

Project. Satisfiability test of clauses and its application

Rushikesh Khaml
Juily Kulkarni
Samuel Abiola

November 07, 2022

Problem Statement To demonstrate how to reduce problems to satisfiability and solve them using a SAT solver.

N-Queens Problem

- Given a chessboard of size $N \times N$, Find a way to arrange N queens so that no two queens may take each other.
- The queens can attack each other if they are placed in:
 - Same column
 - Same row
 - Same diagonal
- Algorithm:
 1. Begin with the first column on the left.
 2. For each row of the column:
 - (a) Position the queen so that it cannot attack the queens in previous columns.
 - (b) Add this cell to the solution set if such a placement is feasible, and then run the function on the next column to recursively verify whether this leads to a solution. Return one if it does.
 - (c) If not, take this cell out of the solution set.
 3. If after the end of step 2 there is no solution, go back to the previous column by returning zero.
 4. Once all of the queens have been positioned, halt the recursion.

Conjunctive Normal Form and Propositional Satisfiability

- In Conjunctive Normal Form (CNF), a compound proposition that is a conjunction of disjunctions is referred to. A CNF is, in other words, an AND of ORs.
- Example - $(p \vee \neg q) \wedge (\neg p \vee q) \wedge (p \vee q)(p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r)$
- Satisfiability can be checked using a truth table. If there is a truth assignment to a variable in a compound statement that makes it TRUE, the proposition is satisfiable. The composite proposition cannot be satisfied if such assignments are lacking.
- The conjunction of several phrases, each of which is a disjunction of a number of variables or their negations, constitutes a satisfiability problem in conjunctive normal form.
- Assess the satisfiability of a propositional sentence and, if it is, identify the propositions that must be true for the sentence to be true. A sentence in a specific format needs to have a truth assignment that is satisfying, which is the challenge of SAT.
- A trivially true empty conjunction of clauses $\vee \emptyset$ (satisfied by every assignment) and A disjunction of literals in an empty sentence, $\wedge \emptyset$ is trivially false (satisfied by no assignment).

Satisfiability Problem (SAT) and SAT Solver (MiniSAT)

- The Satisfiability Problem (SAT) is a problem solver that uses SAT formulas to handle binary variables related by logical relationships like OR and AND.
- Given a SAT formula (CNF), the Satisfiability Problem is used to determine whether it is satisfiable (or consistent). We occasionally would like to compute a suitable assignment if the CNF SAT formula is satisfiable or consistent (or model).
- A SAT solver is a program that takes a CNF formula as input and outputs either a satisfied Boolean assignment to the CNF formula's variables if it is consistent or UNSAT if it is not.
- Typically, these solvers are binaries that read input in the form of a text file containing the CNF formula and report the pertinent information to the console.
- An example of CNF is -
 $(x1 \mid \neg x5 \mid x4) \ \&$
 $(\neg x1 \mid x5 \mid x3 \mid x4) \ \&$
 $(\neg x3 \mid x4).$
- MiniSat is a straightforward, open-source SAT solver that was created for both researchers and developers and is distributed under the "MIT license."

- If an expression is written using only AND, OR, NOT, parentheses, and boolean variables, a SAT solver can decide if it is possible to discover assignments to boolean variables that would make the expression true. Most SAT solvers, including MiniSAT, can additionally display a set of assignments that make the expression true if it is satisfiable.

- **Input Format for MiniSAT**

- Like the majority of SAT solvers, MiniSAT accepts input in the streamlined "DIMACS CNF" format, which is a plain text format. The format is primarily line-oriented.
- The first line that starts with "c" is a comment. This is a SAT problem in CNF format, as shown by the "p cnf" line. The first line after the comment must be of the format - p cnf *Number of Variables* *Number of Clauses*
- Example of MiniSAT Input Format -
c SAT Expression for N=2
c Board has 4 positions
p cnf 4 10
1 2 0
-1 -2 0
3 4 0
-3 -4 0

