Major Project Report

on

# Parallel Algorithms for Expediting Vehicle Routes

Submitted by

**Ashith Shetty 21BCS019**

**Smruti Milind Patil 21BCS082**

**Pratik Prakash Pakhale 21BCS085**

**Shreyansh Tiwari 21BCS114**

Under the guidance of

**Dr. Pramod Yelmewad**

**Assistant Professor, CSE**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD

17/11/2024

# *Certificate*

This is to certify that the project, entitled **Parallel Algorithms for Expediting Vehicle Routes**, is a bonafide record of the Major Project coursework presented by the students whose names are given below during 2024-2025 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

| Roll No | Names of Students |
|---|---|
| 21BCS019 | Ashith Shetty |
| 21BCS082 | Smruti Patil |
| 21BCS085 | Pratik Pakhale |
| 21BCS114 | Shreyansh Tiwari |

Dr. Pramod Yelmewad (Project Supervisor)

# Contents

# List of Figures

## 1 Introduction

The Capacitated Vehicle Routing Problem (CVRP) is a classical and extensively studied problem in the field of combinatorial optimization, operations research, and logistics. It is a vehicle routing problem in which a fleet of identical vehicles of finite capacity and located at a central depot must service a set of customers with known demands. The time taken to solve these instances is excessively long for a desired accuracy. To tackle large-scale or time-constrained instances, heuristic and metaheuristic algorithms have gained prominence. Among the most notable heuristic methods are the Clarke-Wright Savings Algorithm, known for its simplicity and efficiency, and Ant Colony Optimization (ACO), which simulates the foraging behavior of ants to iteratively construct promising routes. Metaheuristics such as Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO) provide flexible frameworks for exploring complex solution spaces. These techniques do not guarantee optimality but offer high-quality solutions in reasonable computational time, making them suitable for practical applications.

CVRP has become a cornerstone in the optimization strategies of many industries. Logistics and courier companies like FedEx and UPS leverage CVRP-based solutions to optimize package deliveries. Waste collection, school bus scheduling, and field service management also benefit significantly from these routing strategies. In healthcare, CVRP helps optimize the delivery of medical supplies and vaccines, especially under capacity and time-sensitive constraints. Moreover, manufacturers and retailers use CVRP to manage inbound and outbound logistics efficiently, ensuring timely and cost-effective distribution of goods.

As supply chains become more dynamic and customer expectations continue to rise, the importance of solving CVRP effectively is expected to grow. Advances in computational power, machine learning, and real-time data integration are driving the development of adaptive and scalable CVRP solvers.

# 2 Literature Review

The Capacitated Vehicle Routing Problem (CVRP) is a foundational problem in combinatorial optimization and logistics, where the goal is to determine a set of routes to serve a set of customers with capacity-constrained vehicles while minimizing total cost. Over the years, solution strategies have evolved from exact methods to increasingly sophisticated heuristics and hybrid metaheuristics, driven by the need to handle large, real-world instances efficiently.

## 2.1 Classical Metaheuristics and Genetic Approaches

One of the most enduring and impactful methods in CVRP literature is the *Hybrid Genetic Search* (HGS) by Vidal et al., later refined into an open-source C++ implementation known as HGS-CVRP. This algorithm combines permutation-based genetic crossover with powerful local search neighborhoods. A notable enhancement, the *SWAP** neighborhood, allows customer exchanges across routes without fixed positions, leading to richer local optima. HGS-CVRP maintains a balance between conceptual simplicity and high performance, consistently achieving state-of-the-art results on benchmark instances.

Building on this, MAESN (Memetic Algorithm with Effective Search Neighbors) extends the search space by introducing six additional sequence-based operators. While computationally more demanding, it significantly improves solution quality, particularly on complex CVRP benchmarks such as those from the 12th DIMACS Implementation Challenge. These enhancements reflect a broader trend: hybridization of genetic algorithms with local search to achieve both exploration and exploitation in solution spaces.

