Efficient Route Optimization for Capacitated Vehicle Routing Problem

Junyi Pan, Yinuo Xu

May 15, 2024

Abstract :

This study addresses the Capacitated Vehicle Routing Problem (CVRP). The problem presents a significant challenge in transportation and delivery management. Its primary objective is to optimize the routes of vehicles to efficiently deliver goods to a predetermined set of customers while minimizing overall transportation costs. The complexity of the CVRP arises from the exponential growth of possible routes with the increasing number of customers and vehicles. In our project, we specifically focus on the Capacitated VRP, where vehicles are constrained by limited carrying capacities and are tasked with pickups or deliveries at multiple locations. This constraint adds an additional layer of complexity to the optimization problem, as it requires careful consideration of vehicle capacities to ensure that all customer demands are met while maintaining operational feasibility.

Index Terms: Capacitated Vehicle Routing Problem (CVRP), Genetic Algorithm, OpenMP, OSRM.

1 Sequential Solution

The sequential version of the Capacitated Vehicle Routing Problem (CVRP) represents a foundational aspect of developing algorithms to address complex VRP challenges, incorporating various components. We developed Genetic Algorithms (GA), Population management, Splitting, and Local Search techniques, to efficiently generate and refine potential solutions.

Genetic Algorithms We firstly employed the Genetic Algorithms that simulate the process of natural selection as a cornerstone in the sequential approach to CVRP. GA efficiently explores the solution space by selecting and crossing over parent solutions to generate offspring. This mechanism allows for the exploration of diverse solution possibilities, enabling the identification of promising potential solutions.

Population Population plays a pivotal role in organizing and refining the potential solutions within the GA framework. Through the creation of a population of individuals, each representing a potential solution, the GA algorithm can evaluate and update solutions iteratively. The population class facilitates the initialization, addition, and evaluation of individuals based on their fitness and penalty scores. By biasing the selection towards feasible solutions, the population component contributes to identify the best feasible solutions.

Split Furthermore, Splitting represents a critical step in partitioning routes into subroutes while adhering to their constraints. Split function divides the routes of an individual solution into multiple sub-routes, each designed to adhere to capacity and duration constraints. In addition, Bellman's algorithm is utilized in a topological manner to partition routes without considering a limited fleet. This algorithm iteratively computes the optimal cost of each sub-route, factoring in capacity and duration constraints. This approach ensures the efficient allocation of customers to vehicles while maintaining operational feasibility.

Moreover, the split function accounts for scenarios with a limited fleet, incorporating a similar approach. This variant optimizes the partitioning process for situations where the number of available vehicles is restricted. By carefully allocating customers to sub-routes, this component helps in efficiently utilizing the available fleet resources while meeting operational constraints.

Local Search Local Search techniques, 2-Opt or 3-Opt, complement the sequential approach by refining solutions through iterative optimization. These techniques explore the neighborhood of existing solutions, identifying and implementing route segment swaps to improve solution quality. By iteratively refining solutions, Local Search enhances the convergence towards optimal or near-optimal routing plans, thereby improving overall performance.

The development and refinement of the sequential approach to CVRP are integral to advancing the field of logistics optimization. By enhancing the efficiency and effectiveness of CVRP-solving methodologies, we could put our effort on addressing real-world challenges more effectively. The rest of essay will mainly discuss the potential improvement, which may includes the integration of advanced optimization techniques, such as ORSM and Parallelism , to further improve CVRP-solving performance.

2 Methods

Our approach includes an OSRM engine solution and a hybrid solution combining Genetic Algorithms (GA) and local search techniques to address the Capacitated Vehicle Routing Problem (CVRP). Genetic algorithms, inspired by the principles of natural selection, offer a heuristic optimization method for evolving solutions to complex optimization problems. Initially, a pool of randomly generated genes representing routes is created. The algorithm then selects parents from this gene pool, employing crossover and local search operations to generate offspring solutions. These offspring are subsequently added to the pool, replacing less fit individuals. Through iterative refinement over successive generations, the algorithm progressively enhances the overall solution, aiming to minimize transportation costs while satisfying vehicle capacity constraints.