- **Output Format for MiniSAT**

- When it is executed, miniSAT provides a variety of statistics about its operation to standard error.
- Depending on whether the expression can be satisfied or not, it will either print "SATISFIABLE" or "UNSATISFIABLE"
- MiniSAT will write text to a file if you provide it with an OUTPUT-FILE. It will say "SAT" or "UNSAT" on the first line. If SAT, a list of assignments to the boolean variables that satisfy the expression will be on the second line.
- Example of MiniSAT Output Format -
SAT
1 2 -3 4 5 0

CNF SAT version for the N-Queen problem

- In N-Queens Problem, we have below Variables and Constraints -

1. **Variables** - N Variables, B_{ij} for $0 \leq i, j < N$. If the queen is in position (i,j), $B_{ij} = T$; else, $B_{ij} = F$
2. **Constraints** -
 - (a) Exactly one queen per row - $Row(i) = B_{ij}$, $j = 0 \dots N - 1$, and $j \neq I$ which means queen not on same row

- (b) Exactly one queen per column - $Column(j) = B_{ij}, i = 0 \dots N - 1$, and $i \neq I$ which means queen not on same column
- (c) At most one queen on diagonal -
 - i. $Diagonal(k-) = B_{ij}, i - j = k = -N + 1 \dots, N - 1$
 - ii. $Diagonal(k+) = B_{ij}, i + j = k = 0 \dots, 2N - 2, k \neq I$ which means queen not on same diagonal

• **Rules for Exactly 1 queen per row**

1. They need at least one queen and no more than one queen in each row to avoid being attacked.
2. One queen in a row i , exactly
 - $B_{i0} \vee B_{i1} \vee B_{i2} \vee B_{i3}$ [Initially, Queen can be placed in any column for the selected row]
 - $B_{i0} \rightarrow \neg B_{i1} \wedge \neg B_{i2} \wedge \neg B_{i3}$ [If Queen is placed in column 0 for the row i , then next queen should not placed in other remaining column 1, 2, 3.]
 - $B_{i1} \rightarrow \neg B_{i2} \wedge \neg B_{i3}$ [If Queen is placed in column 1 for the row i , then next queen should not placed in other remaining column 2, 3.]
 - $B_{i2} \rightarrow \neg B_{i3}$ [If Queen is placed in column 2 for the row i , then next queen should not placed in other remaining column 3.]
 - Atleast one queen on every Row :

$$\bigwedge_{i=1}^8 \bigvee_{j=1}^8 b_{ij}$$

- Atmost one queen on every Row:

$$\bigwedge_{i=1}^8 \bigwedge_{0 < j < k \leq 8} (\neg b_{ij} \vee \neg b_{ik})$$

3. Atmost one queen on $Row(i)$
4. If we have a list of variables and only one of them can be True, then if we take two of them, neither of them can be True, and one of them must be false.

• **Rules for Exactly 1 queen per column**

1. They need at least one queen and no more than one queen in each column to avoid being attacked.
2. One queen in a column i , exactly
 - $B_{0j} \vee B_{1j} \vee B_{2j} \vee B_{3j}$ [Initially, Queen can be placed in any row for the selected column]

- $B_{0j} \rightarrow \neg B_{1j} \wedge \neg B_{2j} \wedge \neg B_{3j}$ [If Queen is placed in column 0 for the row i , then next queen should not placed in other remaining row 1, 2, 3.]
- $B_{1j} \rightarrow \neg B_{2j} \wedge \neg B_{3j}$ [If Queen is placed in column 1 for the row i , then next queen should not placed in other remaining row 2, 3.]
- $B_{2j} \rightarrow \neg B_{3j}$ [If Queen is placed in column 2 for the row i , then next queen should not placed in other remaining row 3.]
- Atleast one queen on every Column :

$$\bigwedge_{j=1}^8 \bigvee_{i=1}^8 b_{ij}$$

- Atmost one queen on every Column:

$$\bigwedge_{j=1}^8 \bigwedge_{0 < i < k \leq 8} (\neg b_{ij} \vee \neg b_{kj})$$

3. Atmost one queen on $Column(i)$
4. If we have a list of variables and only one of them can be True, then if we take two of them, neither of them can be True, and one of them must be false.