## 2.2 Hybrid and Large-Scale Optimization Frameworks

Several algorithms focus on scaling to large CVRP instances by adopting modular and hybrid designs. POP-HGS integrates HGS within the POPMUSIC framework, re-optimizing subsets of the solution space iteratively. This allows scalable refinement without compromising global quality. Likewise, FHCSolver employs a dynamic control strategy that switches between HGS, FILO (Fast Iterated Local Optimization), and their hybrid variants based on instance features, achieving first place in the CVRP track of the 12th DIMACS Challenge.

FSP4D (FILO with Set Partitioning for DIMACS) merges rapid local search with a global optimization layer via set partitioning, delivering high-quality solutions especially for large-scale instances. However, it relies on commercial solvers like Gurobi or CPLEX, which may limit accessibility for some researchers.

UFF-IC (Unified Fast and Flexible Iterated Cooperative framework) offers a modular, general-purpose metaheuristic that emphasizes cooperation between components. It alternates between diversification and intensification phases to maintain diversity and robustness. Its cooperative search strategy proves effective on large and diverse instances but can suffer from memory overhead and sensitivity to parameter tuning.

## 2.3 GPU-Accelerated and Parallel Methods

To address the bottleneck of runtime in large-scale CVRP instances, recent research has turned to GPU acceleration. Abdelatti and Sodhi propose a fully GPU-implemented genetic algorithm using CUDA, enabling parallel evolution and evaluation of thousands of solutions. Their approach achieves up to 450× speedup over CPU versions, with minimal compromise on solution quality. Nonetheless, the reliance on CUDA-specific hardware and fixed hyperparameters limits broader applicability.

Similarly, Yelmewad and Talawar present a GPU-parallel local search heuristic using customer- and route-level thread allocation. Their method achieves runtime reductions exceeding 100×, solving instances with up to 30,000 customers. However, the focus remains primarily on computational efficiency, with solution quality comparable to classical heuristics rather than surpassing them.

## 2.4 Domain-Specific, Explainable, and Multi-Objective Approaches

As real-world routing problems increasingly demand transparency and adaptability, new methods have emerged to address these needs. AXIOM exemplifies this trend by integrating symbolic AI planning with preference learning. Unlike traditional black-box metaheuristics, AXIOM produces interpretable, multi-objective delivery plans using logic-based transformations. While promising for decision-support systems in logistics, AXIOM's reliance on symbolic planning tools like Fast Downward and its sensitivity to data quality raise concerns for scalability and robustness.

HLNS (Hybrid Large Neighborhood Search), on the other hand, simplifies adaptive strategies to improve performance on classical VRP and VRPTW instances. Though efficient for smaller problems, HLNS lacks sparsification or decomposition techniques, making it less effective for large-scale CVRP scenarios.

## 2.5 Comparative Benchmarks and the DIMACS Challenge

The 12th DIMACS Implementation Challenge for CVRP has served as a critical benchmark for evaluating modern algorithms. It revealed that hybrid methods like FHCSolver, POP-HGS, and MAESN consistently outperform classical heuristics such as LKH-3 and SISR in both speed and solution quality. Notably, HGS-CVRP and its variants remain competitive baselines, often serving as subsolvers or foundational layers in more complex algorithms.

## 2.6 Research Gaps and Future Directions

While substantial progress has been made, several gaps remain. Many high-performing algorithms require careful parameter tuning or rely on hardware-dependent acceleration, which can limit their accessibility or robustness. Additionally, explainability remains underexplored, despite growing demand for transparent decision-making in logistics. There is also a need for unified frameworks that can balance solution quality, computational speed, and adaptability across diverse CVRP variants.

In this context, the ongoing refinement of hybrid genetic algorithms—particularly those like HGS-CVRP—offers a solid foundation for future research. Their flexibility, open-source availability, and strong performance across benchmark datasets make them ideal candidates for further extension, whether through integration with learning models, GPU acceleration, or explainable components.

