A Comprehensive Benchmark of Deep Reinforcement Learning and Traditional Methods for the Dynamic Vehicle Routing Problem

Ryan Schapiro
University of Cape Town
Cape Town, South Africa
schrya010@myuct.ac.za

Krupa Prag
University of Cape Town
Cape Town, South Africa
krupa.prag@myuct.ac.za

ABSTRACT

The Dynamic Vehicle Routing Problem presents significant challenges for modern logistics operations, requiring algorithms that adapt to real-time changes without compromising solution quality. This research presents a comprehensive empirical evaluation of traditional optimization methods and deep reinforcement learning approaches for solving the dynamic vehicle routing problem. This research implements Clarke-Wright Savings and Adaptive Large Neighbourhood Search as traditional methods, compared against a Soft Actor-Critic algorithm utilizing Graph Attention Networks. The experimental framework models customer arrivals as a Poisson process, with 70% of customers available at runtime and 30% revealed dynamically, on adapted Solomon benchmark instances. Results show traditional methods achieve optimal or near-optimal solutions in significantly shorter times, with Clarke-Wright within 8% and Adaptive Large Neighbourhood Search within 1% of optimality. The Soft Actor-Critic agent achieves results within 3-54% of optimality, highlighting challenges in state representation and spatial reasoning. All methods handle dynamic requests with 100% completion rates. This research provides empirical evidence contradicting assumed advantages of deep reinforcement learning for dynamic routing problems, as computational overhead results in worse performance than traditional heuristic methods. The results indicate significant improvements are needed in Soft Actor-Critic and continuous action space algorithms, particularly in spatial intelligence and problem representation, to be considered viable alternatives to heuristics.

1 INTRODUCTION

The Vehicle Routing Problem (VRP), an extension of the NP-hard Travelling Salesman Problem, is a key combinatorial optimisation problem in operations research and transportation logistics. First introduced by Dantzig and Ramser in 1959, the classical VRP aims to determine optimal routes for a fleet of vehicles serving customers from a central depot while minimising route costs [6]. However, real-world logistic scenarios are often dynamic, as factors such as traffic, customer demand, and service requests evolve dynamically during route execution [28, 33]. The Dynamic Vehicle Routing problem (DVRP) extends the traditional VRP by assuming some input parameters are unknown at runtime, and revealed or altered incrementally during the run. This research focuses on the stochastic capacitated vehicle routing problem (SCVRP), a form of dynamic problem, in which customer requests arrive according to a stochastic process with a known probability distribution. The objective function is formulated in Eqn. (1.0.1), subject to the constraints outlined in Eqns. (A.1.1)-(A.1.5) in Appendix A [5].

$$\text{DVRP}(V, K, T) = \min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij}(t)\, x_{ij}^{k}. \tag{1.0.1}$$

The objective function of this research aims to **minimise the total cost of travel** for all vehicles ($k \in K$) across all locations ($i, j \in V$) at different times ($t \in T$). $c_{ij}(t)$ denotes the cost of traveling from location i to location j at time t, and x_{ij}^{k} is a binary variable indicating whether vehicle k travels directly from location i to location j. The goal is to find the set of routes for all vehicles that minimises the sum of these time-dependent travel costs.

Traditional methods used to solve the VRP and its variants are often categorised into exact, heuristic, and metaheuristic methods. Exact methods formulate the problem as a mathematical model and, although computationally expensive, find optimal solutions. Heuristic and metaheuristic methods are rule-based procedures to find near-optimal solutions in a time-effective manner. Heuristic methods are often tailored to solve only a single problem, as they utilize information specific to the problem at hand to achieve their solution. Metaheuristic methods are more generic and capable of handling a wide range of different problems [18].

The stochastic nature of the problem outlined requires the continual reevaluation of routing decisions, as new customer requests arrive during runtime. The growth of dynamic logistic applications in ride-sharing (Uber), last-mile delivery (Amazon), and fleet management has increased the need for methods capable of real-time operational changes in vehicle routing [23]. Reinforcement learning (RL) provides a framework for sequential decision-making under uncertainty, where agents learn optimal policies by interacting with their environment through reward signals. This approach has gained traction as an alternative to traditional methods due to its ability to accommodate changing conditions during runtime.

1.1 Problem Statement

The stochastic nature of DVRPs poses new challenges to traditional algorithms, extending beyond the known challenges as problem complexity scales. Traditional methods face solution degradation as larger instances are introduced, with an increase in computational resources that can render these algorithms infeasible. RL methods emerge as a promising alternative to traditional methods due to their adaptability and efficiency in handling stochastic environments. Current research suggests that RL methods have the potential to bridge this gap; however, these studies have not been

conducted on dynamic problems, nor have they addressed the impact of different action spaces on these problems. Soft Actor-Critic (SAC) has the potential to address these concerns, as it possesses the characteristics necessary to solve complex DVRP instances; however, the requirement of a continuous action space for the algorithm could hamper its effectiveness as the problem is intuitively discrete. Thus, this research investigates whether SAC's continuous action space fundamentally limits its applicability to discrete optimization problems like the VRP. This research aims to rectify the literature gap by extensively documenting the aforementioned concerns.

1.2 Research Aims

This research aims to benchmark the performance of RL methods against traditional methods in solving instances of the SCVRP, namely Clarke-Wright Savings, Adaptive Large Neighbourhood Search (ALNS) and SAC. The methods will be benchmarked on the metrics described in Section 3.3.7, with the intent to answer the following:

- How do RL methods perform compared to traditional methods on the metrics described in Section 3.3.7 in solving different-sized instances of the SCVRP?
- What are the limitations of RL methods, specifically SAC and a continuous state space, in solving SCVRP instances compared to traditional methods, and how do their inherent characteristics impact performance?
- How efficiently do algorithms handle the dynamic aspects of customer arrivals, and what are the implications for real-world deployment?

2 BACKGROUND

The VRP was formally introduced by Dantzig and Ramser in 1959, with their publication of the Truck Dispatching Problem, which proposed a simple match-based heuristic for minimising the distance of a fleet of identical oil trucks to serve a large number of gas stations from a central terminal [6, 16]. The VRP has since evolved into one of the most studied combinatorial optimisation problems, with numerous variants being introduced to address real-world constraints, such as time windows (VRPTW), capacity limitations (CVRP), and multiple depots [9, 28].

