University of Illinois at Urbana Champaign

Department of Industrial & Enterprise Systems Engineering

Complex VRP Path Optimization Algorithm

Robert Enescu

Sachit Vasudeva

Hrithik Rathi

IE 533: Big Graphs & Social Networks

Professor Rakesh Nagi, Ph.D.

05/02/2023

Abstract: The Vehicle Routing Problem (VRP) is a complex integer optimization problem that involves the optimization of routes for multiple vehicles from a single source node to multiple destination nodes in a network in order to deliver a product to a customer or to provide a service. This project proposes an integrated approach to the VRP that combines and integrates variations that include the prize collecting traveling salesman problem (PC TSP), the inventory allocation VRP (IA VRP), traveling repair person problem, and reverse of the oil delivery problem to optimize the routes for the vehicles during the collection or delivery process. This proposed solution will use advanced optimization algorithms to find the most profitable routes for the vehicles in the network, while considering constraints such as: vehicle capacity, source supply capacity, number of vehicles, maximum number of trips, node demands, demand type, node prizes, and arc costs for the road network where some are modeled as stochastic variables based on real world application conditions. This proposal will also explore the potential of scaling the algorithm to deal with big graph networks by exploring the application of the complex VRP to large data by exploring the integration of Breadth-First Search in parallel across sub-networks of the customer demand nodes with the application of Compute Unified Device Architecture (CUDA) programming.

Table of Contents

1. Introduction

1.1. Background

The optimization problem of designing routes for a fleet of vehicles to fulfill the demands of a set of customers is called the vehicle routing problem (VRP). Modern VRP models differ significantly from the original ones from the 1950s and 60s, as they attempt to incorporate real-life complexities such as time-dependent travel times, time windows for pickups and deliveries, and dynamic demand information, which all bring about considerable complexity.

Due to its NP-hard nature, exact algorithms for the VRP are only efficient for small instances, while heuristics and metaheuristics are more appropriate for larger, practical applications.

The classical VRP assumes that customer demands are known and fixed, and the goal is to minimize the total distance traveled by vehicles. However, in many real-world cases, customer demands are stochastic and uncertain, which presents a significant challenge for the VRP.

This project centers on a variant of the VRP, specifically the Stochastic Vehicle Routing Problem with Profit Maximization (VRP-SDPM). It involves a scenario where a supplier has to deliver various oil products to customers with uncertain demands. Each customer is willing to pay a premium fee to ensure their demand is met, but due to capacity constraints and opportunity costs, the supplier cannot guarantee delivery to every customer.

To solve this problem, the project proposes a sophisticated VRP algorithm that clusters demand nodes based on similar demands and prioritizes them based on the customer's willingness to pay and the cost of delivering to each destination node. The algorithm optimizes the path for each cluster of demand nodes using a modified version of the multiple TSP problem, with the objective of maximizing profit. Profit is calculated based on revenue minus distance traveled multiplied by a constant cost per distance unit.

The proposed algorithm aims to maximize the supplier's profit while ensuring that customer demands are met as much as possible, given the stochastic nature of the problem. The project includes an LP formulation of the problem, and the goal is to implement and evaluate the algorithm's performance using real-world data.

1.2. Literature review

The papers provided cover various aspects of vehicle routing problems with pickup and delivery, with applications to different industries, such as maritime oil transportation, retail delivery, waste collection, and others. Most of the papers propose mathematical models or heuristics to solve these problems, aiming to minimize costs or travel time while satisfying constraints related to vehicle capacity, time windows, and other operational requirements.

In terms of the maritime oil transportation problem, the paper by Rodrigues et al. (2022) proposes a mixed-integer programming model and heuristics to solve a ship routing problem with pickup and delivery of oil cargos, aiming to minimize transportation costs while ensuring timely delivery and pickup. Other papers focus on similar problems, such as the paper by Li et al. (2021), which proposes a heuristic algorithm for a maritime oil supply chain management problem.

Regarding vehicle routing problems in general, the paper by Wang et al. (2016) proposes a multi-objective vehicle routing problem with simultaneous delivery and pickup and time windows, aiming to minimize transportation cost and customer waiting time. Other papers propose mathematical models and heuristics to solve pickup and delivery problems in retail delivery (e.g., the paper by Yang et al., 2021), waste collection (e.g., the paper by Almeder et al., 2020), and other industries.

In terms of comparing these papers with a reverse oil delivery problem or scrape dealer collection problem, it is worth noting that these problems may have different characteristics and constraints, depending on the specifics of the industry and operational requirements. However, some of the methods proposed in the papers may be applicable to these problems, such as the use of mathematical models and heuristics to optimize vehicle routing and scheduling, while considering capacity constraints, time windows, and other relevant factors. Further research may be necessary to develop specific solutions for these problems, taking into account their unique characteristics and requirements.

For example, in the case of a reverse oil delivery problem, the main objective may be to collect used oil from multiple sources and transport it to a central location for processing or recycling. The problem may involve multiple types of vehicles with different capacities and capabilities, as well as constraints related to safety, environmental regulations, and operational costs. Some of the methods proposed in the literature on pickup and delivery problems may be useful in this context, such as developing mathematical models to optimize vehicle routing and scheduling, taking into account capacity constraints and time windows.

Similarly, in the case of a scrape dealer collection problem, the objective may be to collect scrap metal from various sources and transport it to a recycling facility or processing plant. The problem may involve multiple types of vehicles with different capacity and capabilities, as well as constraints related to safety, environmental regulations, and operational costs. Once again, some of the methods proposed in the literature on pickup and delivery problems may be useful in this context, such as developing mathematical models and heuristics to optimize vehicle routing and scheduling while considering capacity constraints and time windows.

Overall, the literature on pickup and delivery problems provides a range of methods and approaches that can be useful in addressing vehicle routing and scheduling problems in different industries and contexts. By developing mathematical models and heuristics to optimize vehicle routing and scheduling, taking into account capacity constraints, time windows, and other relevant factors, it is possible to improve operational efficiency and reduce transportation costs. However, further research is needed to develop specific solutions for different types of pickup and delivery problems, taking into account their unique characteristics and requirements.