# 3 Data and Methods

## 3.1 Problem Statement

## 3.2 ParMDS Algorithm

The ParMDS (Parallel Minimum-Distance Search) algorithm is a scalable and efficient metaheuristic designed to solve large instances of the Capacitated Vehicle Routing Problem (CVRP). It starts by constructing a Minimum Spanning Tree (MST) over the complete graph of customers and depot using Prim's algorithm. This MST forms the structural backbone of the solution space and significantly reduces the combinatorial complexity of routing permutations. Once the MST is formed, the algorithm performs a randomized Depth-First Search (DFS) traversal starting from the depot. This random traversal produces a permutation of customer nodes that respects the MST topology while introducing solution diversity through randomization.

Following the DFS, the algorithm partitions the permutation into multiple feasible routes by adhering to the vehicle capacity constraint. Each constructed route is then refined using Traveling Salesman Problem (TSP) heuristics such as the 2-Opt algorithm and the Nearest Neighbor approach. These refinements improve individual route quality by eliminating redundant travel and minimizing total route cost. The entire process is embedded within a parallel framework using OpenMP, where multiple DFS permutations are evaluated in parallel to identify the best solution. This framework makes the ParMDS algorithm particularly suitable for large-scale CVRP instances.

**Algorithm 1** ParMDS Algorithm

```
1: Input: Set of customer nodes and depot, vehicle capacity, demand at each node
2: Compute the Minimum Spanning Tree (MST) using Prim's algorithm
3: for each iteration $\rho$ in parallel (using OpenMP) do
4:     Randomly shuffle the adjacency list of the MST
5:     Perform a randomized DFS traversal from the depot to create a permutation of nodes
6:     Partition the node permutation into feasible vehicle routes based on capacity
7:     Apply TSP heuristics (2-Opt and Nearest Neighbor) to refine each route
8:     Update the best solution if current iteration produces better result
9: end for
10: Output: Best solution found after all iterations
```

To enhance both the quality and efficiency of the solution, the ParMDS algorithm incorporates several optimization strategies. First, it introduces a randomization mechanism by shuffling the adjacency lists of the MST before each DFS traversal. This ensures the generation of diverse route permutations, allowing broader exploration of the solution space without significant computational overhead. Additionally, the algorithm reduces the solution space drastically by relying on the MST structure, which avoids evaluating all possible node sequences and instead confines the traversal to a meaningful subset of paths.

The algorithm is also highly parallelizable. It leverages OpenMP to distribute the permutation search process across multiple threads in a "strided" fashion, which minimizes memory contention and false sharing during parallel execution. Furthermore, once candidate routes are constructed, they are locally optimized using 2-Opt (which iteratively swaps route segments to eliminate crossing paths) and the Nearest Neighbor heuristic (which constructs a quick approximation of the shortest route). This hybrid approach enables ParMDS to find high-quality solutions with impressive computational efficiency. The overall time complexity is approximately $O(n^2 \log n + n^2 + n)$, and empirical evaluations have shown that it achieves an average gap of 11.85% from the best-known solutions while running up to 1189 times faster than competing GPU-based methods.

## 3.3 Extension to the parMDS Algorithm

Despite its impressive computational efficiency and scalability, the ParMDS algorithm suffers from a significant limitation: the lack of inter-route optimization. Once customers are partitioned into specific routes during the *Convert_To_Routes* phase, they are permanently assigned to those routes. This architectural design implies that there is no mechanism to reassign customers across different routes, even when such reassignment could substantially reduce the total travel cost. The algorithm is thus confined to optimizing individual routes in isolation, focusing exclusively on intra-route improvements through techniques like the 2-Opt and Nearest Neighbor heuristics. As a result, the quality of the final solution is tightly bound to the initial DFS-based customer ordering and partitioning, which becomes a rigid structure that the algorithm cannot escape.

From an architectural perspective, the MST-based traversal provides an efficient means to reduce the solution search space, but it also imposes structural constraints that may exclude globally optimal customer groupings. The algorithm's randomization strategy—while useful in producing multiple candidate solutions—only alters the ordering of neighbors in the adjacency lists of the MST. It does not fundamentally reconsider how customers should be grouped into different routes. Consequently, the algorithm's refinement step remains constrained to making improvements within already-established routes, without any scope for inter-route adjustments or swaps.

