Transformation from 3SAT to Experimental Cuisine

Ridwanur Sarder
rrs148
rrs148@scarletmail.rutgers.edu

Nicholas Gutierrez
nag149
ag149@scarletmail.rutgers.edu

Udayveer Singh Andotra
ua118
ua118@scarletmail.rutgers.edu

Abstract—This project addresses the problem of transforming a 3SAT formula F with c clauses and v variables into a corresponding Experimental Cuisine framework, consisting of a maximum discord penalty threshold and a 2D array representing an ingredient discord matrix such that each ingredient represents set of possible variable assignments for a clause that will evaluate a clause to True. Our algorithmic transformation framework is an effective way of transitioning from a 3SAT problem to Experimental Cuisine. We manage to do this in just $O(c^2)$ time and space, where most of our work and space allocation goes into creating our discord matrix. For the act of just transforming our problem we can likely take up to a few thousand clauses with our space complexity in terms of input size. This is, however, mostly dependent on the machine running the programs.

Index Terms—algorithms, computational complexity,, constraint satisfaction, satisfiability, discord matrix, experimental cuisine

I. Input Format

A Comma Separated Values (CSV) file with rows for each clause and columns for each variable, encoding a 3SAT formula F. Each row represents a clause with up to three variables, where positive numbers indicate variables and negative numbers indicate negated variables. The input found at Table 1, for instance, corresponds to the below 3SAT problem instance:

$(\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_4)$

TABLE I
A SAMPLE INPUT CSV FILE

-1	2	4
-2	3	4
1	-3	4
1	2	-4

II. Output Format

The output solution and intermediate outputs are viewable as printed statements in our implementation. The Experimental Cuisine output is the selected ingredients with a total discord below the penalty threshold. The 3SAT final output is a valid variable assignment in the form of a printed dictionary sorted by the numbers representing the variables, increasing, if a valid variable assignment exists. These can be seen in Fig. 1, with verification looking like Fig. 3. If no such assignment exists (i.e. the 3SAT problem instance is unsatisfiable) then it will print that there is no such valid variable assignment, like in Fig. 2, with verification looking like Fig. 4. There is also an extra verification step of the output to double check the result.

```
Input 3SAT is satisfiable
Selected Ingredients:
{'clause_ind': 0, 'assignment': [-1, 2, 4]}
{'clause_ind': 1, 'assignment': [2, 3, 4]}
{'clause_ind': 2, 'assignment': [-1, 3, 4]}
{'clause_ind': 3, 'assignment': [-1, 2, 4]}
Solution:
x1: False
x2: True
x3: True
x4: True
```

Fig. 1. A satisfiable output screenshot

```
Input 3SAT is not satisfiable
```

Fig. 2. An unsatisfiable output screenshot

```
3SAT problem: [[-1, 2, 4], [-2, 3, 4], [1, -3, 4], [1, 2, -4]]
Selected variable values:
1: False
2: True
3: True
4: True
Variable Values satisfy the 3SAT problem (All clauses given variable assignments evaluated to true)

Selected working selection: (0, 11, 20, 26)
Discord: 0
Penalty Threshold: 0
Cuisine satisfies the penalty threshold
```

Fig. 3. A satisfiable output verification screenshot

```
3SAT problem: [[-1, -2, -3], [-1, -2, 3], [-1, 2, -3], [-1, 2, 3], [1, -2, -3], [1, -2, 3], [1, 2, -3], [1, 2, 3]]
3SAT is unsatisfiable.

Cuisine is unsatisfiable.
```

Fig. 4. An unsatisfiable output verification screenshot

III. Transformation

A. Construction of Experimental Cuisine Framework

The Experimental Cuisine framework consists of a symmetric and square discord matrix and a penalty threshold. The discord matrix represents the incompatibility between any two ingredients, where $cell_{i,j}$ represents the discord penalty incurred by ingredients i and j. Given these two inputs, the goal is to find the maximum number of ingredients such that the total discord penalty incurred by all combinations of the ingredients is less than or equal to the penalty threshold.