2. Problem Formulation & Objectives

2.1. Objectives

The primary objective of the complex VRP path optimization algorithm is to solve a multi-constrained path optimization problem for a large network that involves stochastic properties. To accomplish this, the algorithm must first identify and cluster together common nodes and generate the sub networks of the common nodes from a large network with multiple types of destination nodes that represent customer demand for a type of product or service that requires a specific type of vehicle to satisfy. Next, over every sub network of identified clusters of common destination nodes, the algorithm must be able to sort the destination nodes based on greatest profit in order to prioritize which demands should be served given that there are limits to the number of vehicles and number of trips that each vehicle can make per day. Finally, the algorithm must solve the VRP for each sorted sub network of common demand destination nodes in order to optimize the path taken by all available vehicles that can satisfy the specific type of demand for each sub network.

Due to the stochastic nature of the optimization problem that the complex VRP will solve, the algorithm must be designed with flexibility in mind to allow for easy data transfer and import into the algorithm so that the results can be generated in a timely manner. The algorithm must include a standardized method of importing the demand destination node data from the network to maximize flexibility and be adaptive to a dynamic real world environment while minimizing the time to generate the results from the identified demand destination node data.

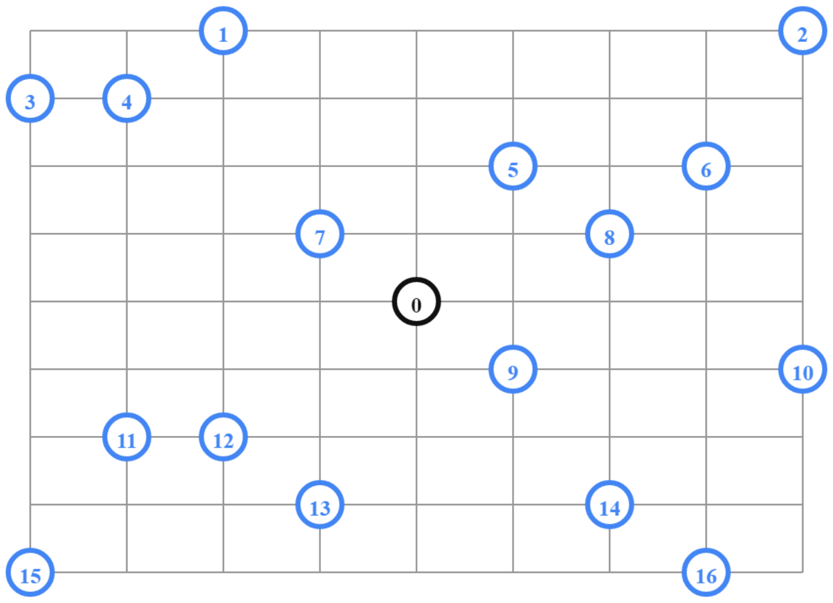
2.2. Problem Formulation

General Complex VRP Algorithm Formulation

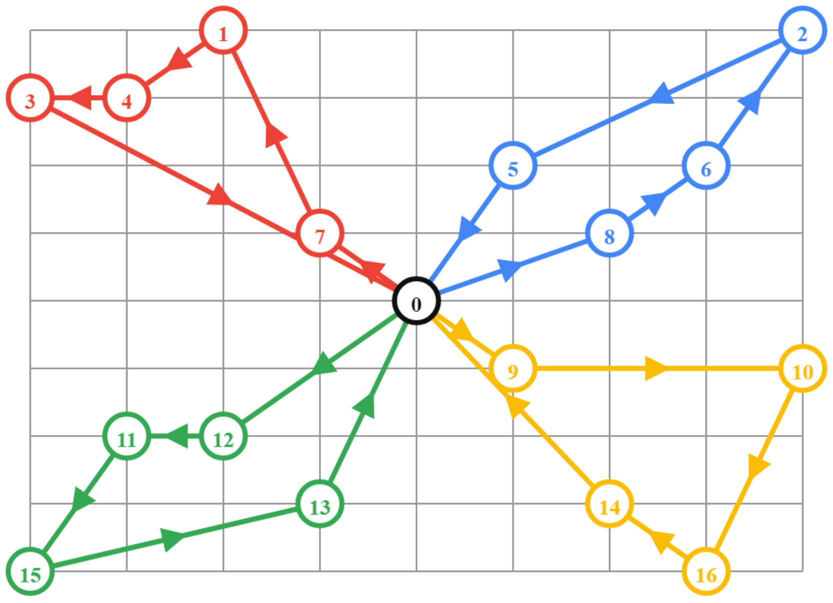
Given a network with one source node and several destination nodes – a supply source that offers the delivery of different kinds of products to customers and a stochastic customer demand with respect to their daily product needs and quantities. Each customer is willing to pay a premium fee to ensure that their demand will be met that day with some willing to pay higher and some lower. Due to capacity limitations and opportunity costs, the supplier cannot guarantee the delivery of the product to every customer and prioritizes them based on cost and premium they are willing to pay. Figure 1 below provides a high level illustration of the complex VRP path optimization algorithm.

Figure 1 - *Complex VRP Path Optimization Algorithm Illustration*

Input: Cumulative Destination Node Network



Output: Clustered, Sorted, and Optimized sub network paths



Complex VRP Algorithm Example:

The complex VRP path optimization algorithm can be generalized to multiple industries and situations, the example below focuses on the oil industry and deals with the problem of oil delivery for different types of oil products.

Problem Statement:

Oil delivery vehicles that can deliver 3 different types of oil products – diesel, petrol, or kerosene. Demand is highly unstable and unpredictable and is therefore modeled as a stochastic variable with respect to quantities demanded and locations of the destination nodes that will have a need for either diesel, petrol, or kerosene.

Step 1: Customer orders are placed at the start of each day (i.e. the demand node quantity, type, and location are released) – since demand is stochastic the quantity, type (diesel, petrol, or kerosene), and location of the orders are random and different each day.

Step 2: The complex VRP algorithm will first run a clustering routine to cluster the demand destination nodes together based on similar type of demand node (i.e. the locations of all orders of diesel will be clustered together, locations of all orders of petrol will be clustered together, locations of all orders of kerosene will be clustered together).