This design choice results in a noticeable trade-off between speed and solution quality. Although ParMDS is reported to be 36 to 1189 times faster than state-of-the-art GPU-based implementations, it consistently delivers solutions that are on average 11.85% worse than the best-known results. This performance gap underscores the algorithm's inability to perform global optimization. In contrast, other methods like Hybrid Genetic Search for CVRP (HGS-CVRP) offer superior solution quality by incorporating inter-route operations and guided local search techniques. Even some classical heuristics, despite being computationally more intensive, can outperform ParMDS in terms of route optimization because they allow customer reassignments between routes.

## 3.4 Improvements to parMDS Algorithm

### 3.4.1 Enhanced MST Approaches: Implementation Details and Results

To explore avenues for improving solution quality in the ParMDS algorithm, we introduced two primary modifications centered around the construction of the Minimum Spanning Tree (MST). The first involved enhancing the basic MST construction step. Through systematic experimentation, we evaluated whether adopting alternative MST algorithms or modifying parameters could yield improved results. However, we observed that this phase was already highly efficient and had limited influence on overall algorithm performance. As such, further optimization of the MST construction process was deemed unnecessary.

The second and more impactful modification was the implementation of a multi-MST approach. Instead of relying on a single MST followed by randomized DFS traversals, we developed a parallel strategy that simultaneously generated $k$ distinct MSTs, each forming an alternative foundation for route planning. This diversification was achieved by varying the starting node for MST construction, perturbing edge weights with small random values to encourage structural differences, and employing randomized tie-breaking during edge selection. These distinct MSTs were then processed in parallel through the entire ParMDS pipeline, which included DFS traversal, route construction, and local refinement using heuristics.

Despite the theoretical appeal of this multi-MST approach, our experimental results were unexpectedly negative. There was no observable improvement in solution quality or total routing cost compared to the original single-MST approach. In fact, the best-performing solutions consistently stemmed from the original MST configuration. Furthermore, parallel processing of multiple MSTs led to resource contention, where computational threads allocated to alternative MSTs detracted from the more effective random DFS explorations on the primary MST. This inefficient allocation of computational effort resulted in increased runtime without any measurable gain in optimization.

These findings underscore an important structural insight into the CVRP problem and the ParMDS algorithm: the primary source of solution diversity and quality lies in the randomized DFS traversal rather than in altering the MST itself. As such, concentrating computational resources on generating a larger number of DFS permutations from a high-quality single MST proves more fruitful than dispersing efforts across multiple MSTs with fewer traversals each. Our experiment highlights the importance of empirical validation in algorithm development, as

**Algorithm 2** MST Enhancements

```
1: Input: MST T from graph G = (V, E), vehicle capacity Q, customer demands D, number of iterations rho
2: Output: Best CVRP solution R* with cost CR
3: for each iteration i from 1 to rho in parallel do    ▷ NEW: Fully parallelized superloop using OpenMP
4:     T_i ← Shuffle_Adjacency_List(T, seed = i)    ▷ NEW: Randomized DFS via shuffling adjacency lists
5:     pi_i ← DFS_Traversal(T_i, starting from depot)
6:     R_i ← Split_Tour_By_Capacity(pi_i, Q, D)
7:     R_A ← TSP_Approximation(R_i)    ▷ NEW: Heuristic TSP-style postprocessing
8:     R_B ← TwoOpt_Improvement(R_i)    ▷ OLD: 2-opt already in original
9:     R_i* ← Select_Better_Routes(R_A, R_B)    ▷ NEW: Layered dual-route comparison
10:    CR_i ← Evaluate_Cost(R_i*)
11:    if CR_i ¡ CR then
12:        CR ← CR_i
13:        R* ← R_i*
14:    end if
15: end for
16: return R*, CR
```

enhancements that are intuitively promising may ultimately be counterproductive in practical settings.

### 3.4.2 Advanced Inter-Route Refinement for CVRP: Implementation, Results, and Analysis

To address the lack of inter-route optimization in the original ParMDS algorithm, we implemented a comprehensive strategy that allows customer reassignment between routes. While ParMDS was effective in optimizing individual routes, it restricted customers to their initially assigned routes, leaving room for improvement through inter-route adjustments. Our refinement strategy focused on two primary operations: relocation and swap moves.

