Transformation from 3D Matching to Zero-One Equation (ZOE)

Raghvesh Prasad
rp1399
rp1399@scarletmail.rutgers.edu

Anirban Chakraborty
ac1567
ac1567@scarletmail.rutgers.edu

***Abstract*—This project explores the transformation of the 3D Matching problem into the Zero-One Equation (ZOE) format. We detail the algorithm used to represent 3D Matching as a ZOE, including its time complexity and the size of inputs it can handle. Our conclusions demonstrate the application of this transformation using computational scripts via various brute-force methodologies.**

I. Input Format

Input for this transformation are two CSV files input.csv and size.csv.
Input.csv contains rows representing a triple with distinct variables from each set. In our case, we utilized boys, girls, and pets, indicating a potential match. Each row contains exactly three elements: a boy's name, a girl's name, and a pet's name. Example:

```
Boy_1,Girl_1,Pet_1
Boy_2,Girl_2,Pet_2
```

...
Size.csv contains a single numeric value indicating the size of all the individual sets used in the project. Example:

```
16
```

II. Output Format

The output is a binary vector x that satisfies the equation $Ax = 1$, where A is a matrix constructed from the input triples. If a valid matching exists, x will indicate which triples form the valid matching. Otherwise, the output indicates no solution exists. The output is printed via terminal, with the set of distinct triples given from the input and the 'A' matrix. Thus, the output simply verifies if the 3D Matching input can be satisfied.

III. Transformation

The transformation involves creating a matrix A where each column corresponds to a triple and each row represents an individual (boy, girl, or pet). A '1' in matrix A indicates the inclusion of the individual in the corresponding triple. Solving $Ax = 1$ finds a valid 3D Matching where each individual is included exactly once.

A. Construction of Matrix A

For each triple (boy, girl, pet) in the input CSV, a column is added to matrix A with '1's in rows corresponding to b, g, and p. The complexity of this construction is $O(n \times m)$, where n is the number of triples and m is the total number of individuals.

B. Solving for Vector x

The vector x is solved using a brute force approach, evaluating all possible combinations of triples to see which combination satisfies $Ax = 1$. By calculating every binary representation of 'x', we utilize parallel processing to find which representation satisfies the ZOE equation $Ax = 1$.

IV. Programming Language and Libraries Used

- Python 3
- NumPy for matrix operations
- concurrent.futures for multithreading
- os for CSV implementation

V. Solving for x using Relaxation

While ZOE is bound to the constraint $x \in (0, 1)$.
We added a relaxation to this constraint by taking $x \in (\mathbb{R})$.
This allowed us to compute Pseudo-Inverse of A which yields the solution to x by computing $x = A^{-1} \cdot b$
The result of this dot product gives us a column vector (x) which has values very close to 1 and values comletely away from 1.
This can be transposed back to the solution set by including the Triples at index where values were closer to 1 in the solution set M. Using this method we could achieve a time complexity of $\simeq O(n^3)$.
However, this method uses a major relaxation on ZOE and thus has a chance to diverge from giving correct results.

VI. Conclusions

This project demonstrates the transformation of 3D Matching to ZOE and evaluates its feasibility on how ZOE can transform the problem into a integer programming problem for computers to solve. While our implementation is still bounded by the worst case time exponential complexity, using parallel processing on the latest processing power only added more time to compute the vector x. As the main thread waits for all children threads to exit before continuing its process. This results in extra wait time even though one of the threads could have found a potential solution to x. We thought of killing the threads that have not yet found a solution abruptly, and keep the thread which returned with the solution to x. This approach destabilizes the processing of the main thread and also creates problems with memory leaks.

The primary limitation on our solution relies on memory and processing power for computation, not algorithmic flaws in worst case scenarios. A future addition to our solution can involve automated pruning of potential solutions that would be incorrect to reduce space complexity.

We explored alternatives, such as the Pseudo-Inverse and Simplex Method. However, they did not ensure a decisive decision on whether the input generates a solution due to numerical precision instability. They have a chance to yield false positives on inputs that clearly provided no solution from the brute force.

VII. References

1) Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. 2006. Algorithms (1st. ed.). McGraw-Hill, Inc., USA.
2) https://docs.python.org/3/library/concurrent.futures.html
3) https://www.johndcook.com/blog/2018/05/05/svd/
4) https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html