2.1 OSRM Engine

OSRM (open-source routing machine) is a high-performance routing engine designed to compute the fastest paths between locations. We decided to integrate this engine into our sequential implementation because it is implemented in C++, which is highly compatible with our sequential C++ code and is scalable for future optimization. OSRM provides an API that takes advantage of to its powerful routing algorithms, allowing us to seamlessly incorporate routing functionality into our application. By utiliz-

ing the OSRM API, we can efficiently request routing information, such as shortest paths, travel times, and route geometries, for various transportation modes like driving, walking, or cycling.

Upon sequential implementation, we replace the existing method of finding the optimal route, which is splitting and local search functions we discussed earlier, with the OSRM engine. cURL commands are used for sending requests to the server. The inputs are the coordinates of waypoints and optional parameters such as overview geometry. The API will return an array of Route objects consisting of the duration and distance for each route, ordered by descending recommendation rank.

```
# Query on Berlin with three coordinates and no overview geometry returned:
curl 'http://router.project-osrm.org/route/v1/driving/13.388860,52.517037;13.
397634,52.529407;13.428555,52.523219?overview=false'
```

Figure 1. an example cURL command

After the first stage of sequential when the population is generated, the coordinates of the clients are collected and fed to the OSRM API. The API will return the list of optimal routes. This streamlined interaction with the API allows us to effortlessly incorporate optimal routing functionality, enhancing its capabilities without the need for extensive development efforts. An inherent advantage of OSRM lies in its utilization of OpenStreetMap data, ensuring that routing computations are based on real-world map information. This reliance on OpenStreetMap empowers users to select any desired area for routing analysis, leveraging the richness and accuracy of OpenStreetMap's data. Additionally, OSRM's flexibility extends to its deployment options, as users can set up their own routing engine by utilizing the pre-configured Docker image provided by OSRM. This enables local requests, facilitating efficient testing and development processes on localhost environments. Overall, the integration of OSRM not only enhances routing capabilities but also offers flexibility and ease of deployment, making it a valuable asset for our application.

2.1.1 Test and Results In this project scope, the genetic sequential solution consistently demonstrates faster execution times on average compared to utilizing the OSRM API. On average, the execution time of the OSRM API is approximately 1.33 times that of the sequential approach. However, as illustrated by the two line graphs depicting the impact of increasing the number of clients, a notable trend emerges. While both approaches experience an increase in execution time with a growing number of clients, the slope of the line representing the OSRM API approach is notably flatter on the right side. This flatter slope indicates that the execution time grows at a slower rate as the number of clients increases when utilizing the OSRM API.

The bar chart below further illustrates the comparison between the execution times of the two approaches. The blue bars represent the sequential execution, while the red bars depict the utilization of the OSRM engine. Consistently across various scenarios and client loads, the genetic sequential solution outperforms the OSRM API in terms of execution time. However, the analysis also highlights the scalability advantage of the OSRM API approach, particularly evident in the slower growth rate of execution time as the number of clients increases.