**Relocation moves** involved systematically evaluating the cost impact of removing a customer from its current route and inserting it into a different route. For each customer originally in route $R_1$, we calculated the cost saved by removing it, and for every feasible insertion into route $R_2$, we computed the new cost and the net benefit. If the resulting total cost decreased beyond a minimal threshold (`-1e-6`), the move was executed.

**Swap moves** considered exchanging customers between two distinct routes. For each candidate pair of customers $i \in R_1$ and $j \in R_2$, we checked feasibility and computed the combined cost effect of swapping them. If a net cost reduction was achieved, the swap was performed.

We parallelized the evaluation of these operations to improve efficiency. Multiple threads concurrently evaluated relocation and swap options, though the approach was brute-force in nature and lacked sophisticated filtering mechanisms.

**Expected improvements** included significant reductions in travel costs and enhanced overall route quality. We believed the flexibility introduced by inter-route reassignment would allow for competitive results, even when compared to state-of-the-art solvers.

**Observed outcomes**, however, were mixed. On smaller CVRP instances (hundreds of customers), we observed modest improvements in route quality, but execution times increased significantly. On large instances (thousands of customers), such as Brussels2 with over 15,000 customers, the algorithm failed to complete due to excessive runtime. The lack of scalability stemmed from several interrelated issues:

**Computational Complexity:** The algorithm's complexity grew quadratically with the

**Algorithm 3** relocationMove()

```
Require: Collection of routes $R$, Distance matrix $D$
Ensure: Refined routes with reduced cost
 1: for each pair of routes $(R_1, R_2)$ in $R$ do                        ▷ Parallel execution
 2:     for each customer $i \in R_1$ do                                   ▷ Parallel execution
 3:         cost_remove ← cost saving from removing $i$ from $R_1$
 4:         residual_capacity_R1 ← $R_1$.demand - demand($i$)
 5:         if residual_capacity_R1 $\geq 0$ then
 6:             for each position $p$ in $R_2$ do
 7:                 residual_capacity_R2 ← $R_2$.demand + demand($i$)
 8:                 if residual_capacity_R2 $\leq$ Capacity then
 9:                     cost_insert ← cost of inserting $i$ at position $p$ in $R_2$
10:                     $\Delta$ cost ← cost_remove - cost_insert
11:                     if $\Delta$ cost $< -1e - 6$ then                  ▷ Move is beneficial
12:                         Move customer $i$ from $R_1$ to position $p$ in $R_2$
13:                         Update $R_1$.demand and $R_2$.demand
14:                         break                 ▷ Optional: accept first improving move
15:                     end if
16:                 end if
17:             end for
18:         end if
19:     end for
20: end for
21: return $R$
```

**Algorithm 4** swapMove()

```
Require: Collection of routes $R$, Distance matrix $D$
Ensure: Refined routes with reduced cost
 1: for each pair of routes $(R_1, R_2)$ in $R$ do                      ▷ Parallel execution
 2:     for each customer $i \in R_1$ do                                 ▷ Parallel execution
 3:         for each customer $j \in R_2$ do
 4:             new_demand_R1 ← $R_1$.demand - demand($i$) + demand($j$)
 5:             new_demand_R2 ← $R_2$.demand - demand($j$) + demand($i$)
 6:             if new_demand_R1 ≤ Capacity and new_demand_R2 ≤ Capacity then
 7:                 cost_remove1 ← cost saving from removing $i$ from $R_1$
 8:                 cost_insert1 ← cost of inserting $j$ at $i$'s position in $R_1$
 9:                 cost_remove2 ← cost saving from removing $j$ from $R_2$
10:                 cost_insert2 ← cost of inserting $i$ at $j$'s position in $R_2$
11:                 $\Delta$ cost ← (cost_remove1 - cost_insert1) + (cost_remove2 - cost_insert2)
12:                 if $\Delta$ cost $< -1e - 6$ then                        ▷ Swap is beneficial
13:                     Swap customers $i$ and $j$ between routes $R_1$ and $R_2$
14:                     Update $R_1$.demand and $R_2$.demand
15:                     break                        ▷ Optional: accept first improving move
16:                 end if
17:             end if
18:         end for
19:     end for
20: end for
21: return $R$
```

number of customers. Each potential move required costly distance and cost calculations. For large instances, the number of moves became intractable, leading to billions of operations.

