given the partitioning key $k = (0,4)$ and the partitions f_0, f_1, f_2, f_3 it is not possible to reconstruct the formula F. To correct this deficiency, partitioning is extended.

Improving partitioning; continued example

Partitioning as presented is extended to allow the whole to be reconstructed from the parts. Towards this goal

f_3 is further partitioned further into 4 disjunct partitions $f_{30}, f_{31}, f_{32} f_{33}$ by leveraging the polarity of partition key variables. Clauses from f_3 are added to

- f_{30} when both literals of the partitioning key are negative
- f_{31} when the first partitioning key literal is positive and the second is negative
- f_{32} when first partitioning key literal is negative and the second is positive
- f_{33} when both literals from the partitioning key are positive

f_2 is further partitioned into 2 disjunct partitions f_{20}, f_{21} by leveraging the polarity of the second variable of the partitioning key. Clauses from f_2 are

- added to f_{20} when the polarity of the second variable of the partitioning key is negative
- added to f_{21} when the polarity of the second variable of the partitioning key is positive

f_1 is further partitioned into 2 disjunct partitions f_{10}, f_{11} by leveraging the polarity of the first variable of the partitioning key. Clauses from f_1 are

- added to f_{10} when the polarity of the first variable of the partitioning key is negative
- added to f_{11} when the polarity of the first variable of the partitioning key is positive

In the case of the running example, apply these rules yield:

$$f_{30} = f_{33} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, f_{31} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, f_{32} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&-1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

$$f_{20} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&-1&0&0&-1&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&-1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, f_{21} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&-1&0 \\ 0&0&0&0&0&0&0&0&-1&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

$$f_{10} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&-1&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, f_{11} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

Do take note that the example formula F can be reconstructed the partitioning key k, f_0 and the sub partitions $f_{10}, f_{11}, f_{20}, f_{21}, f_{30}, f_{31}, f_{32}, f_{33}$.

Two variable partitioning in general

In general, given a partitioning key $k = (x, y), x \neq y$ and formula $F = \boxed{\begin{matrix} l_{0,0} & l_{0,1} & \cdots & l_{0,|I|-1} \\ l_{1,0} & l_{1,1} & \cdots & l_{1,|I|-1} \\ \vdots & \vdots & \vdots & \vdots \\ l_{|v|-1,0} & l_{|v|-1,1} & \ldots & l_{|v|-1,|I|-1} \end{matrix}}$ The key k partitions F into formulae f_0, f_1, f_2, f_3 where

- f_3 represents the subset of columns of F having literals of both the row x and y. All entries of row $F[x]$ and row $F[y]$ are set to 0. Furthermore f_3 is sub partitioned into $f_{30}, f_{31}, f_{32} f_{33}$ such that clauses from f_3 are
  - added to f_{30} when both literals of the partitioning key are negative
  - added to f_{31} when the first partitioning key literal is positive and the 2nd is negative
  - added to f_{32} when first partitioning key literal is negative and the 2nd is positive
  - added to f_{33} when both literals from the partitioning key are positive
- f_2 represents the subset of columns of F that includes literals in row x and necessarily includes no literals in row y. All rows $F[x]$ are set to 0. Furthermore f_2 is sub partitioned into f_{20}, f_{21} such that clauses from f_2 are
  - added to f_{20} when the polarity of the second variable of the partitioning key is negative
  - added to f_{21} when the polarity of the second variable of the partitioning key is positive
- f_1 represents the subset of columns of F that with no literals of in row x and necessarily includes literals in row y. All $F[y]$ are set to 0. Furthermore f_1 is sub partitioned into f_{10}, f_{11} such that clauses from f_1 are
  - added to f_{10} when the polarity of the first variable of the partitioning key is negative
  - added to f_{11} when the polarity of the first variable of the partitioning key is positive
- f_0 represents the subset of columns of F that includes no literals in row x or y.