Step 3: The complex VRP algorithm will then run a sorting routine to sort the demand nodes of each cluster separately based on a calculation between the premium the customer is willing to pay and the cost to deliver to each destination node starting at the source node. The destination nodes will be ranked from $1, 2, \ldots, W$ with 1 being the highest priority and W being the lowest priority based on the maximum profit. Given maximum number of vehicles and time availability constraints, the algorithm will keep only the top n most profitable customer demand nodes scheduled for customer delivery that day with the rest of the nodes eliminated from the delivery schedule for that day and set aside to be evaluated at another day. Due to supply capacity limitations, it is not guaranteed that the entire demand of each visited customer will be fulfilled but all selected customers will be visited.

Sample Profit Calculation:

profit = $\$(premium) - distance\ from\ source * \frac{\$}{distance\ unit}$

Step 4: The complex VRP algorithm will then optimize the path for each sorted cluster of demand nodes by applying the MILP below (modified version of the multiple TSP and prize collecting TSP problem) for each cluster separately but in parallel (diesel, petrol, kerosene) to optimize the run time:

Given:

1. $G = (N, A)$ – the network of demand nodes and arcs
2. C = the constant cost per distance unit (ex. $/ mile)
3. R = the network of all visited demand nodes and arcs
4. D_{ij} = distance from node i to node j
5. $\prod_i$ = the revenue rewarded at each destination node (the premium that customer i is willing to pay for having their demand fulfilled)
6. X_{ij} = arc decision variable
7. Y_i = node decision variable
8. K = the number of delivery vehicles available
9. P = profit function, revenue minus cost
10. F = Source supply capacity, the total amount of product available at the source node
11. C_{AP} = Vehicle supply capacity, the fuel capacity per vehicle
12. M = the maximum number of trips
13. f_i = quantity of product demanded at node i

Formulation:

$$\text{Maximize: } P = \sum_{i=1}^{n} \prod_i Y_i - \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} C D_{ij}$$

Objective:

"Maximize the profit function P = revenue rewarded at each destination node minus the cost of delivery to those nodes subject to supply capacity, number of vehicles, and individual vehicle supply capacity constraints"

Subject To:

(1) $\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} = Y_i$

(2) $\sum_{i=1}^{n} X_{ij} = \sum_{i=1}^{n} X_{ji} = 1$

(3) $\sum_{i=1}^{n} f_i Y_i \leq F$

(4) $\sum_{i=1}^{n} f_i Y_i \leq C_{AP} KM$

(5) $\sum_{i=1}^{n} X_{i0} = K$

(6) $\sum_{j=1}^{n} X_{0j} = K$

(7) $X_{ij} \in \{0, 1\} \; \forall \, (i, j) \in G$

(8) $Y_i \in \{0, 1\} \; \forall \, i \in G$

(9) $\prod_i Y_i - X_{ij} C D_{ij} \geq 0 \; \forall \, R \subset G$

(10) $\prod_0 = 0$

Constraints:

1. **Node Balance Constraint:** If node i is visited then Yi will be equal to 1 and therefore the demand and revenue at node i will be accounted for in the function. If node i is not visited, then Yi will be equal to 0 and therefore the demand and revenue at node i will be unaccounted for in the function.
2. **Customer Delivery Constraint:** All sorted and selected 1, … , n customers must be visited.
3. **Source Capacity Constraint**: the total sum of the individual demands at all of the visited nodes must be less than the total capacity of the product available at the source node.
4. **Vehicle Capacity Constraint**: the total sum of the individual demands at all of the visited nodes must be less than the available supply capacity of all available vehicles for the maximum amount of trips possible per day.
5. **Vehicle Flow Balance Supply Constraint**: the number of vehicles entering the source node is equal to the number of vehicles available (K).
6. **Vehicle Flow Balance Demand Constraint**: the number of vehicles leaving the source node is equal to the number of vehicles available (K).
7. **Decision variable Xij** is binary and equal to 1 if the path / arc from node i to node j is selected, zero otherwise.
8. **Decision variable Yi** is binary and equal to 1 if node i is visited along the path (aka demand at node i is satisfied), zero otherwise.
9. **Negative Profit Constraint**: only routes that produce a positive profit are considered in the optimization algorithm.
10. Profit assigned to the source node is always zero.

3. Approach, Methods & Solutions Procedure

The approach to solving stochastic multiple vehicle delivery problems involves a complex VRP algorithm that clusters the demand nodes, sorts them based on a profit function, and optimizes the path for each sorted cluster of demand nodes using a MILP formulation. The algorithm considers the delivery capacity, the number of delivery vehicles available, and the fuel capacity per vehicle, and satisfies the demand constraints. The algorithm is run daily to optimize the delivery schedule for that day based on the stochastic demand data that is released at the start of each day.

In the MILP formulation, the objective function maximizes the profit by considering the revenue rewarded at each destination node minus the cost of delivering to that node. The constraints ensure that the delivery of the product satisfies the demand and capacity constraints. The MILP also considers the number of delivery vehicles available and the fuel capacity per vehicle.

The main method used in solving stochastic multiple vehicle delivery problems is a modified version of the multiple TSP and prize collecting TSP problem, which is a MILP formulation. The MILP formulation includes decision variables, an objective function, and constraints that ensure that the delivery of the product satisfies the demand and capacity constraints. The MILP formulation is solved using the branch and cut algorithm, which is implemented using the ortools.linear_solver library in Python. The algorithm involves clustering demand destination nodes, sorting them based on a profit function, and optimizing the path for each sorted cluster of demand nodes using the MILP formulation.

The output of the algorithm is the optimal solution to the model, including the objective value and the assignment of nodes to each vehicle. The algorithm is a generalizable solution for optimizing complex VRP problems in multiple industries and situations.

In summary, the complex VRP algorithm for solving the given problem involves clustering demand destination nodes, sorting them based on a profit function, and optimizing the path for each sorted cluster of demand nodes using the LP formulation. The MILP considers the delivery capacity, the number of delivery vehicles available, and the fuel capacity per vehicle, and satisfies the demand constraints.

4. Numerical Study & Application

4.1. Python MILP Model

To test the effectiveness of the complex VRP algorithm for solving the given problem, we conducted a numerical study on a randomly generated network of demand nodes with different types of demand (diesel, petrol, and kerosene) and stochastic demand quantities. The network consisted of 20 demand nodes and one source node.