**Parallel Processing Issues:** Despite using multithreading, we encountered serious bottlenecks due to thread contention, cache coherence problems, lock contention, and false sharing. Threads often accessed or modified shared route structures, resulting in degraded performance due to synchronization and memory interference.

**Memory Inefficiencies:** The use of non-local and random memory access patterns led to frequent cache misses and poor utilization of memory bandwidth. With multiple threads in play, these issues compounded, converting the problem from CPU-bound to memory-bound.

**Algorithmic Limitations:** The brute-force nature of the implementation meant all possible moves were evaluated without any heuristic guidance. The absence of filtering techniques, delta evaluation, spatial prioritization, or more advanced acceptance strategies (e.g., simulated annealing or tabu search) meant that many computational resources were wasted on unpromising candidate moves.

While the inter-route refinement conceptually enhances solution flexibility and quality, our naive implementation proved inefficient for large-scale CVRP instances. The results highlight the importance of incorporating tailored optimization strategies, such as guided local search and move filtering, when designing scalable algorithms. Our experience underscores that even well-motivated algorithmic enhancements can fall short if not supported by performance-aware design choices.

### 3.4.3 CUDA-Based Randomized Solutions for Intelligent Swaps

Following the performance bottlenecks observed with the CPU-based exhaustive inter-route move evaluations, we transitioned to a GPU-based implementation using CUDA. This approach leverages the inherent parallelism of the GPU architecture to accelerate customer swaps between routes.

The implementation adopts a hierarchical parallelization strategy. At the higher level, each CUDA thread block is responsible for handling a specific pair of routes. This isolation naturally reduces data contention and simplifies synchronization. Within each block, individual threads compute potential swap gains concurrently, with each thread evaluating a specific pair of customers. Once these gains are computed, warp-level primitives such as `__shfl()`, `__ballot()`, and `__any()` are employed to aggregate results and identify the most promising swap candidates. These warp-synchronous operations avoid the overhead of global synchronization and exploit fast shared memory communication.

This GPU-driven design is expected to address many of the limitations we faced earlier. By parallelizing swap evaluations across multiple route pairs and avoiding global memory contention, the CUDA implementation should achieve significant speedup over the CPU-based brute-force version. Additionally, by embedding intelligence in move selection and focusing computational effort on promising candidates, the approach is positioned to improve overall route quality. While the precise numerical results were not documented at the time of writing, early observations suggested better scalability, particularly in instances involving thousands of customers where CPU-based methods struggled.

The CUDA-based swap mechanism successfully exploits GPU architecture to bypass the performance bottlenecks of brute-force CPU methods, offering a scalable solution for inter-route refinement. Its effectiveness hinges on efficient memory management and intelligent move selection, and it offers a promising direction for large-scale CVRP instances.

### 3.4.4 Optimized Route Splitting Using Dynamic Programming

In parallel with improvements to inter-route optimization, we also focused on enhancing the route construction phase. The original ParMDS algorithm employs a greedy route splitting mechanism: whenever the addition of the next customer would violate the vehicle capacity constraint, a new route is initiated. Although efficient, this greedy approach often leads to suboptimal route configurations, particularly in instances with tight capacity constraints or uneven customer distributions.

To address this, we developed a dynamic programming (DP) based route splitting algorithm. The method begins by computing a cost matrix $C[i:j]$ representing the cost of assigning customers $i$ to $j$ to a single route. This matrix respects vehicle capacity constraints and accounts for travel costs. Using this matrix, the algorithm then constructs a DP table where $dp[j]$ denotes the minimum cost to serve customers from 1 to $j$, and $split[j]$ stores the optimal partition point that resulted in this cost. The recurrence used is $dp[j] = \min_{i<j}(dp[i] + C[i:j])$, which is solved in $O(n^2)$ time. Once the table is filled, backtracking through the *split* array reconstructs the optimal sequence of route boundaries.

