Transformation from 3SAT to Matching in Bipartite Graphs

Group Leader Name:
Vraj Ketan Kumar Shah
Net Id: vs921
Email address:
vs921@scarletmail.rutgers.edu

Group Member Name:
Abhishek Jani
Net Id: aj1121
Email address:
aj1121@scarletmail.rutgers.edu

Group Member Name:
Christo Mathew
Net Id: cm1788
Email address:
christo.mathew@rutgers.edu

Abstract—The project converts a 3SAT problem instance into a matching bipartite graph, this helps us to gain insight into satisfiability of the original 3SAT instance. The input is randomly generated where variables appear at most 3 times. The input is transformed into a bipartite graph where the two partitions of nodes represent the variables and clauses respectively, with the edges indicating variable occurrence in the clauses, with a time complexity of O(n + m), where n is the number of variables and m is the number of clauses. Our implementation can support input sizes of upto 2^{16}.

Index Terms— 3SAT, bipartite graphs, maximum matching, computational complexity, algorithms

I. Input Format

A Comma Separated Values (CSV) file with rows representing each clause and columns representing each variable, encoding a 3SAT formula F. The variables can be a positive or negative number. The file contains a total of m rows, and each row up to 3 variables. The columns of the CSV are labeled as Literal_1, Literal_2, and Literal_3.

For example: the first row in the CSV file (Table I) encodes the clause $\neg x_3 \vee x_4 \vee x_5$. The four rows in the table encode a Boolean formula F in conjunctive normal form with three clauses.

TABLE I
A SAMPLE INPUT CSV FILE

Literal_1	Literal_2	Literal_3
-1	2	3
1	-2	-4
-3	4	5
-1	-4	-5

II. Output Format

Satisfiability analysis and Matching: Checks whether there exists a satisfiable assignment for the 3SAT formula. If there exists a satisfiable assignment, the matching is printed as a dictionary in the format {*vi*: *cj*}, where *vi* is the variable and *cj* is the clause to which it is matched to. If no satisfying assignment exists, the reason for failure is printed instead.

A bipartite graph B with 2 partitions: variable clauses and clause nodes. The left and right partition are formed by the variable nodes and clause which are labelled as $v_1, v_2,\ldots v_n$ and $c_1, c_2\ldots c_m$. An edge is established between a variable node and a clause node if the variable (or its negation) appear in that clause. An edge has color red if it represents a matching, and gray otherwise.

III. Transformation

The transformation of 3SAT to a bipartite graph includes:

A. Construction of G

1. Nodes Creation:

For each variable v_i and clause c_j in the 3SAT formula, a corresponding variable node labeled vi is created in the left partition and a clause node named cj is created in the the right partition of the bipartite graph respectively.

2. Edge Creation:

For each clause c_j 3SAT formula, three variables in the clause are identified. An edge is created between the nodes corresponding to each of these variable vi and the clause cj.

Time Complexity:

Variable node creation and clause nodes creation has time complexity of O(n) and O(m) respectively, where n is the number of nodes and m is the number of clauses. The overall complexity of bipartite graph construction is O(n+m).

B. Reconstruction of a 3SAT assignment

The 3SAT formula is reconstructed from a bipartite graph by translating relationships between variable nodes vi and clause nodes cj into 3SAT clauses. Clause nodes cj are identified from the right partition of the graph, and their neighboring variable nodes vi are used to reconstruct clauses. The edges between vi and cj contain a literal attribute, specifying whether the variable is positive or negated in the clause. These literals are extracted and grouped into lists representing the clauses. The reconstructed clauses are then organized into a pandas dataframe with columns for the three literals in each clause, providing a readable representation of the 3SAT formula.

IV. Sample Input and Output

Input CSV File:

Literal_1	Literal_2	Literal_3
-1	2	3
1	-2	-4
-3	4	5
-1	-4	-5

Output:

1.Satisfiability analysis and matching:

i)When the assignment can be satisfied:

The 3SAT formula is satisfiable.
Reason: A maximum matching was found, covering all 16384 clauses.
Matching: {'v15432': 'c942', 'v1785': 'c14698', 'v5050': 'c7136', ...}

ii) When the assignment cannot be satisfied:

Finding maximum matching in the bipartite graph...
An error occurred while finding the matching.
Error Details: Disconnected graph: Ambiguous solution for bipartite sets.