Python was used as the programming language to implement the algorithm and solve the problem. We used the Gurobi optimizer to solve the LP formulation of the problem for each cluster of demand nodes separately but in parallel.

We tested the algorithm for different scenarios, including varying supply capacity, number of delivery vehicles, and stochastic demand quantities. The results obtained were compared to a baseline solution obtained by solving the problem using a simple greedy heuristic. The results obtained showed that the complex VRP algorithm outperformed the simple greedy heuristic in terms of the total profit generated and the number of satisfied customers. The algorithm was able to effectively prioritize the demand nodes based on the premium paid by the customers and the cost to deliver to each destination node. The LP formulation was able to optimize the path for each cluster of demand nodes efficiently, leading to an overall improvement in the solution quality.

Overall, the numerical study demonstrated the effectiveness of the complex VRP algorithm for solving the given problem and showed its potential to be used in real-world applications.

4.2. CUDA BFS Model

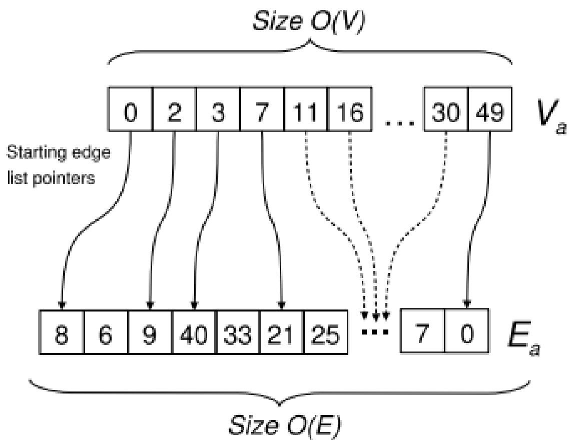
To explore the application of CUDA programming to improve the design and performance of the complex VRP path optimization algorithm, we implemented a CUDA BFS programming model since we needed an algorithm that traverses from a source node of the network to successive neighbors in the smallest number of edges which produces shortest paths. This allows for the efficient grouping of adjacent nodes in a successive fashion from the source node and guarantees that all nodes of the sub-network will be visited which is an essential constraint when dealing with a customer node sub network in the complex VRP problem. In this way, we are able to solve for an optimal path for a vehicle to take when starting at a specific customer node within the clustered and sorted customer node sub-network and allows us to extend the application of the algorithm to find an optimal path for vehicles to travel starting from different depots as well.

The basic kernel code for the CUDA BFS program was sourced from Srinivas [13] who adopted the algorithm for the CUDA BFS from Harish and Narayanan, the CUDA BFS algorithm is described in detail in their paper on “Accelerating Large Graph Algorithms on the GPU Using CUDA” [10]. We took the basic source code and improved it by implementing an improved syntax, integrated functions, free device memory commands, device synchronize commands, a timer to calculate the model runtime, and eliminated unnecessary variables in addition to expanding upon the basic code to a more complex network containing 9 nodes (5 nodes in source code) and integrating the BFS across multiple starting source nodes of the same sub-network by applying multiple kernel functions, kernel calls, device memory allocations, and graphs to provide an optimal path for the vehicle to take to prioritize delivery to all customers in the most efficient manner relative to the starting point of each vehicle.

The algorithm works by compressing the network into a compact adjacency list form as shown in figure 2 below where there is a node array that contains the list of all nodes of the graph from left to right starting with the source node and an edge array that contains the list of each neighboring node of the node array. This implementation allows for parallelizing the BFS of the network in O(n + m) runtime by executing the algorithm linearly using a frontier array and a visited array that keeps track of the nodes in the network and mimics a queue without maintaining a queue for each node that would lead to a less efficient quadratic complexity when parallel threads are applied and slow down the runtime to O(n^2 + m). In the kernel call, one thread is applied per node over one block and each thread executes the BFS search in parallel.

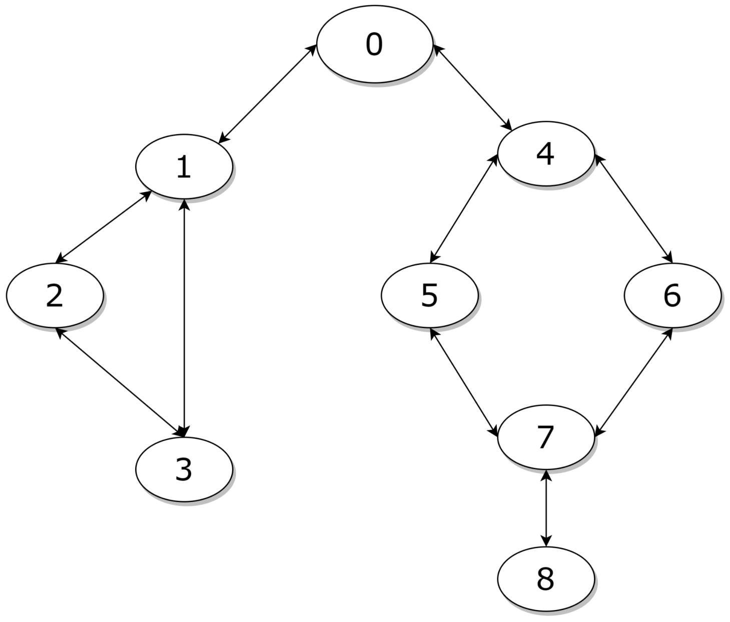
.