• **Rules for Exactly 1 queen per diagonal**

1. They need at least one queen and no more than one queen in each diagonal to avoid being attacked.
2. One queen in a diagonal $k-$, exactly
 - $B_{20} \rightarrow \neg B_{31}$ [Queen can be placed in any diagonal]
 - $B_{00} \rightarrow \neg B_{11} \wedge \neg B_{22} \wedge \neg B_{33}$ [If Queen is placed in column 0 for the row 0, then next queen should not placed in other remaining row and column 1, 2, 3.]
 - $B_{11} \rightarrow \neg B_{22} \wedge \neg B_{33}$ [If Queen is placed in column 1 for the row 1, then next queen should not placed in other remaining row and column 2, 3.]
 - $B_{22} \rightarrow \neg B_{33}$ [If Queen is placed in column 2 for the row 2, then next queen should not placed in other remaining row and column 3.]
3. One queen in a diagonal $k+$, exactly
 - $B_{01} \rightarrow \neg B_{10}$ [Queen can be placed in any diagonal]
 - $B_{30} \rightarrow \neg B_{21} \wedge \neg B_{12} \wedge \neg B_{03}$ [If Queen is placed in column 0 for the row 0, then next queen should not placed in other remaining row and column 1, 2, 3.]
 - $B_{21} \rightarrow \neg B_{12} \wedge \neg B_{03}$ [If Queen is placed in column 1 for the row 1, then next queen should not placed in other remaining row and column 2, 3.]
 - $B_{12} \rightarrow \neg B_{03}$ [If Queen is placed in column 2 for the row 2, then next queen should not placed in other remaining row and column 3.]

4. At most one queen on $Diagonal(i)$
5. If we have a list of variables and only one of them can be True, then if we take two of them, neither of them can be True, and one of them must be false.
6. Total formula for 8*8 queens problem :

$$\bigwedge_{i=1}^8 \bigvee_{j=1}^8 b_{ij} \wedge$$

$$\bigwedge_{i=1}^8 \bigwedge_{0 < j < k \leq 8} (\neg b_{ij} \vee \neg b_{ik}) \wedge$$

$$\bigwedge_{j=1}^8 \bigvee_{i=1}^8 b_{ij} \wedge$$

$$\bigwedge_{j=1}^8 \bigwedge_{0 < i < k \leq 8} (\neg b_{ij} \vee \neg b_{kj}) \wedge$$

$$\bigwedge_{0 < i < i' \leq 8} (\bigwedge \neg b_{ij} \vee \neg b_{i'j'})_{j:j':i+j=i'+j'-j=i'-j'}$$

Demonstrate the procedure to check the CNF is satisfiable or not

For validating CNF is satisfiable or unsatisfiable, for this assignment, we are using Minisat SAT solver tool. It will takes CNF files as input and generate output file with result of satisfiability or unsatisfiability. Following steps are followed for finding satisfiability of the CNF code.

- Minisat requires DIMACS CNF formatted file as input file. So, we need to generate CNF file while executing N-Queens problem. We need to generate CNF clauses and then need to convert it into DIMACS format.
- We have used python file to generate CNF format from N-Queen's problem. Let's discuss **Algorithm** for generate DIMACS CNF Format -
 1. Accept $N - Queensize$
 2. Check variable in given Array B is true for below 3 condition -

- (a) func *exactlyOne*(*B*) to check exactly one variable in Array *B* is true
temp = *atleastOne*(*B*) + *atmostOne*(*B*)
return temp
 - (b) func *atleastOne*(*B*) to check atleast one variable in Array *B* is true
for x in A: temp = temp + " " + str(x)
temp = temp + " 0"
return temp
 - (c) func *atmostOne*(*B*) to check atmost one variable in Array *B* is true
for x in A:
for y in A[A.index(x) + 1:]: temp = temp + " -" + str(x) + " -" + str(y) + " 0"
return temp
3. Check exactly 1 queen per row
for row in range(0, N):
A = []
for column in range(0, N):
position = varmap(row, column, N)
A.append(position)
temp = temp + exactlyOne(A)
 4. Check exactly 1 queen per column
for column in range(0, N):
A = []
for row in range(0, N):
position = varmap(row, column, N)
A.append(position)
temp = temp + exactlyOne(A)
 5. Check Atmost 1 queen per negative diagonal from top
for column in range(1, N):
A = []
for x in range(0, N-column):
A.append(varmap(x, column+x, N))
temp = temp + atmostOne(A)
 6. Check Atmost 1 queen per positive diagonal from right
for row in range(N-1, -1, -1):
A = []
for x in range(0, N-row):
A.append(varmap(row+x, N-1-x, N))
temp = temp + atmostOne(A)
 7. Check Atmost 1 queen per positive diagonal from top
for column in range(N-2, -1, -1):
A = []
for x in range(0, column+1):
A.append(varmap(x, column-x, N))
temp = temp + atmostOne(A)