The NP-hard nature of the VRP and its variants has driven extensive research into solution methodologies that can scale with the complexity of the problems. Traditional VRP methodology consists of exact, heuristic, and metaheuristic methods. Exact methods guarantee globally optimal solutions; however, their use is limited to small instances due to their exponential growth in computational efficiency as problem sizes scale [1].

Heuristic methods emerged as practical solutions to the computational complexity of VRPs, with the introduction of the Clarke-Wright Savings algorithm in 1964, which established the foundation for construction-based approaches that remain in use to this day [4]. The savings heuristic led to a multitude of variants and improvements, including nearest-neighbor algorithms, insertion heuristics, and sweep methods, with each subsequent implementation exploiting different geometric aspects of the problem [9].

Metaheuristic methods, built upon the success of heuristics, gained prominence in the 1980s and 1990s. These methods sought to implement sophisticated search strategies to solve VRPs. Tabu search, first implemented by Glover in 1986, aimed to utilise memory structures to guide the search process and prevent cycling back to previously visited solutions [8]. Taillard championed Tabu Search implementations on VRPs in 1993, where he introduced sophisticated neighbourhood structures and adaptive memory mechanisms to achieve record solutions of that time [27]. Osman introduced Simulated Annealing to the VRP in 1993, demonstrating the potential of local search methods to escape local optima [21].

The Adaptive Large Neighbourhood Search algorithm was introduced by Ropke and Pisinger in 2006, and represented significant advancements in metaheuristic applications by combining multiple destroy and repair operators to achieve near-optimal solutions across diverse problem instances [24]. Population-based metaheuristics, such as Genetic Algorithms and Ant Colony optimisation, have contributed to the vast pool of solution methods capable of achieving near-optimal solutions on benchmark instances.

Advances in deep learning and neural network architecture have shifted the VRP research paradigm to RL approaches. Nazari et al. introduced attention mechanisms for the VRP in 2018, demonstrating that neural networks are capable of learning routing policies that compete with traditional methods on small instances [20]. This innovation was followed by Kool et al.'s Attention Model approach, which further bridged the gap to traditional methods through an improved encoder-decoder architecture and training procedures [14]. Subsequent research by Kwon et al. introduced Proximal Optimisation with Multiple Optima (POMO), which addressed the issue of multiple optimal solutions through population-based training approaches that outperformed traditional methods on benchmark instances [15]. Recent studies have extended to solving differing variants of the VRP, including dynamic problems, with reports finding that RL algorithms outperform traditional methods on solving benchmark instances of the prescribed problems [12, 13].

3 METHODOLOGY

This section details the methodology used in the experimental design, algorithm implementations, and evaluation metrics to compare traditional and RL approaches to the DVRP.

3.1 Traditional Methods

Traditional methods for solving the DVRP are often categorised into exact, heuristic and meta-heuristic methods.

3.1.1 Clarke Wright Savings. The Clarke-Wright Savings algorithm, introduced by Clark and Wright in 1964, represents one of the fundamental algorithms for solving the capacitated VRP, with succeeding algorithms often measured against its performance [4, 26]. The algorithm consolidates routes on the principle of savings maximization, where the savings S_{ij} for combining customers i and j into a single route is calculated as:

$$S_{ij} = c_{0i} + c_{0j} - c_{ij}. \tag{3.1.1}$$

where c_{0i} and c_{0j} denote the distance between customers i and j from the depot, and c_{ij} denotes the distance between customers i and j. The formula calculates the cost reduction of serving customers i and j consecutively as opposed to separate depot visits.

An initial solution is constructed by routing an individual vehicle to each customer in a direct depot-customer-depot trip. These routes are then systemically merged in descending order of savings value, adhering to predetermined capacity constraints and route connectivity requirements.
The algorithm does not guarantee optimal solutions; however, it can produce close to optimal solutions on moderately sized instances in time complexity $O(n^2 logn)$. This lends itself to the continued use of the algorithm as a suitable baseline for comparative analysis [28].

3.1.2 Adaptive Large Neighbourhood Search. ALNS is a metaheuristic framework that extends the Large Neighbourhood Search (LNS) algorithm through adaptive operator selection, first introduced by Ropke and Pisinger in 2006 [19, 24]. LNS algorithms systemically explore large subsections of the solution space by alternately destroying and repairing solutions based on a single destroy or repair method [22]. ALNS deploys multiple destroy and repair methods, each with its own weight that determines how often they are used during the search. These weights are adjusted dynamically during the search so that the methods adapt to the specific problem instance, based on the method's performance.
The ALNS algorithm starts with an initial solution and, in each iteration, selects destroy and repair operators to modify it. Destroy operators remove a subset of the current solution, and the repair operators reinsert these solutions, intending to find a better solution.

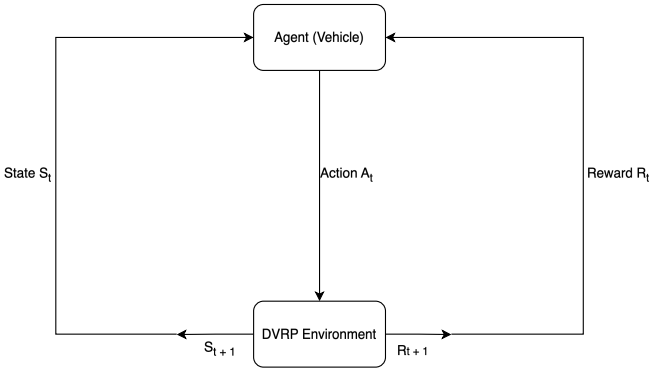
3.2 Reinforcement Learning

RL provides a distinct approach to solving DVRPs, as the problem can be framed as a sequential decision-making problem under uncertainty [15]. RL agents can learn optimal policies through interaction with the environment, potentially discovering routing strategies that are limited by explicit mathematical programming or heuristic design [2]. The theoretical foundation of RL relies on the Markov Decision Process (MDP) framework, in which an agent makes sequential decisions, called actions, based on current state information to maximise cumulative reward over time [20]. The DVRP can be formulated as an MDP, defined by the tuple (S, A, P, R, y), where S represents the state space, the set of all possible states in which the environment can be, A represents the action space, the set of all possible actions the agent can take in each state, P is the transition probability defining the probability of transitioning from state S to state S' given action A, and R being the numerical reward assigned to the agent for transitioning from state S to state S'. y is the discount factor, a number between 0 and 1 (inclusive), that weighs the importance of future rewards to immediate rewards [15]. The framework is outlined in Fig. 3.2.1