We define our ingredients to be the clauses that make up the 3SAT problem. More specifically, we take every single possible variable assignment for a given clause that will evaluate the individual clause to True, and list those as ingredients (each ingredient is stored as a key-value pair of clause index: variable assignment list). This comes out to 7 ingredients per clause, since out of the 8 possible outcomes the only False evaluation possible is when all three literals

come out to be False in the clause. For every ingredient, we check its compatibility with every other ingredient - if the resulting variable assignments of one clause contradict that of another clause, we set the discord between the two ingredients to 1. We also check the clause self compatibility - we can't have a variable and its negation both present in the same clause. Once this is complete, we have successfully transformed our 3SAT problem into a problem of Experimental Cuisine.

The time complexity of this is $O(n^2) = O(c^2)$, where n is the number of ingredients and c is the number of clauses. We say this particularly because the dominating factor timewise is the discord matrix formation, which has nonscaling time operations on a 7c x 7c matrix. This comes out to $49c^2$ time complexity, which simplifies to $O(c^2)$. The space complexity of this is also $O(n^2) = O(c^2)$ since usage of space is dominated by the discord matrix.

B. Reconstruction of a 3SAT assignment

Given a 3SAT problem turned Experimental Cuisine framework, there are only a select number of ingredient combinations that will yield 0 cumulative discord, and there will always be as many ingredients as there are clauses, with each ingredient corresponding to a specific clause. We can transform this into a 3SAT assignment by parsing all assignments from every ingredient, maintaining a set of the variables we have seen and whether they were selected to be True or False. In the end, we will have the variable assignments of all variables present in the 3SAT problem that will satisfy the problem.

The time complexity of this is O(c) where c is the number of clauses, because we perform nonscaling time actions for each clause (we iterate through each variable of the clause but there are always 3 variables in a clause). The space complexity is also O(c) since our list of ingredients (size c) is dominated by 3 variables and one clause index.

C. Solving Experimental Cuisine

We did a brute force solving of Experimental Cuisine where we get all possible combinations of ingredients such that there is 1 ingredient from each clause. This comes out to 7^c possible combinations, where c is the number of clauses in the 3SAT problem.

The time complexity of the brute force approach comes out to $O(7^c c^2)$, since we spend c^2 time on each combination of ingredients to check for satisfiability. The space complexity of this is $O(7^c c^2)$ since usage of space is dominated by the collection of possibilities being iterated.

IV. Sample Input and Output

We used several examples to test our code, to make sure our algorithm succeeds in the case of there being a 3SAT input with multiple answers (Table 2), one possible answer, (Table 3), or no answers (unsatisfiable, Table 4).

TABLE 2
MANY ANSWER INPUT (T, T, T)

1	2	3
1	-2	-3
-1	2	-3
-1	-2	3

TABLE 3
ONE ANSWER INPUT (T, T, T)

1	2	3
1	2	-3
1	-2	3
1	-2	-3
-1	2	3
-1	2	-3
-1	-2	3

TABLE 4
UNSATISFIABLE INPUT

-1	-2	-3
-1	-2	3
-1	2	-3
-1	2	3
1	-2	-3
1	-2	3
1	2	-3
1	2	3

V. Programming Language and Libraries Used

List all software tools used.

- Jupyter Notebook/Google Colab
- Python 3.10
- Pandas
- Numpy
- Itertools
- Python-Sat
- ChatGpt[1]

VI. Conclusions

We were able to successfully transform our 3SAT problem instances into Experimental Cuisine in polynomial time, showing that Experimental Cuisine is indeed NP-Complete. It is NP-Hard because we could transform into it from 3SAT in polynomial time, and it is NP because we could check our answers in polynomial time.

References

[1] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. 2006. Algorithms (1st. ed.). McGraw-Hill, Inc., USA.

[1] Used for minor syntax and debugging purposes only