8. Save temp content in CNF file format
with open('DIMACSCNFInput.cnf', 'w') as f:
f.write("c SAT Expression for N=" + str(N) + "
f.write("c Board has " + str(spots) + " positions" + "
f.write('p cnf ' + str(N*N) + ' ' + str(temp.count("")) + "
f.write(temp)
- After execution of the above algorithm, we will get DIMACS CNF formatted value which looks like -
c SAT Expression for N=4
c Board has 16 positions
p cnf 16 50
1 2 3 4 0
-1 -2 0
-1 -3 0
-1 -4 0
 - Execute the above generated CNF file in Minisat SAT Solver in terminal with following command - "minisat DIMACSCNFInput.cnf output2.txt".
It will generate output file with result of SAT (satisfiability) or UNSAT (unsatisfiability) as follows -
SAT
-1 -2 -3 4 5 -6 -7 -8 -9 -10 11 -12 -13 14 -15 -16 0

Output Screenshot

- Execute Python Code to generate CNF File


```

main.py DIMACS_CNF_Inputs...
1  #!/usr/bin/env python
2
3  #Construct an N-queens SAT encoding
4
5  import sys
6
7  def exactly_one(A):
8      temp = ""
9      temp = temp + atleast_one(A)
10     temp = temp + atmost_one(A)
11     return temp
12
13  def atleast_one(A):
14      temp = ""
15      for x in A:
16          temp = temp + " " + str(x)
17
18      temp = temp + " 0\n"
19      return temp
20
21  def atmost_one(A):
22      temp = ""
23      for x in A:
24          for y in A[A.index(x) + 1: ]:
25              temp = temp + " -" + str(x) + " -" + str(y) + " 0\n"
26      return temp
27
28  def varmap(r, c, N):
29      return r * N + c + 1
30
31  if len(sys.argv) > 1:
32      N = int(sys.argv[1])
33  else:
34      N = 4
35
36  if N<1:
37      print("Error N<1")
38      sys.exit(0)
39
40  spots = N*N
41
42  temp = ""
43  for row in range(0, N):
44      A = []
45      for column in range(0, N):
46          position = varmap(row, column, N)
47          A.append(position)
48      temp = temp + exactly_one(A)
49
50  for column in range(0, N):
51      A = []
52      for row in range(0, N):
53          position = varmap(row, column, N)
54          A.append(position)
55      temp = temp + exactly_one(A)
56
57  for row in range(N-1, -1, -1):
58      A = []
59      for x in range(0, N-row):
60          A.append(varmap(row+x, x, N))
61      temp = temp + atmost_one(A)
62
63  for column in range(1, N):
64      A = []
65      for x in range(0, N-column):
66          A.append(varmap(x, column+x, N))
67      temp = temp + atmost_one(A)
68
69  for row in range(N-1, -1, -1):
70      A = []
71      for x in range(0, N-row):
72          A.append(varmap(row+x, N-1-x, N))
73      temp = temp + atmost_one(A)
74
75  for column in range(N-2, -1, -1):
76      A = []
77      for x in range(0, column+1):
78          A.append(varmap(x, column-x, N))
79      temp = temp + atmost_one(A)
80
81  with open('DIMACS_CNF_Input.cnf', 'w') as f:
82      f.write("c SAT Expression for N=" + str(N)+'\n')
83      f.write("c Board has " + str(spots) + " positions" + '\n')
84      f.write('p cnf ' + str(N*N) + ' ' + str(temp.count('\n')) + '\n')
85      f.write(temp)
86
87
88