The objective of an RL algorithm is to maximise the expected sum of rewards, as shown in Eqn. (3.2.1), by learning a policy $\pi(a_t|s_t)$ that maximises said objective [11].

$$\mathbb{E}_{(s_t,a_t)\sim\rho^\pi}[r(s_t,a_t)]. \tag{3.2.1}$$

Figure 3.2.1: Markov Decision Proccess



3.2.1 Soft Actor-Critic. SAC is an off-policy actor-critic algorithm that utilises maximum entropy principles to negate the exploration-exploitation trade-off that affects RL usage for optimisation problems [10, 11]. SAC extends the RL objective by adding an entropy term that encourages exploration, allowing the algorithm to learn more robust policies in complex environments. A stochastic policy is used, measured by

$$H(\pi(\cdot \mid s_t)),$$

representing the entropy of the policy at state s_t, in contrast to the deterministic policies used by its predecessors, since the objective is now to maximise both cumulative reward and entropy. The objective has thus evolved into:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))\right], \tag{3.2.2}$$

where $r(s_t, a_t)$ is the reward at time t, and α represents the temperature parameter that controls the weight of the entropy term.

SAC follows the actor-critic architecture, comprising an actor that learns the policy mentioned above, and two critic networks that estimate Q-values. The dual critic design is borrowed from the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, serving to mitigate overestimation bias by using the minimum of the Q-values [7].

The critic networks are updated using the soft Bellman equation:

$$Q^*(s,a) \approx r + \gamma(Q^*(\tilde{s},\tilde{a}') - \alpha \log \pi(\tilde{a}'|\tilde{s}')) \quad \text{where} \quad \tilde{a}' \sim \pi(\cdot|\tilde{s}'), \tag{3.2.3}$$

where $Q^*(s,a)$ is the optimal Q-value, $(Q^*(\tilde{s},\tilde{a}')$ is the optimal Q-value for the next action-state pair, $\log \pi(\tilde{a}'|\tilde{s}'))$ is the log probability of action $\tilde{a}'$ given state $\tilde{s}'$ and $\tilde{a}' \sim \pi(\cdot|\tilde{s}')$ is the next action sampled from the policy distribution given the next state.

One of SACS' key innovations is the automatic adjustment of the temperature parameter, α, which ensures that a target entropy level is maintained throughout training, bypassing the need for manual hyperparameter tuning of the exploration-exploitation balance and allowing SAC to remain robust among various problem domains.

These characteristics are well-suited for combinatorial optimisation problems, such as the VRP, where sparse rewards and large action spaces hinder algorithm performance. SAC requires a continous action space, which contradicts the discrete nature of the

problem. While this creates a fundamental architectural challenge, investigating this limitation provides valuable insights for future RL-VRP research.

3.3 Design and Implementation

This section outlines the experimental design of this research. The implementation methodologies for algorithms are detailed, and the dataset and preprocessing setup are outlined. The metrics used to evaluate and compare algorithms are discussed.

3.3.1 Experimental Settings. The performance of the algorithms was assessed on a MacBook Air (2020), with specifications detailed in Table 3.3.1.

Table 3.3.1: Hardware Specifications

Component	Specification
Device	2020 MacBook Air
Processor	Apple M1 (8-core CPU, 7-core GPU)
Memory	8GB Unified Memory
Storage	256GB SSD
Architecture	ARM64
Operating System	macOS Sonoma 14.4.1

All algorithms have been implemented using Python. All code, including algorithm implementation, experimental results, and data input, has been made publicly available in a Github repository, ensuring the reproducibility of these experiments.

3.3.2 Dataset and Preproccessing. The dataset used for this experiment is an adaptation of the Solomon VRPTW benchmark instances [25]. Each Solomon instance has been reduced by removing the time window constraint to model the input as a CVRP. With the removal of time windows, this experiment assesses algorithm performance on 18 unique instances, where each instance has either 25, 50, or 100 customers. Instances are categorised into three distinct patterns depending on customer clustering patterns, namely clustered (C), random (R), and random-clustered (RC). Fig. 3.3.1 plots the customer coordinates for instances C1_100, R1_100 and RC1_100, illustrating the spatial patterns of the different instance types.

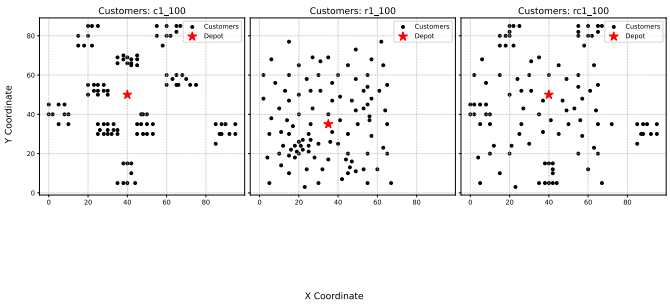
Each algorithm implementation contains a distance matrix function that calculates the Euclidean distance between customers, given the input as coordinates. The stochastic element is introduced by modelling customer arrivals as a Poisson process [17, 33]. The following hybrid model details the static and dynamic split:

- 70% of customers will be static - available to the algorithms
- 30% of customers will be dynamic - they will request service during the run, and will be randomly selected from the customer set.