**Algorithm 5** DP_Route_Split ($\pi$, Q, D)

```
Require: A permutation $\pi$, Capacity Q, Demands D
Ensure: Routes, a set of optimally split routes
                                          ▷ Precompute route costs with prefix sums
 1: prefixDemand[0] $\leftarrow$ 0
 2: for $i \leftarrow 1$ to $n$ do
 3:     prefixDemand[i] $\leftarrow$ prefixDemand[i-1] + D[$\pi[i]$]
 4: end for
 5: for $i \leftarrow 1$ to $n$ do
 6:     for $j \leftarrow i$ to $n$ do
 7:         totalDemand $\leftarrow$ prefixDemand[j] - prefixDemand[i-1]
 8:         if totalDemand $\leq Q$ then                    ▷ Check feasibility
 9:             C[i][j] $\leftarrow$ CalculateRouteCost($\pi$, i, j)
10:         else
11:             C[i][j] $\leftarrow \infty$                  ▷ Infeasible route
12:         end if
13:     end for
14: end for
                                          ▷ Build DP table with optimal splits
15: dp[0] $\leftarrow$ 0
16: for $j \leftarrow 1$ to $n$ do
17:     dp[j] $\leftarrow \infty$
18:     for $i \leftarrow 0$ to $j - 1$ do
19:         if dp[i] + C[i+1][j] ¡ dp[j] then
20:             dp[j] $\leftarrow$ dp[i] + C[i+1][j]
21:             split[j] $\leftarrow i$
22:         end if
23:     end for
24: end for
                                          ▷ Backtrack to construct optimal routes
25: Routes $\leftarrow \emptyset$
26: $j \leftarrow n$
27: while $j > 0$ do
28:     $i \leftarrow$ split[j]
29:     OneRoute $\leftarrow \pi[i + 1...j]$
30:     Routes $\leftarrow$ Routes $\cup${OneRoute}
31:     $j \leftarrow i$
32: end while
33: return Routes
```

This DP-based approach guarantees globally optimal route partitions under capacity constraints and is particularly useful in generating well-balanced routes. The time complexity is quadratic, which makes it viable for mid-sized instances (up to 1000 customers) but less practical for very large datasets. Nonetheless, for datasets within this size range, the method provides tangible improvements over greedy splitting, especially in scenarios with tight resource limits or customer clusters that are poorly served by local decision rules.

The dynamic programming-based route splitter provides an exact solution to the subproblem of route segmentation. While its quadratic complexity limits its scalability, its ability to generate high-quality initial route sets can be particularly valuable when combined with subsequent local refinements.

### 3.4.5 Direction-Aware Tour Construction and Parallel Correctness

In an effort to enhance the ParMDS algorithm's performance and solution quality, we investigated two primary modifications: the introduction of a geometrically-guided heuristic for tour construction and the correction of a critical parallel execution flaw. The original algorithm (`parmds.cpp`) relied solely on randomized shuffling of Minimum Spanning Tree (MST) adjacency lists followed by Depth-First Search (DFS) to generate candidate tours, coupled with a parallel implementation that suffered from data races. Our modified version (`parmds_opt.cpp`) aimed to address these aspects.

1. **Parallel Execution Correction:** A fundamental flaw was identified in the parallel loop of the original implementation. Multiple OpenMP threads were concurrently modifying the shared `mstCopy` data structure during the neighbour list shuffling phase, leading to a data race and unpredictable behavior. This was rectified by ensuring each thread operates on a *private, thread-local copy* (`auto threadMstCopy = mstCopy;`) of the MST within the parallel loop. All subsequent modifications (shuffling or sorting) and traversals (DFS) within that thread's iteration block act only on this private copy, eliminating the data race and ensuring correct, independent parallel execution (See Listing 1). This fix is crucial for the algorithm's stability and the validity of its parallel exploration.