```

- DIMACS CNF file gets generated as output

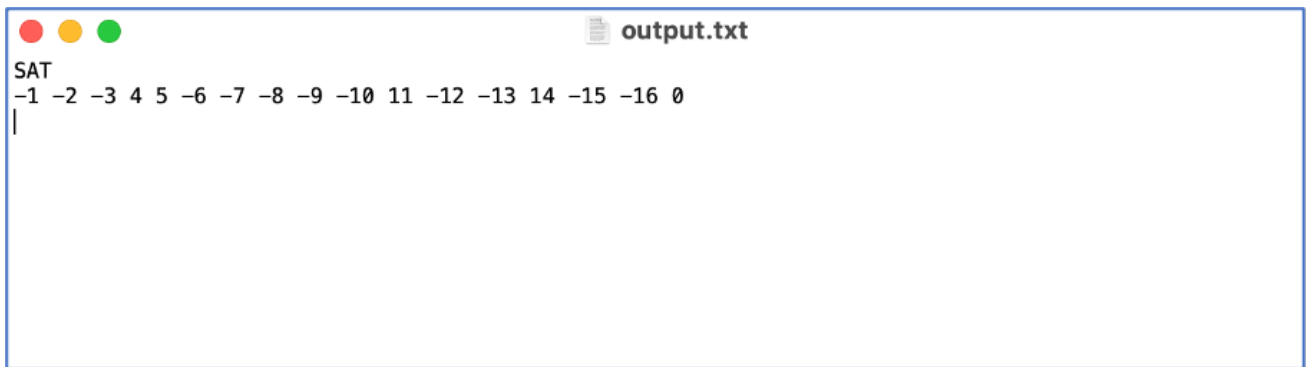
```
main.py DIMACS_CNf_inp... i
1 c SAT Expression for N=4
2 c Board has 16 positions
3 p cnf 16 50
4 1 2 3 4 0
5 -1 -2 0
6 -1 -3 0
7 -1 -4 0
8 5 6 7 8 0
9 -5 -6 0
10 -5 -7 0
11 -5 -8 0
12 9 10 11 12 0
13 -9 -10 0
14 -9 -11 0
15 -9 -12 0
16 13 14 15 16 0
17 -13 -14 0
18 -13 -15 0
19 -13 -16 0
20 1 5 9 13 0
21 -1 -5 0
22 -1 -9 0
23 -1 -13 0
24 2 6 10 14 0
25 -2 -6 0
26 -2 -10 0
27 -2 -14 0
28 3 7 11 15 0
29 -3 -7 0
30 -3 -11 0
31 -3 -15 0
32 4 8 12 16 0
33 -4 -8 0
34 -4 -12 0
35 -4 -16 0
36 0 -16 0
```

- Validate Satisfiability of the CNF Code using MiniSat Solver

```
rushikesh@Rushikeshs-MacBook-Air Final Project % minisat DIMACS_CNf_input.cnf output.txt
===== [ Problem Statistics ] =====
| Number of variables:      16 |
| Number of clauses:       50 |
| Parse time:              0.00 s |
| Eliminated clauses:      0.00 Mb |
| Simplification time:     0.00 s |
|
===== [ Search Statistics ] =====
| Conflicts | ORIGINAL | LEARNT | Progress | | | |
| | Vars | Clauses | Literals | Limit | Clauses | Lit/Cl |
|=====|=====|=====|=====|=====|=====|=====|
restarts      : 1
conflicts     : 1 (543 /sec)
decisions     : 5 (0.00 % random) (2717 /sec)
propagations  : 16 (8696 /sec)
conflict literals : 3 (0.00 % deleted)
Memory used   : 4.40 MB
CPU time      : 0.00184 s

SATISFIABLE
rushikesh@Rushikeshs-MacBook-Air Final Project %
```

- Output File generated by MiniSAT Solver



```
SAT
-1 -2 -3 4 5 -6 -7 -8 -9 -10 11 -12 -13 14 -15 -16 0
|
```

Thank You.