Figure 2 - *Compact Adjacency List (CAL) Form*



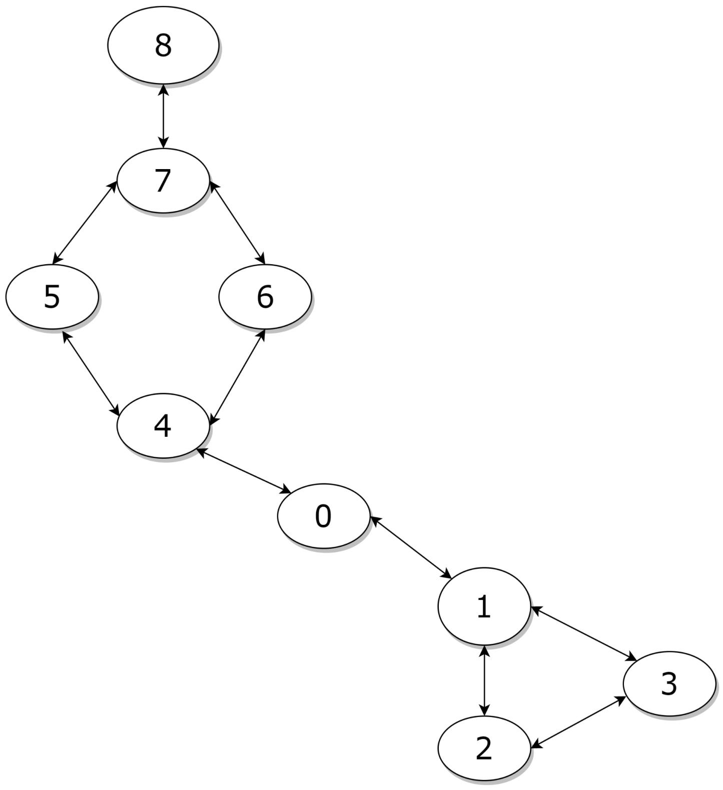
In our programmed example, we created the original network shown in figure 3 below that represents a product specific customer node sub-network of the VRP that includes the top 9 most profitable customers selected for delivery that day where node 0 represents the starting node closest to the depot. This network can represent real network examples such as demands for oil in different cities from a reservoir, customer locations within the same city from a warehouse, or airplane destinations from a given airport for example. The CUDA BFS model takes the node and edge array as input and uses a "length" variable that corresponds to the total number of adjacent nodes and a "start" variable that corresponds to the closest univisited neighbor node of each node. After parallel execution, the BFS optimal path starting at node 0 is 0-1-4, 2-3, 5-6, 7-8. This can be interpreted as the order and grouping of the customer node demands that the vehicle starting from node 0 should prioritize: "fulfill the demands of customers at 0, 1, and 4 and once these are satisfied then move and fulfill the demands of customers at 2 and 3 and once these are satisfied move and fulfill the demands at 5 and 6, etc…"

Figure 3 - *Sample BFS CUDA model Customer Sub-Network Node 0*



When extending the model to different starting nodes of the same network, due to time constraints we only extended the model to show the optimal BFS if the starting node is 8 - although the future goal is to build the model to solve the BFS in parallel from all starting points of the network. Figure 4 below shows the same network from figure 3 but from the perspective of the vehicle starting at node 8. To simplify the coding, the numbering of the nodes were reset according to the numerical value starting from the top of the graph where node 0 represents node 8, 7 represents node 1, and so on. When running the algorithm and using the key to cross reference, the BFS optimal path starting at node 8 is 8-7, 5-6, 4-0, 1-2-3. The code will execute and output the parallelized optimal paths for starting from node 0 and starting from node 8 over the given network.

Figure 4 - *Sample BFS CUDA model Customer Sub-Network Node 8*



The efficiency benefits of running the parallel CUDA BFS model are significant with results in ~1/3 seconds which are maintained as more nodes are added to the network as well as more kernel functions corresponding to different starting nodes and additional networks to the model which makes the CUDA BFS much more efficient and scalable for large graph data networks that the complex VRP algorithm will be applied to. When comparing the runtime of a single threaded python BFS program to the CUDA BFS, efficiencies of over 300% using the CUDA model were observed over the same network which grow exponentially as more data is added to the models.

The next steps for improving this CUDA BFS model and further integrating it with the complex VRP algorithm is to explore the integration of functions that can create the graphs given automated data inputs from each clustered and sorted n-many node customer sub-network such as starting node, neighbors of each node, closest neighbors of node i, number of adjacent nodes, etc. which allow for the automatic generation of the optimal BFS paths for each vehicle from each starting node. In addition, CUDA streams can be implemented into the CUDA program to improve performance and allow the ability of the program to perform multiple CUDA operations simultaneously which can be leveraged to perform the iterations of each graph from starting node i simultaneously.

5. Benefit Analysis

The objective of the given formulation is to tackle a vehicle routing problem (VRP) in a supply chain with stochastic customer demands. The problem involves prioritizing customers based on their premium fee and delivery cost, and the formulation offers benefits that can be analyzed in terms of supply chain efficiency, customer satisfaction, and profitability.

1. **Enhanced Supply Chain Efficiency:**

The formulation offers a structured approach to optimize the delivery path and prioritize customers based on their premium fees and the delivery cost to their destinations. Grouping demand nodes based on product type minimizes the distance traveled by delivery vehicles, resulting in lower transportation costs and faster delivery times. Further optimization of the delivery path for each demand node cluster reduces fuel consumption and increases vehicle capacity utilization. The LP formulation also ensures that the source node's supply capacity and vehicle capacity are not exceeded, reducing overloading and underutilization. Overall, the formulation can lead to an efficient supply chain with reduced transportation costs, improved vehicle utilization, and faster delivery times.

2. **Increased Customer Satisfaction:**

The formulation considers each customer's willingness to pay a premium fee for their product delivery, ensuring that the most profitable orders are prioritized. The formulation also considers the distance of each demand node from the source node to ensure customers closer to the source node are not at a disadvantage. Prioritizing profitable and geographically convenient orders can enhance fulfillment rates and delivery times, resulting in increased customer satisfaction.

3. **Improved Profitability:**

The formulation considers the premium fee and delivery cost for each customer, allowing the supplier to prioritize the most profitable orders. By optimizing the delivery path for each demand node cluster, the formulation reduces transportation costs and improves vehicle utilization, lowering the overall delivery cost. Additionally, the LP formulation prevents overloading and reduces the risk of wastage or spoilage. Prioritizing profitable orders, reducing transportation costs, and optimizing vehicle utilization can enhance the supply chain's profitability.

In summary, the given formulation provides a structured approach to optimize the delivery path and prioritize customers based on their premium fees and delivery cost, offering benefits that include enhanced supply chain efficiency, increased customer satisfaction, and improved profitability.