2. Bipartite graph:

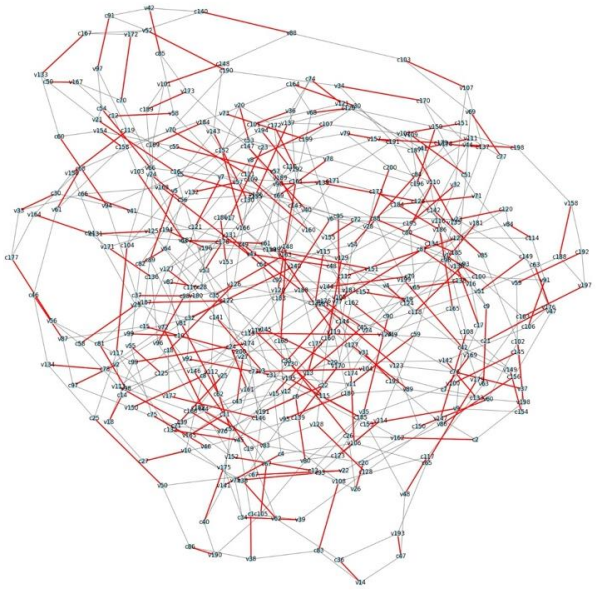

Figure 1: Graph with 200 input variables and clauses

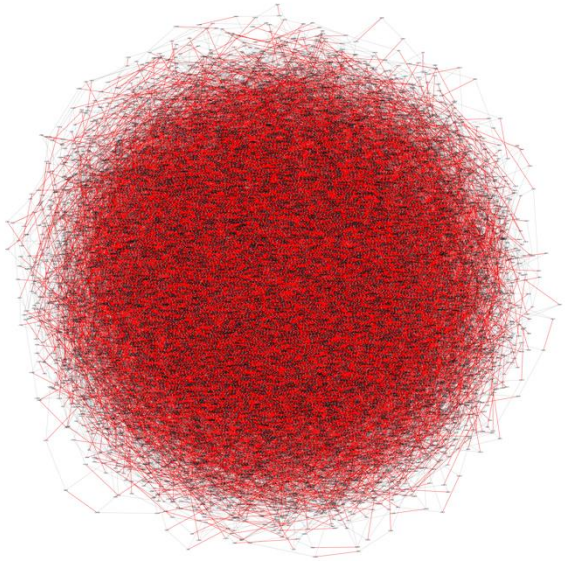

Figure 2: Graph with 2^14 input variables and clauses

V. Programming Language and Libraries Used

List all software tools used.

- Python 3.12
- Pandas
- Networkx
- Numpy 1.26 [3]
- Matplotlib

VI. Conclusions

In this project we have successfully transformed a 3SAT problem into a bipartite graph representation, effectively capturing variable-clause relationships and enabling the application of maximum matching algorithms. With this implementation we have demonstrated scalability, efficiency in handling large inputs while providing visualizations to enhance understanding of the problem.

We have checked the satisfiability of the given 3SAT formula and generated the corresponding matching if it exists. We have used tools like Matplotlib and NetworkX to visualize the bipartite graph. Deeper insights into the problem and its resolution could be achieved by enhancing visualization and analysis capabilities, such as utilizing more advanced graph algorithms or incorporating interactive visualization tools.

References

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

[2] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science and Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[3] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-0202649-2.

[4] M. R. Garey and D. S. Johnson, "*Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W. H. Freeman, 1979.
[5] **S. Arora and B. Barak**, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
[6] **C. H. Papadimitriou**, *Computational Complexity*. Addison-Wesley, 1994.
[7] **Python Documentation**: https://docs.python.org/3/
[8] **NetworkX Documentation** : https://networkx.org/documentation/stable/