Transformation from Rudrata Cycle to the Reliable Network

| Abhishu Oza  ao731  abhishu.oza@rutgers.edu | Vinit Mepani  vm759  vinit.mepani@rutgers.edu | Atharv Karbhari  amk546  amk546@scarletmail.rutgers.edu |
| --- | --- | --- |

***Abstract*—** In this project, we are transforming the Rudrata Cycle problem into the Reliable Network Problem. Given a Rudrata Cycle instance represented by a graph, we need to determine a cycle that visits each vertex exactly once. The output of the transformation are matrices **d**, the reliability matrix **r** and the **budget**, which is an instance of the Reliable Network Problem. In the Reliable Network problem, we need to construct a graph G(V, E) where the total cost of the edges does not exceed a given budget, and the graph satisfies the reliability constraints, ensuring that there are at least rij vertex-disjoint paths between every pair (i, j) of nodes. The transformation is computationally feasible, with a time complexity of O(n2) for transforming the inputs to the Reliable Network.

***Index Terms—algorithms, Rudrata Cycle, Reliable Network, computational complexity.***

**I. INPUT FORMAT**

## The input is provided in the form of a file containing the adjacency matrix **A** for a fully connected graph. The file represents the distances between each pair of vertices, where each cell (i, j) corresponds to the distance between vertex i and vertex j. For a fully connected graph, all vertices are connected by edges, and all entries in the matrix are finite numbers, except for the diagonal entries Dii , which are typically zero (indicating no distance to itself). The distance matrix will be a square matrix of size n × n where n is the number of vertices in the graph. The file will have n rows and n columns, with each row representing the distances from a single vertex to all other vertices.

*Sample Input:*

| 0 | 1 | 0 | 0 | 1 |
| --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

**II. OUTPUT FORMAT**

The output are 2 matrices - **d, r** and a **budget**. Following is the description of the **Reliable Network** problem [1]. We are given two *n × n* matrices, a distance matrix *d₍ᵢⱼ₎* and a *connectivity requirement* matrix *r₍ᵢⱼ₎*, as well as a budget *b*; we must find a graph *G = ({1, 2, ..., n}, E)* such that

(1) the total cost of all edges is *b* or less, and

(2) between any two distinct vertices *i* and *j* there are *r₍ᵢⱼ₎* vertex-disjoint paths.

*Sample Output:*

d\_ij -

| 0 | 1 | 2 | 2 | 1 |
| --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 2 | 1 |
| 2 | 1 | 0 | 1 | 2 |
| 2 | 2 | 1 | 0 | 1 |
| 1 | 1 | 2 | 1 | 0 |

r\_ij -

| 0 | 2 | 2 | 2 | 2 |
| --- | --- | --- | --- | --- |
| 2 | 0 | 2 | 2 | 2 |
| 2 | 2 | 0 | 2 | 2 |
| 2 | 2 | 2 | 0 | 2 |
| 2 | 2 | 2 | 2 | 0 |

budget = 5

# **III. TRANSFORMATION**

## *A. Construction of Reliability and Distance Matrices*

We take a Rudrata cycle graph as input in form of a file, and for the distance matrix **d,** disperse the edges into real and non-real (real edges are assigned a value of 1, while non real edges are assigned 2, diagonal values are set to 0).

For the reliability matrix **r,** we construct a matrix of the same size with all entries equal to 2 (except the diagonal entries which are all 0).

This transformation works because now each pair of vertices needs to have exactly 2 vertex disjoint paths between them. This is not possible except for a simple cycle that covers all vertices. The budget inhibits taking any edges which did not belong in the rudrata cycle (otherwise in trying to cover all vertices, the budget will become more than n) Thus a solution to this reliable network problem is possible if and only if there is a rudrata cycle in the original graph. The matrices thus formed are plotted for visualization.

The runtime of our transformation is **O(n^2)** as editing each pair value requires a minimum of this much runtime. We have used vectorized NumPy operations for efficiency, however for very large n the runtime would still be O(n^2) as the total memory is finite and would not be able to scale.

## *B. Getting a Rudrata (Hamiltonian) Cycle from a solution to reliable network*

The solution to the reliable network problem is a graph. But since this is a transformed problem from the rudrata cycle, the solution graph must be a cycle. Thus it is the exact same graph which is also the solution to the original rudrata cycle problem!

