Transformation from 2SAT to Strongly Connected Components

Group Leader Name: Abhi Patel
Net ID: ap2566
Email address: ap2566@scarletmail.rutgers.edu

Group Member Name: Ziyu Lin
Net ID: zl711
Email address: zl711@scarletmail.rutgers.edu

Group Member Name: Mihir Bapat
Net ID: mb2444
Email address: mb2444@scarletmail.rutgers.edu

Abstract— This project provides a 2-SAT to SCCs path problem transformation, which makes large-scale satisfiability resolution more efficient. The transformation supports sizes of inputs up to 2^20, connecting complicated logical constraints with the graph traversal paradigm, with a linear time complexity of O (n + m), where n is the total number of variables while m is the number of clauses.

Index Terms— Algorithms, 2-SAT, Graph Theory, Strongly Connected Components, Computational Complexity

I. Input Format

Input Format:

The input file, input1.py, plays a pivotal role in enabling the generation of random clauses tailored for solving the 2SAT problem. By offering parameters such as num_clauses and num_vars, it provides users with the flexibility to customize the scale and complexity of the generated clauses according to their specific requirements. In this configuration, the script is configured to produce a substantial number of clauses, with 1024 clauses and 256 variables, ensuring a diverse and comprehensive dataset for analysis. Each clause is meticulously crafted by randomly selecting two integers, var_i and var_j, within the range of 1 to num_vars. This meticulous process ensures the creation of clauses that encapsulate a wide array of logical constraints, crucial for robust evaluation and testing. Once generated, these clauses are efficiently organized and stored in a CSV file named "2sat_problem_adjusted.csv", easing seamless access and use. Additionally, to provide a comprehensive record of the generated clauses and enable further analysis, the script also produces a JSON file, "clause_assignments.json", having a detailed listing of all clauses generated.

II. Output Format

The output is in the form of SCCs graph which has components that are satisfiable and non-satisfiable. The satisfiable components are shown in the form of graph that is connected and the unsatisfiable components are on their own.

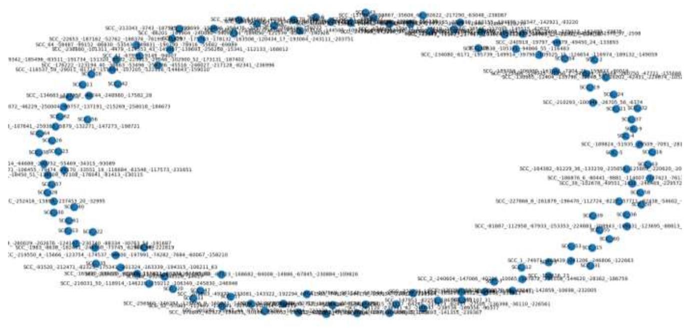


Figure 1: Matplotlib graph

Several outputs are produced by the code, such as the satisfaction status (satisfiable or not satisfactorily). Comprehensive details about SCCs, encompassing every element and the most robust SCC. Parts of the graph with weak connections. SCCs visualized with Matplotlib and NetworkX. exporting data for more analysis as CSV and JSON files. In summary, the transformation entails constructing an ordered graph illustration of the 2SAT problem, which is then enhanced to include SCCs. This allows for the effective resolution of satisfiability and offers insights into the problem's logical structure. The main output of this script is the answer to the question of whether the 2-Satisfiability (2SAT) problem can be solved. It also offers different options for analysis and visualization in addition to insights into the Strongly Connected Components (SCCs) within the transformed graph.

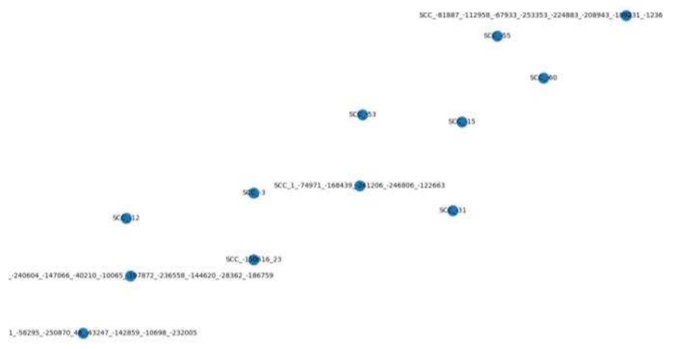


Figure 2: Matplotlib graph (magnified version of nodes of matplotlib graph)

III. Transformation

A. Construction of G_F

The SCC transformation augments the graph by adding edges within each SCC, ensuring that variables within the same SCC share the same value assignment. This guarantees logical consistency, as assignments propagate uniformly within SCCs. This transformation simplifies the resolution process by enabling the identification of satisfiability based on consistent variable assignments within SCCs.