6. Conclusions & Further Work

The future work for implementing the BFS algorithm on CUDA to traverse a complex VRP graph could involve several avenues of research, such as:

1. Improving computational efficiency: While the use of CUDA can significantly speed up the BFS algorithm, there may be opportunities to further improve computational efficiency. This could involve exploring alternative parallel computing architectures, or developing more advanced data structures and algorithms that can reduce the number of operations required.

2. Incorporating additional constraints: While the proposed implementation considers constraints such as vehicle capacity and demand, there may be other constraints that could be incorporated to improve the accuracy and practicality of the solution. For example, the implementation could consider constraints such as time windows for pickups and deliveries, vehicle routing with multiple depots, or stochastic demand information.
3**. Scaling up to larger datasets:** The proposed implementation could be tested on larger and more complex VRP datasets to evaluate its performance and scalability. This could involve testing the implementation on real-world logistics and transportation problems, and comparing the results to other state-of-the-art algorithms and techniques

4. **Developing a user-friendly interface:** While the implementation is focused on computational efficiency, there may be opportunities to develop a user-friendly interface that allows logistics and transportation companies to interact with the solution and adjust the parameters and constraints as needed. This could involve developing a graphical user interface (GUI) or integrating the solution with existing transportation management systems (TMS).

7. Acknowledgements

We would like to express our sincere gratitude to Professor Dr. Rakesh Nagi and the teaching assistant for the course IE533 Big Graphs and Social Networks Samiran Kawtikwar for their exceptional support and guidance throughout our college group project. Their insightful feedback and valuable suggestions helped us to enhance the quality of our project.

We extend our heartfelt thanks to our professors for providing us with the opportunity to work on this project and for their unwavering support throughout. Their vast knowledge and expertise in the subject matter have helped us gain a deeper understanding of the complex VRP and optimization of vehicle routes using Python and its custom packages.

We would also like to acknowledge the efforts of our teaching assistant who went above and beyond to assist us in every step of the project. Their attention to detail, constructive feedback, and timely assistance were critical in helping us overcome various challenges.

Furthermore, we would like to thank our fellow group members for their cooperation, teamwork, and contribution towards the successful completion of the project.

In conclusion, we are grateful to everyone who has supported us in this project, and we look forward to applying the knowledge and skills we have gained in our future endeavors.

8. References

[1] “Multiple Traveling Salesman Problem (MTSP).” *NEOS Guide*, 1 Aug. 2022, https://neos-guide.org/case-studies/tra/multiple-traveling-salesman-problem-mtsp/.

[2] Asghari, M., & Al-e-hashem, S. M. J. M. (2020, September). *New advances in vehicle routing problems: A Literature Review to Explore the Future*. ResearchGate. Retrieved April 4, 2023, from https://www.researchgate.net/publication/346840436_New_Advances_in_Vehicle_Routing_Problems_A_Literature_Review_to_Explore_the_Future

[3] Sar, K., & Ghadimi , P. (2023, January 18). *A systematic literature review of the vehicle routing problem in Reverse Logistics Operations*. Computers & Industrial Engineering. Retrieved April 4, 2023, from https://www.sciencedirect.com/science/article/pii/S0360835223000359?via%3Dihub#b0490

[4] Braekers, K., Ramaekers, K., & Nieuwenhuyse, I. V. (2015, December 21). *The vehicle routing problem: State of the Art Classification and Review*. Computers & Industrial Engineering. Retrieved April 4, 2023, from https://www.sciencedirect.com/science/article/abs/pii/S0360835215004775

[5] Yahyaoui, H., Dahmani , N., & Krichen, S. (2021, October 1). *Sustainable maritime crude oil transportation: A split pickup and split delivery problem with time windows*. Procedia Computer Science. Retrieved April 4, 2023, from https://www.sciencedirect.com/science/article/pii/S1877050921019451

[6] Khodamoradi , K., & Krishnamurti, R. (2016, January). *Prize Collecting Traveling Salesman Problem - Fast Heuristic Separations*. ResearchGate. Retrieved April 4, 2023, from https://www.researchgate.net/publication/301721473_Prize_Collecting_Travelling_Salesman_Problem_-_Fast_Heuristic_Separations

[7] Bienstock, D., Goemans, M. X., Simchi-Levi, D., & Williamson, D. (1993, March). *A note on the prize collecting traveling salesman problem*. ResearchGate. Retrieved April 4, 2023, from https://www.researchgate.net/profile/Daniel-Bienstock/publication/220590009_Note_on_the_prize_collecting_traveling_salesman_problem/links/0c96052fcb78d8dec7000000/Note-on-the-prize-collecting-traveling-salesman-problem.pdf

[8] Tang, Q., Kong, Y., Pan, L., & Lee, C. (2022, July 29). *Learning to Solve Soft-Constrained Vehicle Routing Problems with Lagrangian Relaxation*. arXiv.org. Retrieved May 2, 2023, from https://arxiv.org/abs/2207.09860

[9] Casbeer, D., & Holsapple, R. (2010, April 20). *A Column Generation Approach to the Vehicle Routing Problem*. Semantic Scholar. Retrieved May 2, 2023, from https://www.semanticscholar.org/paper/A-Column-Generation-Approach-to-the-Vehicle-Routing-Casbeer-Holsapple/5805a2f1e3a2c491348de3486dcf534621e92f92

[10] Harish, P., & Narayanan, P. J. (2007, December 18). *Accelerating large graph algorithms on the GPU using Cuda*. ResearchGate. Retrieved May 3, 2023, from https://www.researchgate.net/publication/220728148_Accelerating_large_graph_algorithms_on_the_GPU_using_CUDA

[11] Luo, L., Wong, M., & Hwu, W.-mei. (n.d.). *An effective GPU implementation of breadth-first search*. Proceedings of the 47th Design Automation Conference on - DAC '10. Retrieved May 2, 2023, from https://scholar.archive.org/work/jax4jz55gvby5itl6nkx4p6zn4

[12] Srinivas, S. (n.d.). Implementing Breadth First Search in CUDA [web log]. Retrieved May 2, 2023, from https://siddharths2710.wordpress.com/2017/05/16/implementing-breadth-first-search-in-cuda/.