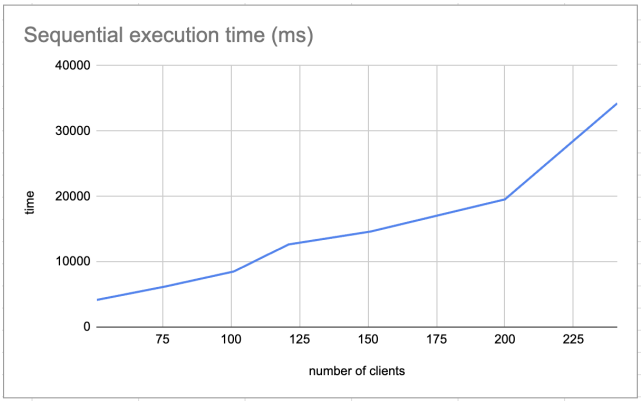

Figure 2. This graph shows the sequential execution time vs number of clients

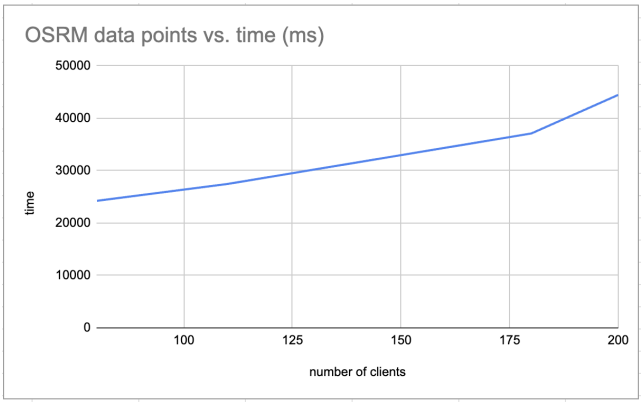

Figure 3. This graph shows the OSRM engine execution time vs number of clients

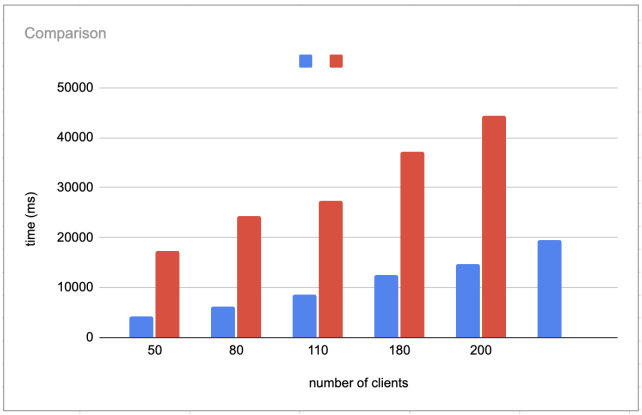

Figure 4. This graph shows the comparison of execution time between sequential and OSRM engine

2.2 OpenMP

Applying OpenMP to parallel code is a powerful technique for enhancing performance by leveraging the computational resources of multi-core processors. OpenMP directives, such as "parallel," enable the specification of code regions to be executed concurrently by multiple threads. This concurrency allows for efficient utilization of available CPU cores, leading to faster execution times and improved throughput. Moreover, synchronization constructs like "barrier" facilitate coordination among threads, ensuring proper sequencing of operations and preventing data inconsistencies. Directives such as "critical" and "atomic" provide mechanisms for managing shared data access, mitigating potential issues related to concurrent access and ensuring the integrity of program execution. By harnessing the capabilities of OpenMP, we could unlock the parallel processing potential of modern hardware architectures, thereby optimizing the performance of their applications across various domains, including computational algorithms, scientific simulations, and data processing tasks. Thus, paralleling the CVRP-solving algorithm using OpenMP offers significant potential for improving performance by harnessing the power of parallel processing. By paralleling key phases of the algorithm, including genetic operations, population management, and route splitting, the algorithm can explore solution space more efficiently and converge towards optimal or near-optimal solutions faster.

GA Parallelizing Genetic Algorithms (GAs) for solving the Capacitated Vehicle Routing Problem (CVRP) offers significant potential for enhancing performance by exploiting parallel processing capabilities. In parallel GA implementations, different pairs of parent solutions can be handled simultaneously, facilitating the concurrent generation of offspring solutions. This parallel processing approach accelerates the exploration of the solution space, leading to faster convergence towards optimal or near-optimal solutions.

Population Similarly, parallelization can be applied to the population phase by generating feasible individuals based on their fitness and penalty scores. The algorithm can efficiently evaluate and update a larger pool of potential solutions. This parallel population generation process enhances the diversity of solutions explored and contributes to the overall effectiveness of the Population algorithm.

Splitting Furthermore, parallelization can be leveraged during the splitting phase of the CVRP solution process. By paralleling the splitting of routes into sub-routes, the algorithm can simultaneously partition multiple routes while ensuring adherence to capacity and duration constraints. This parallel splitting approach optimizes the utilization of computational resources and accelerates the generation of feasible routing plans. In a parallelized implementation, multi-threading is utilized to validate constraints and capacities across multiple threads simultaneously. By distributing the workload across multiple threads, the algorithm can efficiently validate the feasibility of solution candidates in parallel. This parallel constraint validation process improves the scalability and efficiency of the CVRP-solving algorithm, enabling it to handle larger problem instances with greater computational efficiency.

2.2.1 Test and Result

We aimed to evaluate the performance of both sequential and parallel versions of the algorithm under varying time limits. By extending the time limit, the algorithms were allowed more time to explore potential solutions, potentially resulting in better cost outcomes. Despite the increased exploration time, the execution time remained below the specified time limit, indicating the efficiency of both sequential and parallel implementations.

Comparing the performance of the sequential and parallel versions, it was observed that both execution consistently achieved better cost outcomes within increasing time limits. This improvement in cost was achieved alongside reduced execution times, highlighting the effectiveness of both in leveraging processors to accelerate computation.