In summary key, a partitioning key $k = (x, y)$ partitions F into formulae $f_0, f_{10}, f_{11}, f_{20}, f_{21}, f_{30}, f_{31}, f_{32}, f_{33}$. The partitioning scheme introduced in this section is the singular differentiating enabler of polynomial time solution to the satisfiability problem. Straight forward implementation of this partitioning scheme are trivially polynomial time. .

3 An algorithm proving NP=P

3.1 Pseudo Code of algorithm for the Satisfiability problem

Consider the outline of a function *isSatisfiable* that takes as input a boolean satisfiability formula in CNF and returns true when the input formula is satisfiable and false otherwise..

```
boolean isSatisfiable(CNFFormula f, AssignmentSet ^outputAssignment)
{
    if (isSmallFormula(f))
        return isSatisfiableSmall(f, outputAssignment);
    partitioningKey = get_partitioning_key(f);
    AssignmentSet a0, a10, a11, a20, a21, a30, a31, a32, a33;
    CNFFormula f0, f10, f11, f20, f21, f30, f31, f32, f33;
    f0, f10, f11, f20, f21, f30, f31, f32, f33 = partition(f, partitioningKey);
    return
        isSatisfiable(f0 ,  a0) and
        isSatisfiable(f10, a10) and
        isSatisfiable(f11, a11) and
        isSatisfiable(f20, a20) and
        isSatisfiable(f21, a21) and
        isSatisfiable(f30, a30) and
        isSatisfiable(f31, a31) and
        isSatisfiable(f32, a32) and
        isSatisfiable(f33, a33) and
        mergeAssignmentSets(f, partitioningKey, a0, a10, a11, a20, a21,
        a30, a31, a32, a33, outputAssignment);
}
```

The function *isSatisfiable* recursively partitions the input formula into has nine parts without. It makes use of a few utility functions

isSmallFormula(CNFFormula f) returns true when

- the variable count of f is two or less.
- the clause count of f is two or less

isSatisfiableSmall*(CNFFormula f, AssignmentSet ^outputAssignments)* assumes that *isSmallFormula*(f) returns true. It returns

- false when f is not satisfiable. The variable *outputAssignments* is set to the empty set in this case
- true when f is satisfiable. In this case all assignments satisfying f are assigned to the output variable *outputAssignments*.

get_partitioning_key(CNFFormula f) given a CNF formula this function returns two available variables to be used for partitioning. This function is trivially polynomial time.

partition(CNFFormula f, PartitioningKey k) partitions f using k as presented in section 2. This function returns nine partitions of f as discussed in section 2.

mergeAssignmentSets(...) has 12 parameters. The objective of this function is merge assignment sets of nine partitions of f into a set of none conflicting assignments if possible. This function returns

- false when it is not possible to merge the assignments into a set of assignments without conflicts. The variable *outputAssignments* is set to the empty set in this case
- true when it is possible to merge the nine indivitual assignments sets into a single set of assignments that satisfies all nine sub formulae. In this case all assignments satisfying f are assigned to the output variable *outputAssignments*.

3.2 The time complexity

The function *isSatisfiable* function presented in the previous section is a classic divide and conquer algorithm to which the *master theorem* applies. The master theorem determines the computational complexity of recurrence relations whenever recurrence relations have the form: $T(n) = aT(\frac{n}{b}) + f(n)$ where

- n is the input size
- $a \geq 1$ is the number of sub problems
- $b > 1$ is the factor by which the problem size is divided
- $f(n)$ is the cost of partitioning and recombine the sub problems

For the *isSatisfiable* function both a and b are equal to 9 and $f(n) = n^k$. Applying the master theorem using these parameters yields a polynomial time complexity for the *isSatisfiable* function.

4 Conclusion

$NP = P$ is proven by reducing arbitrary boolean satisfiability formulae in the *CNF* to 2-SATs problems in polynomial time using an equivalence relation that efficiently partitions CNF formulae. The master theorem is used to prove that the operations involved are polynomial time.

References

[1] Wikipedia “Master theorem (analysis of algorithms)”.