The probability of observing k arrivals within interval time t follows the Poisson distribution in Eqn. (3.3.1). The arrival rate, λ, controls the average number of customers per unit time.

$$P(X = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}. \tag{3.3.1}$$

Figure 3.3.1: Customer Location for 100-customer instances



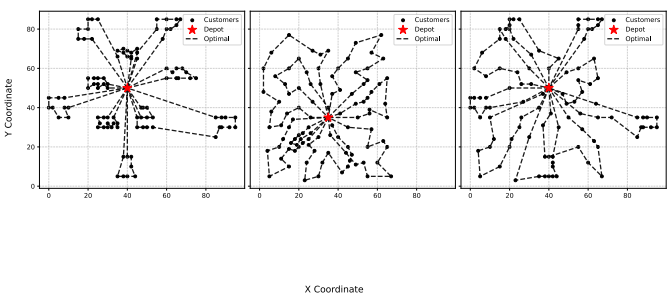
The time between consecutive arrivals is exponentially distributed with a mean of $\frac{1}{\lambda}$. λ and can be adjusted to control how frequently new customers request service.

The speed of the Clarke-Wright and ALNS algorithms, demonstrated in Table 4.2.1, does not allow for real-time customer introduction. The stochastic customer introduction occurs on a time ticker implemented in each algorithm, at which the algorithm checks if any new customers have been made available and adds them to the unassigned list. The RL implementation reveals new customers when the reset function in the environment is called.

3.3.3 Benchmark. Algorithm performance is measured against PYVRP's state-of-the-art genetic algorithm solver [32]. PYVRP's solver has been benchmarked with results within 0.25% of optimal CVRP solutions on version 0.11.0, rendering it an appropriate benchmark to compare this research's algorithms against, as optimal solutions for the reduced Solomon files used in this research have not been computed in prior literature. Table C.0.1 in Appendix C shows the optimal objective cost on each instance, and serves as the base for all algorithmic comparisons.

Fig. 3.3.2 shows the optimal routes for differing 100-customer instances.

Figure 3.3.2: Optimal Routes on 100-customer Instances



3.3.4 Clarke-Wright Savings. The implementation of the Clarke-Wright Saving algorithm for this research follows the core procedures, with several additional design decisions to handle the dynamic nature of the DVRP, specifically the introduction of stochastic customer arrivals and the handling of capacity constraints. The initial solution is constructed by creating individual depot-customer-depot routes for each available customer at the start of the optimisation phase.

The savings values for all pairs are then calculated using the Euclidean distance between customer pairs, rounded to the nearest integer value to prevent floating-point precision errors during route optimisation. The savings list is then sorted in descending order using Python's Timsort algorithm, with a time complexity of $O(n^2 logn)$ for the sorting phase.

Strict route merging logic is implemented, and routes can only merge if the following conditions are met: (1) The combined customer demands in a merged route do not exceed vehicle capacity,

$$\sum_{i \in \text{route1}} d_i + \sum_{j \in \text{route2}} d_j \leq Q,$$

where d is the demand of the customer and Q is the capacity of the vehicle, (2) The customers to be connected are at the end points of the route to ensure a valid construction of the tour.

For dynamic arrivals, the algorithm collects all currently assigned customers, unassigned customers, and recently arrived customers and performs a complete reoptimisation. This approach is more computationally expensive than a greedy insertion approach; however, it ensures that new arrivals are optimally inserted into the current route structure.

Routes that exceed capacity constraints are automatically discarded, with customers in these routes being moved to the unassigned list at a penalty cost to the objective function. This approach maintains the feasibility of the solution whilst quantifying the cost of failure.

The algorithm terminates early if savings values become negative, negating the cost of computation on infeasible merges. The linear search overhead associated with locating clients is bypassed by maintaining an index of routes during the merging phase.

3.3.5 Adaptive Large Neighbourhood Search. The ALNS implementation follows the framework proposed by Ropke and Pisinger, using Wouda's ALNS Python package, version 7.0.0 [24, 31]. The algorithm employs two destroy operators, random and string removal, as well as a singular greedy repair operator.

The greedy repair operator iterates over the unassigned list to find the best route and position to insert a customer, being the insertion with the lowest cost. It evaluates all insertion positions across all routes and chooses the one with the least cost, subject to capacity constraints. If no route can accommodate a new customer, a new single customer route is created.

The initial solution is built using the greedy repair operator. The algorithm shuffles the unassigned list before finding the best insertion point for a customer, iteratively building a solution. This approach enables the rapid construction of initial routes, yielding better solutions than those generated by random route constructions. Other heuristic constructions, such as nearest neighbour, could be considered.

The random destroy operator removes a random subset of the solution, leaving the state incomplete. The degree to which the current solution is destroyed is controlled by the destruction rate parameter, which in this implementation is set to 0.2, meaning 20% of the current route is removed.

The string removal destroy operator implements a proximity-based strategy that removes connected sequences of customers [3]. The operator removes partial routes, known as strings, that are spatially related by selecting a random centre customer and removing said strings from nearby routes. The size of the removal of the string is limited by the max_string variable, set to 6, to prevent excessive destruction. The max_removals variable, set to 2, controls how many routes can be affected by substring removal. String removal exploits the spatial structure of VRPs, making it an appropriate choice for the problem.

A roulette wheel scheme is implemented to choose destroy and repair operators during the search process. Each operator i is assigned a weight $w_i = 1$, and repair and destroy operators are selected based on the current weights. The operators are applied to the current solution to create a new candidate solution m, where the Roulette wheel assigns scores, s_j, where $j = (1, ..., 4)$, for the following four outcomes:

- **1.** The candidate solution is a new global optimal - $s_j = 25$
- **2.** The candidate solution is better than the current solution, but not globally optimal - $s_j = 5$
- **3.** The candidate solution is accepted - $s_j = 1$
- **4.** The candidate solution is rejected - $s_j = 1$

After observing outcome j, the operator weights are updated as detailed in Eqn. (3.3.2),

$$w = \theta w + (1 - \theta) s_j, \tag{3.3.2}$$

where θ is the rate of decay, set to 0.8, controlling how quickly the algorithm adapts to recent performance.

ALNS accepts solutions based on Record-to-Record Travel (RTT), where new solutions must meet a dynamically updating threshold to be accepted. This threshold begins at 2% of the current solution's objective value and gradually reduces to zero over the optimisation iterations.

The algorithm handles dynamic customer arrivals through re-optimization iterations. At time t, the newly revealed customers are added to the unassigned list, and a new RTT acceptance criterion is invoked. The stopping criterion for each re-optimisation phase is set to 3000 iterations, ensuring sufficient time for solution space observation whilst maintaining computational efficiency.