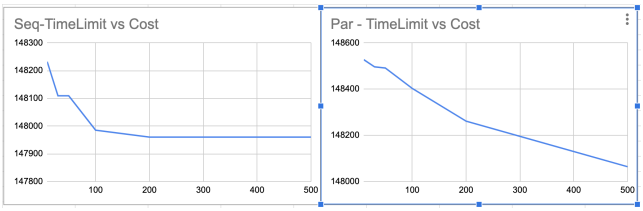

Figure 5. This graph shows the cost within increasing time limit

However, comparing the execution times between sequential and parallel implementations revealed notable differences. The parallel version consistently demonstrated shorter execution times across all tested time limits compared to its sequential counterpart. This efficiency gain can be attributed to the concurrent execution of tasks by multiple threads in the parallel version, leveraging the processing power of multi-core architectures effectively.

Moreover, the parallel implementation maintained its efficiency advantage over the sequential version regardless of the time limit. Even as the time limit increased, the parallel algorithm consistently outperformed the sequential one in terms of execution time, highlighting the scalability and performance benefits of parallel computing.

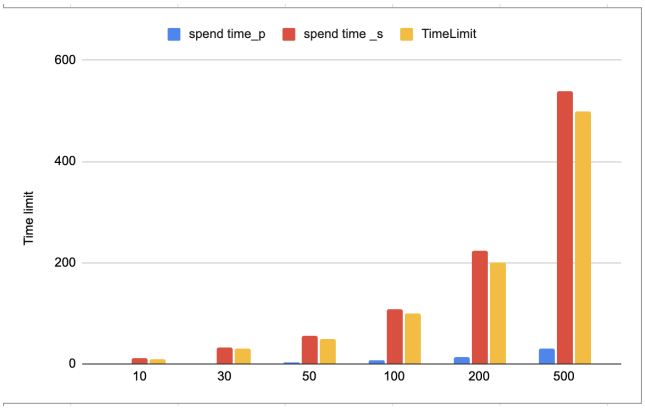

Figure 6. This graph shows the execution time of Sequential and Parallel version under different time limit

Overall, our test results indicate that both sequential and parallel versions of the algorithm were able to produce satisfactory solutions within the given constraints. However, the parallel implementation exhibited superior efficiency, offering faster execution times and better scalability, particularly as the time limit was extended. This underscores the effectiveness of parallel computing techniques in optimizing algorithm performance and achieving computational goals efficiently.

3 Future Work

We discussed multiple strategies for further exploration and optimization. One such direction involves experimenting with alternative optimization algorithms, such as simulated annealing, ant colony optimization, and particle swarm optimization. By substituting genetic algorithms with these alternatives and integrating them with local search strategies, we can explore different combinations to potentially enhance our solution's efficiency and effectiveness.

One limitation we encountered when using the OSRM API in this project is the disparity between the data inputs required by the genetic algorithm (VRP files) and those accepted by OSRM (OSM data). To address this, leveraging the libosrm cpp library would be a possible future step to take. This library offers bindings for C++ applications, enabling seamless interaction with the OSRM backend server. Furthermore, we can explore optimization techniques such as Intel's Threading Building Blocks (TBB) and OpenMP to optimize the library code, enhancing its performance and scalability.

Additionally, we can further refine the genetic algorithm by exploring additional optimization techniques, such as integrating TBB to parallelize computations. By leveraging parallel processing capabilities, we can potentially expedite the execution of genetic algorithms, improving their efficiency and scalability for larger problem instances. These steps collectively aim to refine our solution, enhancing its performance, scalability, and applicability to real-world routing challenges.

Acknowledgements

OSRM API documentation. (n.d.). https://project-osrm.org/docs/v5.24.0/api/route-service Vidalt. (n.d.). HGS-CVRP/readme.md at Main · Vidalt/HGS-CVRP. GitHub. https://github.com/vidalt/HGS-CVRP/blob/main/README.md The vehicle routing problem | Society for Industrial and Applied Mathematics. (n.d.-b). https://epubs.siam.org/doi/book/10.1137/1.9780898718515 Docker. (n.d.). https://hub.docker.com/r/osrm/osrm-backend Göbel, N. (n.d.). The vehicle routing problem explained. Home. https://conundra.eu/blog/the-vehicle-routing-problem-explained A genetic algorithm for the vehicle routing problem. (n.d.-a). https://trace.tennessee.edu/cgi/viewcontent.cgi?article=6358context=utk$_{g}radthes$