## *C. Solving for Reliable network*

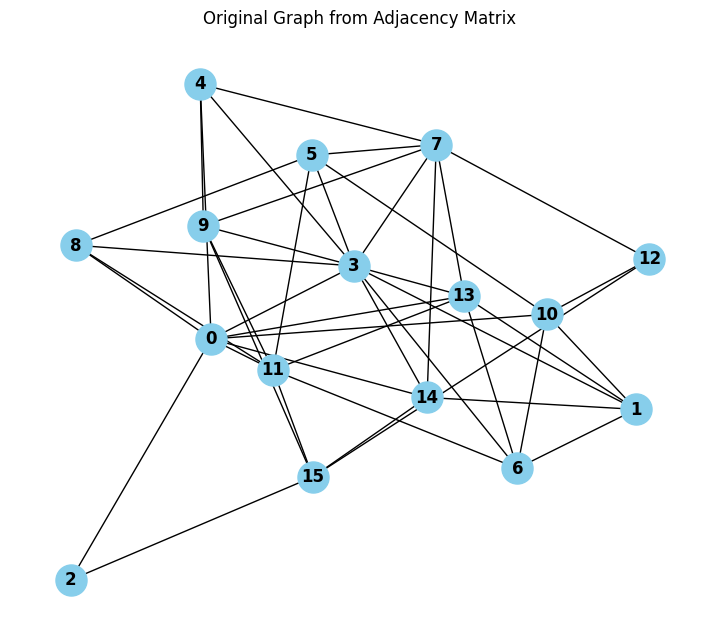
We now move forward with solving for a reliable network. Our solution uses brute force with backtracking and pruning which tests relevant edge subsets in the graph.

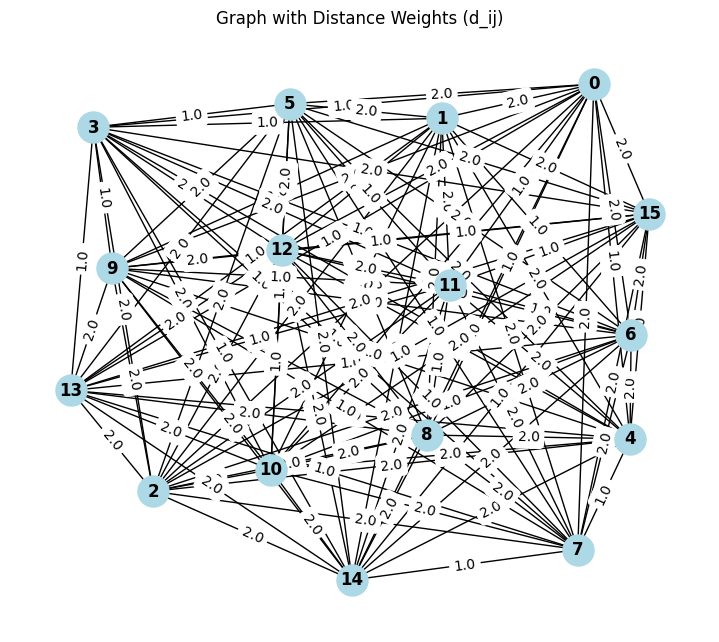
Backtracking explores all the subsets by making a decision at each step of including or excluding an edge in the current subset. Pruning is used to cut off branches early when we know that the current set cannot be part of a solution, and so is not worth exploring.

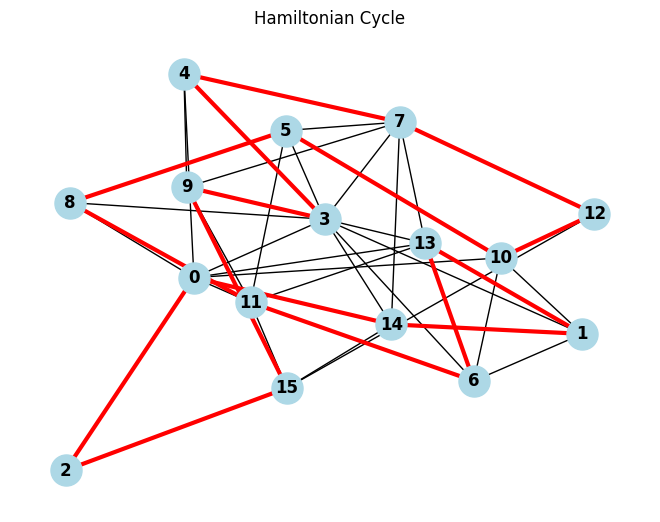
Backtracking is implemented via recursion on the backtrack() function. Pruning is implemented by checking if the set has exceeded the budget already or if the best solution is already found.

Our function halts when it takes more than a minute to check the solution. We were unable to come up with a quicker brute force checker for the Reliable Network problem, and so although our brute force for Rudrata Cycle problem is able to run quickly for a 16 node graph, this function is unable to run quickly for n =`16. However, we have verified solutions for n = 5 in it.

**IV. SAMPLE INPUT AND OUTPUT FIGURES**







**V. PROGRAMMING LANGUAGE AND EXTERNAL LIBRARIES USED**

* Python 3.10
* Matplotlib [2]
* Numpy [3]
* Networkx

# VI. CONCLUSIONS

From this project we have demonstrated that a polynomial time transformation is feasible from the Rudrata Cycle problem to the Reliable Network problem. Our brute force algorithms verify that whenever there is a solution for one problem, there is also a solution to the other problem. Hence our transformation is verified. For future work, we will attempt to make our reliable network brute force function more quicker such that larger graphs can be verified.

## 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.