3.3.6 Soft Actor-Critic. The implementation of SAC for this research is accomplished through the stable-baselines 3 Python package. This research uses their available functions, hyperparameter defaults, and custom environment compatibility to establish the problem.

The environment extends OpenAI Gym's interface and draws from Nazari et al.'s deep reinforcement learning (DRL) approach [20]. The environment supports instances of up to 100 customers through a fixed observation space of 504 dimensions (4 state features + 5 customer features), applying padding to smaller instances to evaluate one model per instance type.

An action mask is implemented to ensure only valid actions are available to the agent at each step. Valid actions are defined as serving revealed, unvisited customers, and depot returns. The stochastic customer arrivals are implemented by keeping an unassigned list, with dynamic customers added to the list when the environment calls the reset function.

This research employs a custom graph attention network (GAT) feature extractor that processes the observation space. The feature extractor facilitates the processing of spatial relationships and state information, enabling a simplified observation space [29]. The extractor embeds customer features (coordinates, demand, revealed status, visited status) into a 64-dimensional space and processes state features (current position, remaining capacity, time) separately. Customer embeddings are processed through two GAT layers with multi-head attention (4 heads) and dropout (0.1) [14].

VRP agents require a discrete customer selection to build routes, as the agent must select which customer to visit next in the range of $[0..n]$, where n is the number of customers. SAC requires a continuous action space; thus, a custom wrapper is implemented to map SAC's continuous action values (-1, 1) to discrete actions using savings calculations, as detailed in Section 3.3.4. The wrapper computes the savings of routing decisions, uses an adaptive capacity-based depot return decider that scales with customer size, and utilises an exploration vs exploitation tradeoff approach to map the continuous value to discrete actions.

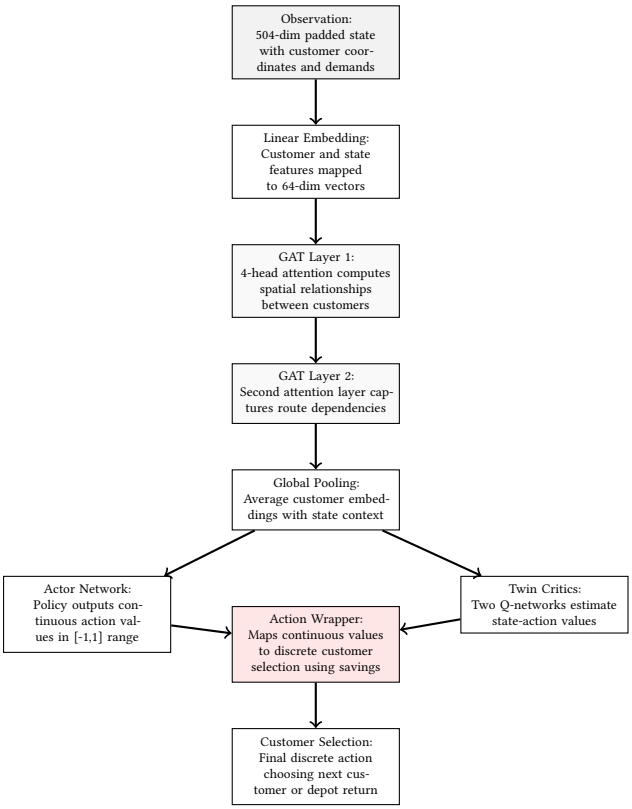


Figure 3.3.3: SAC with GAT Architecture

Fig. 3.3.3 illustrates the SAC architecture pipeline, outlining the process undertaken from receiving an observation to customer selection. Following customer selection, the agent receives feedback through a reward.

The reward function for this implementation follows the best practices established by prior work, specifically the negative value of the total tour length [14, 20]. This approach allows for immediate feedback on route construction decisions.

This research trains three models, one for each type instance (C, R, RC). The model is trained until convergence, using SB3's StopTrainingOnNoModelImprovement callback. Training terminates if no improvement occurs over 5 consecutive evaluations (10,000 time-steps). The final model is then saved and used to evaluate all instances of the type that have been trained.

3.3.7 Metrics. The performance of the algorithms implemented for this research has been evaluated on the following key metrics, with each algorithm run twenty times for validation:

- **Solution Quality** refers to the minimisation of the total cost of the tour, as detailed in Eqn. (1.0.1). An additional cost per vehicle is incurred by the algorithms, set at 50 for this problem. The results achieved by each algorithm are compared with the optimal benchmark result generated by the PYVRP genetic algorithm solver, detailed in Section 3.3.2. The best-case performance of each algorithm is used to evaluate its performance across multiple runs. Performance is expressed as the percentage deviation from the optimal solution.
- **Computational Time** measures the time an algorithm takes to run until completion. For the traditional methods, computation time is measured as the direct solution time. For RL, this will include the one-time overhead of training a model as its own metric, as well as the time required to evaluate a pre-trained model in comparison to traditional methods. This metric provides key insight into the feasibility of using these algorithms in real-time applications.
- **Feasibility** shows the proportion of valid solutions generated by each algorithm, defined by a solution in which all dynamic customers are served, and all capacity and connectivity constraints are met. This metric allows for a distinction between algorithmic capability and solution quality performance.
- **Scalability** is used to measure an algorithm's capability to produce solutions for increasing problem complexities. The percentage changes in solution quality and computational time will be used to demonstrate how algorithms perform on more complex instances, thereby determining their ability to be employed on problems beyond those addressed in this research.
- **Success Rate** measures the frequency with which an algorithm solves within a 10%, 5% and 1% gap to the optimal benchmark. This provides insights into the reliability of near-optimal solutions across different problem characteristics, cluster types, and numbers of customers.

4 RESULTS

This section discusses the results obtained by each algorithm, using the comparative metrics described in Section 3.3.7 to evaluate performance.

4.1 Solution Quality

Comparative analysis of solution quality reveals significant discrepancies between algorithms, as detailed in Table 4.1.1.

ALNS finds the globally optimal solution in 94% of instances, with an average deviation of 0.054%, demonstrating superior optimisation capabilities. The solution deviation occurs on instances r2_100 (0.14%), rc1_100 (0.57%), and rc2_100 (0.27%), indicating a slight decrease in solution quality in more complex instances.