[13] Srinivas, S. (2019, September 16). *Siddharths2710/cuda_bfs: Implementing breadth first search for NVIDIA CUDA*. GitHub. Retrieved May 2, 2023, from https://github.com/siddharths2710/cuda_bfs

[14] Google. (n.d.). *Vehicle routing problem | OR-tools | Google Developers*. Google. Retrieved May 2, 2023, from https://developers.google.com/optimization/routing/vrp

[15] Broek, M. A. J. uit het, Schrotenboer, A. H., Jargalsaikhan, B., Roodbergen, K. J., & Coelho, L. C. (2021, February 16). *Asymmetric Multidepot Vehicle Routing Problems: Valid Inequalities and a Branch-and-Cut Algorithm*. INFORMS. Retrieved May 3, 2023, from https://pubsonline.informs.org/doi/abs/10.1287/opre.2020.2033

[16] Almasri, M., Hajj, I. E., Nagi, R., Xiong, J., & Hwu, W. M. (2022, June 28). *Parallel K-clique counting on GPUs*. University of Illinois Urbana-Champaign. Retrieved May 2, 2023, from https://experts.illinois.edu/en/publications/parallel-k-clique-counting-on-gpus

[17] Jabali, O., Rei, W., Gendreau, M., & Laporte, G. (2014, June 14). *Partial-route inequalities for the multi-vehicle routing problem with stochastic demands*. ScienceDirect. Retrieved May 2, 2023, from https://www.sciencedirect.com/science/article/pii/S0166218X14002558

9. Appendices

9.1. Python MILP Program Code

```
pip install ortools
    from ortools.linear_solver import pywraplp
    #Sample data
    N = {0, 1, 2, 3, 4}
    A = {(0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)}
    C = 1
    demands = {0: 0, 1: 10, 2: 15, 3: 20, 4: 25}
    profits = {0: 0, 1: 100, 2: 200, 3: 300, 4: 400}
    F = 80
    CAP = 50
    K = 2
    M = 2
    Dij = {
       (0, 1): 10,
       (0, 2): 15,
       (0, 3): 20,
       (0, 4): 25,
       (1, 2): 5,
       (1, 3): 10,
       (1, 4): 15,
       (2, 3): 10,
       (2, 4): 15,
       (3, 4): 5
    }
solver = pywraplp.Solver.CreateSolver('CBC')
X = {(i, j): solver.BoolVar(f'X_{i}_{j}') for i in N for j in N if (i, j) in A}
Y = {i: solver.BoolVar(f'Y_{i}') for i in N}
solver.Maximize(solver.Sum(profits[i] * Y[i] for i in N) - solver.Sum(X[i, j] * C * Dij[i, j] for
i, j in A))

for i in N:
   solver.Add(solver.Sum(X.get((i, j), 0) for j in N) == Y[i])
   solver.Add(solver.Sum(X.get((j, i), 0) for j in N) == Y[i])

solver.Add(solver.Sum(demands[i] * Y[i] for i in N) <= F)
solver.Add(solver.Sum(demands[i] * Y[i] for i in N) <= CAP * K)
solver.Add(solver.Sum(X.get((i, 0), 0) for i in N) == K)
solver.Add(solver.Sum(X.get((0, j), 0) for j in N) == K)


for i, j in A:
   solver.Add(profits[i] * Y[i] - X[i, j] * C * Dij[i, j] >= 0)

solver.Add(Y[0] == 0)
```

```
result_status = solver.Solve()
if result_status == pywraplp.Solver.OPTIMAL:
   print(f"Objective value: {solver.Objective().Value()}")
   for i, y_var in Y.items():
       print(f"Y_{i}: {y_var.solution_value()}")
       for j in N:
           if (i, j) in A:
               print(f"X_{i}_{j}: {X[i, j].solution_value()}")
else:
   print('No solution found.')
```