2. **Direction-Aware Search Heuristic:** As an alternative to pure randomization, we implemented an optional strategy to guide the DFS tour construction using geometric

information. This heuristic, controlled via a command-line flag (`-diraware`), involves:

- Calculating angles between nodes using `atan2`.
- Employing `DirectionalMSTSort` to sort MST neighbour lists based primarily on angular proximity to a systematically varied `referenceAngle` (0-360 degrees across parallel iterations), biasing the search towards specific directions.
- Optionally using `DirectionAwareShortCircutTour`, a modified DFS that dynamically attempts to maintain the current direction of travel by prioritizing neighbours aligned with the last traversed segment.

The intent was to leverage spatial locality to generate potentially "smoother" or more direct initial tours compared to random shuffling.

Listing 1: Conceptual Fix for Parallel Data Race

```
// Inside the '#pragma omp parallel for' loop:

// --- Original (parmds.cpp - Incorrect) ---
for (auto &list : mstCopy) { // Operates on SHARED mstCopy
    std::shuffle(list.begin(), list.end(), ...); // DATA RACE!
}
ShortCircutTour(mstCopy, ...); // Reads potentially corrupt mstCopy

// --- Optimized (parmds_opt.cpp - Correct) ---
auto threadMstCopy = mstCopy; // Create THREAD-LOCAL COPY
optional DirectionalMSTSort(threadMstCopy, ...) ...
or shuffling:
for (auto &list : threadMstCopy) { // Operates on PRIVATE threadMstCopy
    std::shuffle(list.begin(), list.end(), ...); // Safe
}
ShortCircutTour(threadMstCopy, ...); // Reads private, valid threadMstCopy
```

**Expected Improvements:** The parallel correctness fix was expected to yield a stable and correctly functioning parallel algorithm, enabling reliable scaling. The direction-aware heuristic was hypothesized to accelerate convergence towards high-quality solutions by generating more structured initial tours, potentially reducing the number of iterations required or finding superior solutions compared to pure randomization by exploiting geometric insights.

**Observed Outcomes and Analysis:** Contrary to expectations, the modifications implemented in `parmds_opt.cpp` resulted in significantly degraded performance compared to the original `parmds.cpp`. On average, the solution cost **increased by 3.50%**, and the execution time **increased by 179.23%** (nearly 3x slower).

Several factors contribute to this negative outcome:

- **Computational Overhead:** The direction-aware heuristic introduces substantial computational cost. Calculating angles (`atan2`), angular differences, and repeatedly sorting neighbour lists (either statically via `DirectionalMSTSort` or dynamically within `DirectionAwareShortCircutTour`) is significantly more expensive than the fast `std::shuffle` operation used in the original version. This overhead is the primary driver behind the drastic increase in runtime.

- **Heuristic Ineffectiveness for CVRP:** The core assumption that geometrically "smooth" or directional tours are beneficial appears flawed in the context of CVRP. The binding capacity constraint dominates route feasibility. A directionally efficient path might lead to poor vehicle utilization or force awkward splits by the greedy `convertToVrpRoutes` procedure, ultimately resulting in higher overall costs compared to partitions derived from "lucky" random permutations.

- **Reduced Exploration / Local Optima**: By imposing geometric structure, the direction-aware search explores a potentially less diverse set of candidate tours compared to the vast space covered by randomization. Furthermore, the resulting "smoother" tours might be less amenable to improvement by the subsequent 2-Opt refinement stage, which excels at fixing blatant edge crossings often present in random tours, potentially trapping the search in poorer local optima.

While the parallel correctness fix using thread-local copies is essential for a valid parallel implementation, its benefits were completely overshadowed by the negative impact of the direction-aware heuristic. The results strongly suggest that for this particular ParMDS framework, the strength lies in the massive volume of computationally cheap, randomized explorations rather than the attempted, more expensive, guided search. The geometric guidance proved counterproductive for finding high-quality, capacity-constrained routes.

# 4 Results and Discussions

# 5 Conclusion

# References