Clarke-Wright consistently produces good solutions in all instances, with an average deviation of 4.26% from the optimal solutions. The algorithm shows stable performance, with average deviation ranging from 0.92% (rc1_50) to 7.8% (r2_50 & r2_100), highlighting the robustness of the savings-based construction approach when applied to VRPs.

SAC demonstrates substantially inferior solution quality, with an average deviation of 18.98% from optimal solutions. SAC's performance is characterised by high variability, with solutions ranging from 3.85% (rc1_25) to 53.54% (rc2_100) off the optimal solutions. Seven instances exhibit deviations greater than 20%, indicating fundamental limitations in SAC's architecture that hinder effective vehicle routing, as results contradict those in prior RL VRP research [14, 15, 20, 30] in which the proposed methods outperform traditional methods.

Table 4.1.1: Solution Quality

Instance	Clarke-Wright		ALNS		SAC	
	Cost	Dev%	Cost	Dev%	Cost	Dev%
c1_25	338.00	2.42	330.00	0.00	376.00	13.93
c1_50	603.00	1.34	595.00	0.00	717.00	20.50
c1_100	1307.00	0.93	1295.00	0.00	1608.00	24.16
c2_25	248.00	4.20	238.00	0.00	248.00	4.20
c2_50	454.00	5.58	430.00	0.00	494.00	14.88
c2_100	745.00	5.97	703.00	0.00	976.00	38.83
r1_25	450.00	4.90	429.00	0.00	484.00	12.82
r1_50	760.00	6.59	713.00	0.00	846.00	18.65
r1_100	1261.00	6.06	1189.00	0.00	1410.00	18.59
r2_25	373.00	4.19	358.00	0.00	390.00	8.94
r2_50	539.00	7.80	500.00	0.00	572.00	14.40
r2_100	774.00	7.80	718.00	0.14	891.00	24.09
rc1_25	447.00	1.36	441.00	0.00	458.00	3.85
rc1_50	767.00	0.92	760.00	0.00	917.00	20.66
rc1_100	1457.00	4.00	1401.00	0.57	1761.00	25.70
rc2_25	284.00	4.80	271.00	0.00	307.00	13.28
rc2_50	428.00	3.63	413.00	0.00	457.00	10.65
rc2_100	765.00	4.22	734.00	0.27	1127.00	53.54
Average	–	**4.26**	–	**0.054**	–	**18.98**

Fig. 4.1.1 shows the absolute solution cost achieved by each algorithm across all instances. The figure displays a cost hierarchy between the algorithms, with ALNS consistently achieving the lowest cost, Clarke-Wright producing marginally higher costs, and SAC achieving substantially higher costs across most instances. The notable exception occurs on c2_25, where SAC matches Clarke-Wright's performance.

Figure 4.1.1: Solution Cost for Each Instance

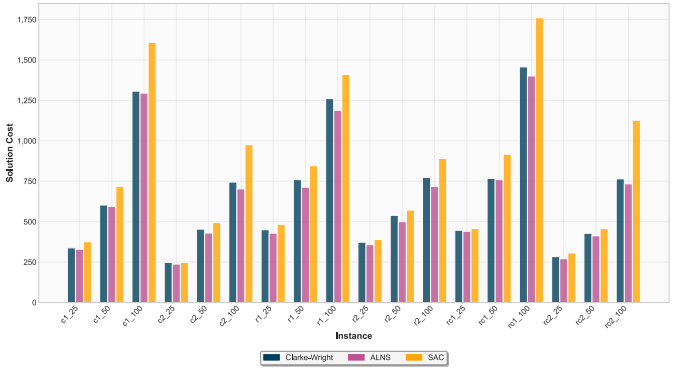


Fig. 4.1.2 illustrates the scalability of each algorithm's problem size. Scalability with respect to problem size differs significantly between algorithms. On 25 customer instances, the performance gap between algorithms is moderate, with ALNS achieving 0% average deviation, Clarke-Wright achieving 3.61%, and SAC achieving 9.52%. This gap expands exponentially as the problem size increases. On 100-customer instances, ALNS maintains near-optimal performance with an average deviation of 0.16%, Clarke-Wright demonstrates controlled solution quality degradation at 4.76%, while SAC shows severe performance collapse with the average deviation rising to 30.82%. The 3.23 fold degradation indicates fundamental architectural limitations in SAC's ability to solve complex VRP instances. The continuous action mapping design, which maps the neural network outputs to discrete actions through savings calculations, results in increasingly suboptimal customer selection as the solution space grows exponentially, as the amount of viable options within similar continuous ranges increases.

Figure 4.1.2: Average Deviation per Customer Count

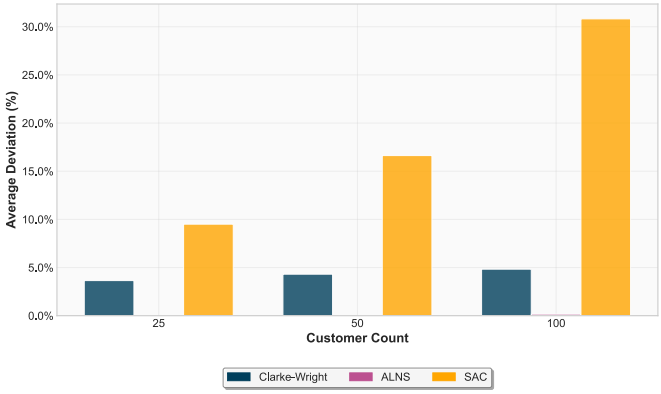
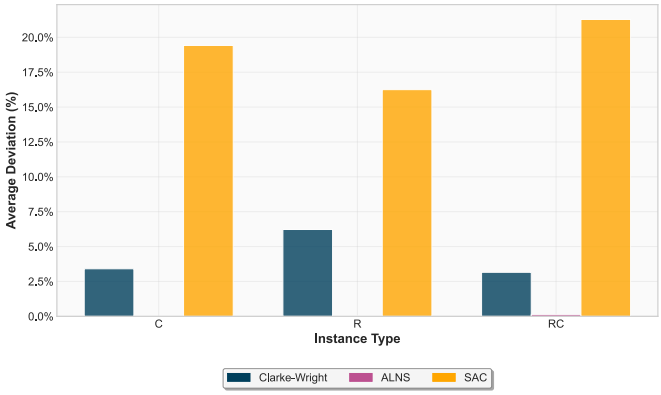


Fig. 4.1.3 illustrates that instance characteristics significantly influence algorithm performance. Traditional methods favour clustered instances (C-type), with Clarke-Wright achieving 3.68% average deviation on these instances compared to 5.66% on random instances. This preference is attributed to the savings formulas' inherent geometric bias, which favours merging customers that are proximal to each other. ALNS maintains consistent optimal or near-optimal performance across all instances, highlighting the algorithm's robustness in handling varying spatial configurations.