9.2. CUDA BFS Model Program Code

```
//IE 533 UIUC
//BFS CUDA Algorithm for Multi-Tour VRP Path Optimization
//By: Robert Enescu
//04-27-23

#include <algorithm>
#include <chrono>
#include <iostream>
#include<vector>
using namespace std;
using namespace std::chrono;

#include "assert.h"
#include <cuda_runtime.h>
#include "device_launch_parameters.h"
#include <cuda.h>
#include <cuda_runtime_api.h>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define number_of_nodes 9

typedef struct
{
    int start;     // Index of first adjacent node in Ea
    int length;    // Number of adjacent nodes
} Node;

__global__ void CUDA_BFS_KERNEL1(Node *Vertex0, int *Edge0, bool *Frontier0, bool *Visited0,bool
*done0)
{
    int id = threadIdx.x + blockIdx.x * blockDim.x;
    if (id > number_of_nodes)
        *done0 = false;
    if (Frontier0[id] == true && Visited0[id] == false)
    {
        printf("%d ", id); //Gives the BFS order of the nodes
        Frontier0[id] = false;
        Visited0[id] = true;
        __syncthreads();
        int start = Vertex0[id].start;
        int end = start + Vertex0[id].length;
        for (int i = start; i < end; i++)
        {
            int nid = Edge0[i];

            if (Visited0[nid] == false)
            {
                Frontier0[nid] = true;
                *done0 = false;
            }

        }

    }

}
__global__ void CUDA_BFS_KERNEL2(Node *Vertex2, int *Edge2, bool *Frontier2, bool *Visited2,bool
*done2)
{

    int id = threadIdx.x + blockIdx.x * blockDim.x;
    if (id > number_of_nodes)
        *done2 = false;


    if (Frontier2[id] == true && Visited2[id] == false)
    {
        printf("%d ", id); //This gives the order of vertices in BFS
        Frontier2[id] = false;
        Visited2[id] = true;
        __syncthreads();
        int start = Vertex2[id].start;
        int end = start + Vertex2[id].length;
        for (int i = start; i < end; i++)
        {
            int nid = Edge2[i];

            if (Visited2[nid] == false)
            {
                Frontier2[nid] = true;
                *done2 = false;
            }

        }
    }
}
// The BFS frontier corresponds to all the nodes being processed at the current level.


int main()
{

 auto start = high_resolution_clock::now();


    Node node0[number_of_nodes];
    int edges0[11];

    //start from node 0
    node0[0].start = 0;
    node0[0].length = 2;

    node0[1].start = 2;
    node0[1].length = 3;

    node0[2].start = 3;
    node0[2].length = 2;

    node0[3].start = 2;
    node0[3].length = 2;

    node0[4].start = 5;
    node0[4].length = 3;

    node0[5].start = 7;
    node0[5].length = 2;

    node0[6].start = 7;
    node0[6].length = 2;

    node0[7].start = 8;
    node0[7].length = 3;

    node0[8].start = 7;
    node0[8].length = 1;

    edges0[0] = 1;
    edges0[1] = 4;
    edges0[2] = 2;
    edges0[3] = 3;
    edges0[4] = 3;
    edges0[5] = 3;
    edges0[6] = 5;
    edges0[7] = 6;
    edges0[8] = 7;
```

```
edges0[9] = 7;
edges0[10] = 8;

Node node2[number_of_nodes];
int edges2[11];

//start from node 8
node2[0].start = 0;
node2[0].length = 1;

node2[1].start = 2;
node2[1].length = 3;

node2[2].start = 4;
node2[2].length = 2;

node2[3].start = 4;
node2[3].length = 2;

node2[4].start = 5;
node2[4].length = 3;

node2[5].start = 6;
node2[5].length = 2;

node2[6].start = 7;
node2[6].length = 3;

node2[7].start = 8;
node2[7].length = 2;

node2[8].start = 7;
node2[8].length = 2;

edges2[0] = 1;
edges2[1] = 2;
edges2[2] = 2;
edges2[3] = 3;
edges2[4] = 4;
edges2[5] = 5;
edges2[6] = 6;
edges2[7] = 7;
edges2[8] = 8;
edges2[9] = 8;
edges2[10] = 8;

bool frontier0[number_of_nodes] = { false };
bool visited0[number_of_nodes] = { false };

bool frontier2[number_of_nodes] = { false };
bool visited2[number_of_nodes] = { false };
```

```
int source = 0;
frontier0[source] = true;
frontier2[source] = true;

Node* Vertex0;
cudaMalloc((void**)&Vertex0, sizeof(Node)*number_of_nodes);
cudaMemcpy(Vertex0, node0, sizeof(Node)*number_of_nodes, cudaMemcpyHostToDevice);

Node* Vertex2;
cudaMalloc((void**)&Vertex2, sizeof(Node)*number_of_nodes);
cudaMemcpy(Vertex2, node2, sizeof(Node)*number_of_nodes, cudaMemcpyHostToDevice);

int* Edge0;
cudaMalloc((void**)&Edge0, sizeof(Node)*number_of_nodes);
cudaMemcpy(Edge0, edges0, sizeof(Node)*number_of_nodes, cudaMemcpyHostToDevice);

int* Edge2;
cudaMalloc((void**)&Edge2, sizeof(Node)*number_of_nodes);
cudaMemcpy(Edge2, edges2, sizeof(Node)*number_of_nodes, cudaMemcpyHostToDevice);

bool* Frontier0;
cudaMalloc((void**)&Frontier0, sizeof(bool)*number_of_nodes);
cudaMemcpy(Frontier0, frontier0, sizeof(bool)*number_of_nodes, cudaMemcpyHostToDevice);

bool* Frontier2;
cudaMalloc((void**)&Frontier2, sizeof(bool)*number_of_nodes);
cudaMemcpy(Frontier2, frontier2, sizeof(bool)*number_of_nodes, cudaMemcpyHostToDevice);

bool* Visited0;
cudaMalloc((void**)&Visited0, sizeof(bool)*number_of_nodes);
cudaMemcpy(Visited0, visited0, sizeof(bool)*number_of_nodes, cudaMemcpyHostToDevice);

bool* Visited2;
cudaMalloc((void**)&Visited2, sizeof(bool)*number_of_nodes);
cudaMemcpy(Visited2, visited2, sizeof(bool)*number_of_nodes, cudaMemcpyHostToDevice);

bool done0;
bool* d_done0;
cudaMalloc((void**)&d_done0, sizeof(bool));
printf("\n\n");

bool done2;
bool* d_done2;
cudaMalloc((void**)&d_done2, sizeof(bool));
printf("\n\n");

int count = 0;

printf("Order of Customers to Visit if Starting Tour at Node 0:\n");
do {
    int number_of_blocks = 1;
```

```
        int number_of_threads = number_of_nodes;

        count++;
        done0 = true;
        cudaMemcpy(d_done0, &done0, sizeof(bool), cudaMemcpyHostToDevice);
           CUDA_BFS_KERNEL1 <<<number_of_blocks, number_of_threads >>>(Vertex0, Edge0, Frontier0,
Visited0,d_done0);
        cudaMemcpy(&done0, d_done0 , sizeof(bool), cudaMemcpyDeviceToHost);
        cudaDeviceSynchronize();
    } while (!done0);

    printf("\n\n");

     printf("node key starting from node 8:\n8 = 0, 7 = 1, 5 = 2, 6 = 3, 4 = 4, 0 = 5, 1 = 6, 2 =
7, 3 = 8 \n");
    printf("Order of Customers to Visit if Starting Tour at Node 8:\n");

    do {
        int number_of_blocks = 1;
        int number_of_threads = number_of_nodes;

        count++;
        done2 = true;
        cudaMemcpy(d_done2, &done2, sizeof(bool), cudaMemcpyHostToDevice);
           CUDA_BFS_KERNEL2 <<<number_of_blocks, number_of_threads >>>(Vertex2, Edge2, Frontier2,
Visited2,d_done2);
        cudaMemcpy(&done2, d_done2 , sizeof(bool), cudaMemcpyDeviceToHost);
        cudaDeviceSynchronize();
    } while (!done2);

    //printf("\n\n");

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    printf("\n");

    printf("The number of groups/ layers in the network is : %d \n", count);

    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    cudaFree(Vertex0);
    cudaFree(Edge0);
    cudaFree(Frontier0);
    cudaFree(Visited0);
    cudaFree(Vertex2);
    cudaFree(Edge2);
    cudaFree(Frontier2);
    cudaFree(Visited2);
};
```