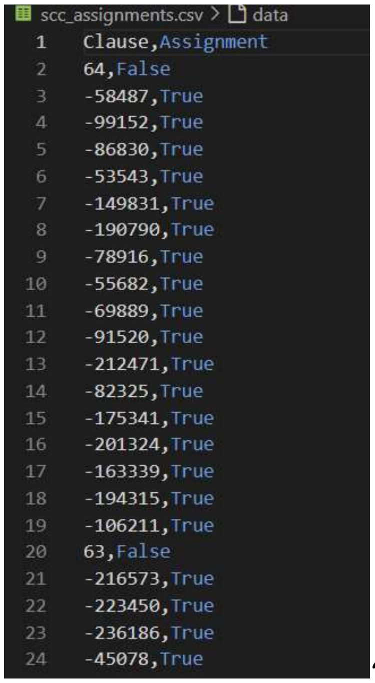
B. Reconstruction of a SCC graph

The SCC transformation augments the graph by adding edges within each SCC, ensuring that variables within the same SCC share the same value assignment. This guarantees logical consistency, as assignments propagate uniformly within SCCs. This transformation simplifies the resolution process by enabling the identification of satisfiability based on consistent variable assignments within SCCs.

C. Solving INDEPENDENT SET

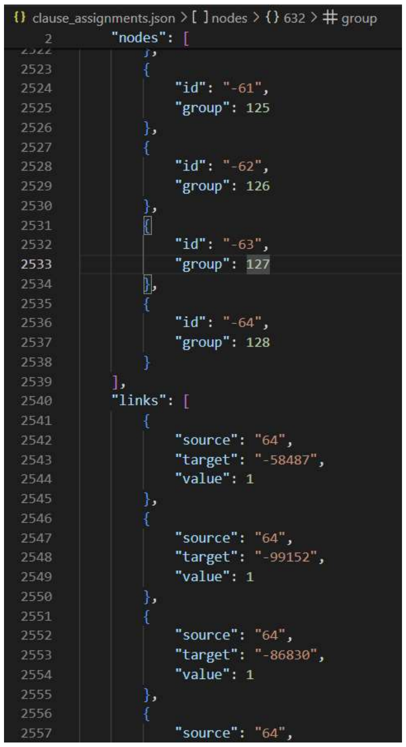
The code employs SCC detection within the transformed graph to solve the 2SAT problem. If any SCC includes both a variable and its negation, showing an inconsistency, the problem is marked as unsatisfiable. In contrast, if no such SCC is found, the problem is considered satisfiable. In this case, variable assignments are derived from the SCCs, ensuring that variables within the same part share consistent assignments, thereby enabling a solution to the problem.

IV. Sample Input and Output



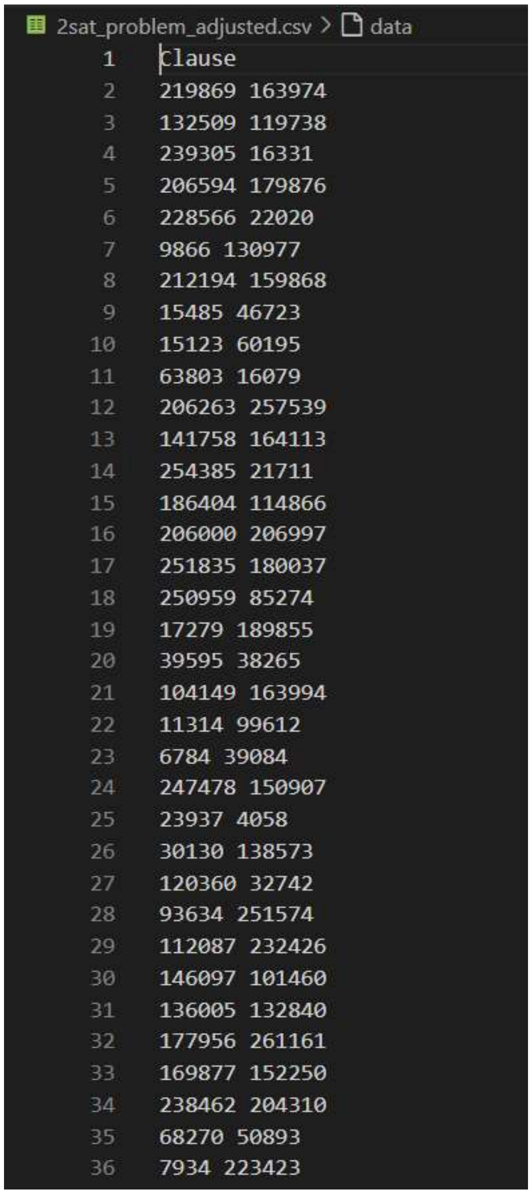
```
Clause,Assignment
64,False
-58487,True
-99152,True
-86830,True
-53543,True
-149831,True
-190790,True
-78916,True
-55682,True
-69889,True
-91520,True
-212471,True
-82325,True
-175341,True
-201324,True
-163339,True
-194315,True
-106211,True
63,False
-216573,True
-223450,True
-236186,True
-45078,True
```

Figure 3: Clauses with assignments of true if it is part of SCC and false if is it not a part of SCC. in csv file



```
    "nodes": [
            },
            {
                "id": "-61",
                "group": 125
            },
            {
                "id": "-62",
                "group": 126
            },
            {
                "id": "-63",
                "group": 127
            },
            {
                "id": "-64",
                "group": 128
            }
        ],
        "links": [
            {
                "source": "64",
                "target": "-58487",
                "value": 1
            },
            {
                "source": "64",
                "target": "-99152",
                "value": 1
            },
            {
                "source": "64",
                "target": "-86830",
                "value": 1
            },
            {
                "source": "64",
```