Figure 4.1.3: Average Deviation per Instance Type



SAC performs relatively better on random instances (R-type), with an average deviation of 16.97% compared to 20.87% on clustered instances. This gap suggests the GAT's architecture struggles to interpret the spatial clustering patterns that the traditional methods naturally exploit. This failure highlights the GAT's inability to capture the hierarchical spatial relationships that are critical to the VRP, and that the sequential reasoning requiring global route context is not effectively modelled by the GAT, which instead focuses on local route optimisation.

Mixed instances (RC-type) pose challenges for all algorithms, with SAC demonstrating particularly poor performance with an average deviation of 18.12%.

Figure 4.1.4: Routes Produced on Instance C1_25

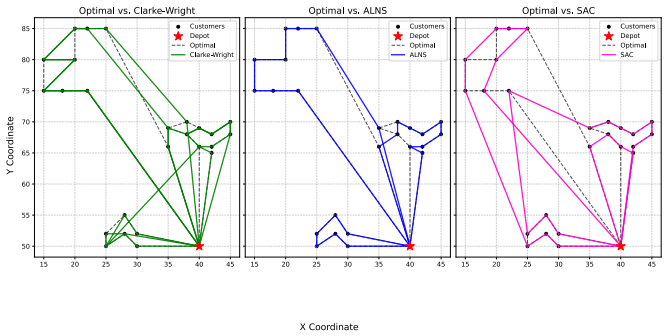


Fig. 4.1.4 visualises the routes produced by each algorithm on instance C1_25, and shows the spatial distribution and routing efficiency between algorithms. Clarke-Wright produces well-structured routes that closely follow the optimal solution. ALNS notably deviates from the optimal benchmark route; however, it matches benchmark performance, as shown in Table 4.1.1, revealing that multiple global optima exist for this instance. SAC produces longer, less efficient routes, highlighting its struggle with accurately capturing customer clusters.

Combining clustered and dispersed customers creates complex spatial relationships that present a particular challenge for neural pattern recognition; however, they remain manageable for traditional methods. Continuous action space mapping hampers performance, as the clear spatial structures inherent to these instances require discrete customer selection decisions.

4.2 Computational Time

The computational efficiency of each algorithm is detailed in Table 4.2.1, with considerable differences in CPU time among the algorithms.

Clarke-Wright presents immense computational efficiency with an average runtime of 0.162 seconds across all instances, and maintains sub-second performance on complex 100-customer instances. The efficiency demonstrated highlights Clarke-Wright's feasibility for deployment in real-time dynamic environments where immediate customer response is paramount.

ALNS requires more computational resources, with an average of 24.71 seconds across all instances. The runtime scales from an average of 2.82 seconds on 25-customer instances to an average of 68.55 seconds on 100-customer instances. The 25-fold increase in problem complexity creates a clear quality-speed tradeoff, as the increased runtime is accompanied by superior solution quality, as discussed in Section 4.1.

SAC's computational requirements differ from traditional methods, as the models incur a one-time training overhead before evaluation. Training times range from 7463 seconds (~2 hours) to 10858 (~3 hours), as shown in Table 4.2.2. This overhead represents a substantial one-time computational investment. Once a model is trained, SAC presents a negligible inference time (> 0.5s), providing

Table 4.2.1: Average Computation Time (seconds)

Instance	Clarke-Wright	ALNS
c1_25	0.054	3.17
c1_50	0.110	11.09
c1_100	0.403	65.56
c2_25	0.059	2.99
c2_50	0.076	13.46
c2_100	0.531	79.82
r1_25	0.049	2.82
r1_50	0.076	13.22
r1_100	0.395	64.68
r2_25	0.055	2.81
r2_50	0.070	10.66
r2_100	0.419	68.84
rc1_25	0.064	2.69
rc1_50	0.143	11.16
rc1_100	0.398	61.53
rc2_25	0.051	2.41
rc2_50	0.068	10.97
rc2_100	0.488	70.84
Average	**0.162**	**24.71**

Table 4.2.2: Average SAC Training Time (Seconds)

Model	Time
C-Model	7463
R-Model	7308
RC-Model	10858
Average	**8543**

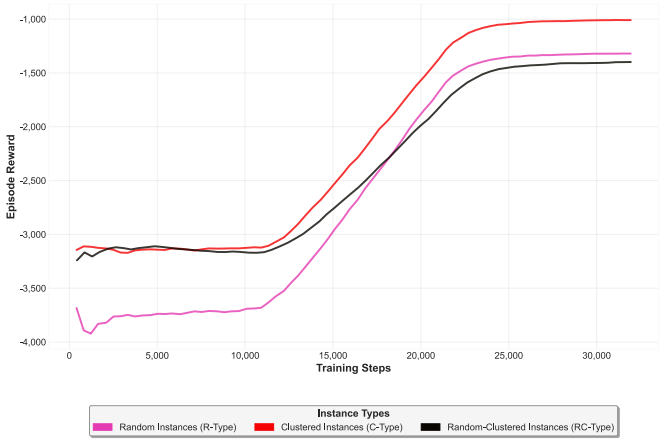
a key computational advantage over traditional algorithms for solving complex instances; however, this advantage is offset by poor solution quality (18.98% average deviation), as detailed in Section 4.1.

Fig. 4.2.1 shows the training logs for each model type. All models converge around 32,000 time-steps.

Clarke-Wright exhibits linear scaling characteristics, with an approximate 10-fold increase from 25 to 100 customer instances. ALNS exhibits polynomial-time scaling behaviour with an approximate 25-fold increase over the same range, which limits its use for large-scale, real-time deployment. SAC is able to train one model for any problem size, allowing no note on its time scalability.