Figure 4: JSON with links for Graphs



```
Clause
219869 163974
132509 119738
239305 16331
206594 179876
228566 22020
9866 130977
212194 159868
15485 46723
15123 60195
63803 16079
206263 257539
141758 164113
254385 21711
186404 114866
206000 206997
251835 180037
250959 85274
17279 189855
39595 38265
104149 163994
11314 99612
6784 39084
247478 150907
23937 4058
30130 138573
120360 32742
93634 251574
112087 232426
146097 101460
136005 132840
177956 261161
169877 152250
238462 204310
68270 50893
7934 223423
```

Figure 5: 2SAT problem clauses in CSV

V. Programming Language and Libraries Used

- Python 3.12.3
- numpy: For numerical computing and array manipulation.
- networkx: For working with graphs and visualizing Strongly Connected Components.
- matplotlib: For creating visualizations, including plotting graphs.
- csv: For reading and writing CSV files.
- json: For working with JSON files.
- random: For generating random numbers.
- threading: For implementing concurrent monitoring.
- time: For timing and sleeping functions.
- psutil: For system monitoring.

VI. Conclusions

In this project, we have devised an effective method to solve the 2-Satisfiability (2SAT) problem. The 2SAT problem has been successfully converted into a Strongly Connected Components (SCCs) path problem. By transforming logical constraints into a graph traversal problem, we can effectively solve large-scale 2SAT instances by utilizing graph theory. Through the identification of SCCs in the transformed graph, we have developed a useful technique to ascertain whether the 2SAT problem is satisfiable. By using this method, we can effectively identify contradictions, inconsistencies, and gain understanding of the problem's logical structure.

We have put in place tools to use Matplotlib and NetworkX to the SCCs. In order to facilitate additional analysis and interpretation of the findings, this visualization helps to understand the connectivity and connections within the graph. The script exports the results to CSV and JSON files, making the solution easily accessible and making it easier to integrate it with other platforms and tools or conduct additional analysis. Deeper insights into the issue and its resolution might be obtained by improving the visualization and analysis capabilities, for example, by putting more sophisticated graph algorithms or interactive visualization tools into use.

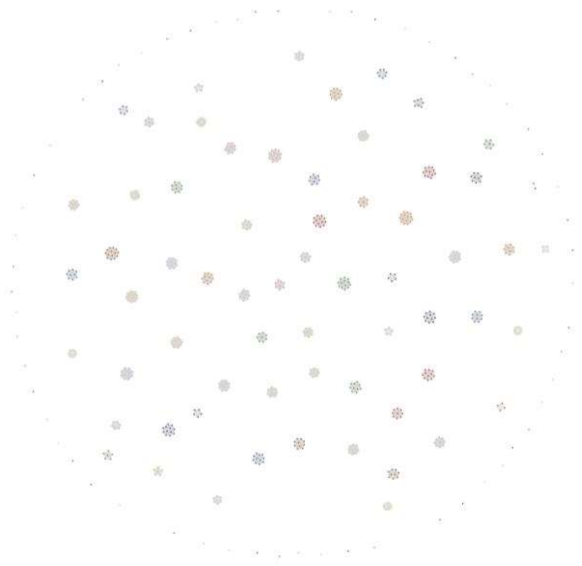


Figure 6: Collection of all Metanodes

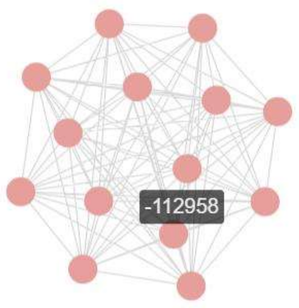


Figure 7: Metanodes Graph of a single metanodes

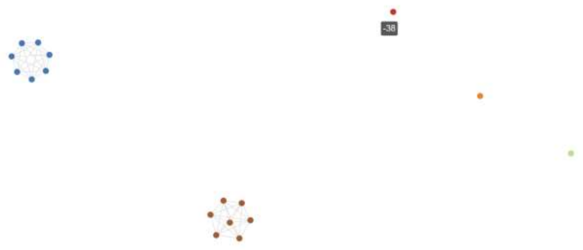


Figure 8: Final transformation from 2SAT to SCC

REFERENCES

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms, Janhunen, T. (2004, August). Representing normal programs with clauses. In ECAI (Vol. 16, p. 358).

[2] Asturiano, V. (2024, April 15). vasturiano/force-graph. GitHub. https://github.com/vasturiano/force-graph

[3] Sanjoy Dasgupta, Papadimitriou, C. H., & Umesh Virkumar Vazirani. (2008). Algorithms. Mcgrawhill Education, Cop.

[4] De Beaudrap, N., & Gharibian, S. (2015). A linear time algorithm for quantum 2-SAT. arXiv pr