For dynamic routing requirements, algorithm speed directly correlates to the algorithm's capability in handling dynamism. Clarke-Wright's sub-second computations allow for immediate optimisation of dynamic requests, highlighting its efficiency in real-time decision making. The moderate computational investment required by ALNS may limit its use in highly dynamic environments; however, it remains viable in applications that can handle brief optimisation

Figure 4.2.1: SAC Training logs per Model Type



delays. SAC's training requirements create limitations in dynamic environments, as a new model must be trained should problem characteristics change. The inference advantage is therefore negated should deployment require inputs that differ in structure from the current model.

Clarke-Wright is best suited for time-critical applications where algorithm speed requirements outweigh the need for solution quality. ALNS is suited towards applications in which solution quality is prioritised over computation speed. SAC's poor quality-to-computation costs reveal its unsuitability for real-time vehicle routing applications, despite the advantages of low inference and model generalisation.

4.3 Feasibility and Success Rate

All algorithms achieve valid solutions across each instance, shown in Table 4.3.1, highlighting their robustness in handling the stochastic nature of the problem.

Table 4.3.1: Success and Feasibility Rates (%) by Algorithm

Algorithm	Success Rate			Feasibility Rate
	90%	95%	99%	
Clarke-Wright	100.0	67.0	11.0	100.0
ALNS	100.0	100.0	100.0	100.0
SAC	17.00	2.00	0.00	100.00

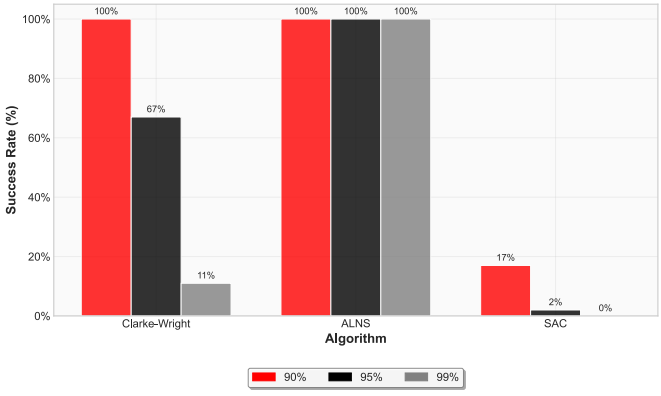
The success rates of each algorithm are detailed in Table 4.3.1, as well as illustrated in Fig 4.3.1. ALNS achieves a 100% success rate at each threshold, consistently finding solutions within 1% of the optimal cost. As detailed in Section 4.1, it finds the optimal solution in 94% of instances, highlighting its reliability for real-world deployment.

Clarke-Wright produces solutions within 10% of optimal in each instance. A decrease in algorithm success is observed within the 5% and 1% range. All 25 customer instances fall within the 5% range, demonstrating Clarke-Wright's ability to solve smaller complexity

problems at an efficient rate. Success generally decreases for Clarke-Wright as more complex instances are tested, with high variability observed in the most complex instances.

SAC fails to solve within 1% of the optimal solution. 17% of SAC's solutions fall within the 90% threshold, and 2% of solutions are within the 95% threshold, with all threshold solutions comprising 25 customer instances. This highlights SAC's unreliability in solving complex instances of the SCVRP, as the complexity of the instance has minimal bearing on the success of the solution.

Figure 4.3.1: Success Rates of Algorithms at each Threshold



5 CONCLUSIONS

This research provides empirical evidence on whether DRL methods can compete against traditional methods in solving DVRPs. The evaluation demonstrates that SAC does not match the performance of traditional methods. SAC has been consistently outperformed by ALNS and Clarke-Wright across solution quality, feasibility, success rate, and scalability, except for computation time, for which the advantage is negated by poor solution quality. SAC's continuous action space can be considered as a main driver of its shortfall, as the need to map continuous actions to discrete customer selections, as required by the VRP, creates a fundamental mismatch in the necessary architecture for combinatorial optimisation problems. This is further supported by the results achieved in prior research, which found that all algorithms follow discrete customer selection approaches [14, 15, 20]. In these studies, DRL algorithms effectively bridged the gap to traditional performance. This study finds that algorithm selection for VRP applications should prioritize the context and requirements of the problem, rather than the characteristics of an algorithm that may be well-suited to solving the problem, such as SAC's entropy regulation.

Clarke-Wright and ALNS demonstrate immense reliability in solving the SCVRP. ALNS solution quality results highlight the effectiveness of metaheuristics for solving the VRP and its variants. The sharp increase in solution time as the problem complexity increases raises questions about its deployment on enterprise-level problems, as the time accrued solving the problem could become infeasible.

Clarke-Wright's combination of near-optimal solution quality and minimal computation time reinforces its inclusion as one of the prominent VRP-solving algorithms. The results from this research indicate that the solution quality versus computational time tradeoff achieved by Clarke-Wright highlights its continued use in the logistics industry for quick and accurate routing.

This research concludes that whilst SAC does not outperform traditional methods in solving the SCVRP, RL algorithms have the potential to provide scalable solutions for complex dynamic environments where traditional methods become computationally expensive. Current research suggests that RL algorithms can outperform traditional methods on static instances of the VRP and its variants, providing evidence that these algorithms can be adapted for use in real-time applications.

6 LIMITATIONS

This research has been conducted in the presence of several limitations that should be considered in the findings of this research. All computations and experiments have been conducted on a personal device, which limits the computing power available to solve problems of this complexity. The RL training process requires significant computational power and domain-specific hyperparameter tuning, which is infeasible on the device used for this research. Additionally, this research only implemented a single RL algorithm, SAC, and therefore lacks the baseline to conclusively determine the abilities of RL algorithms to solve the VRP. SAC's continuous action space also limits this research's ability to determine the abilities of discrete action space algorithms in solving the VRP. Other algorithms, such as the Attention Model and Policy Optimisation with Multiple Optima (POMO), have been able to bridge the gap between RL and traditional approaches [14, 15].

The dynamic problem formulation focused on a single stochastic scenario, in which customers were introduced dynamically during runtime following a set Poisson process. Real-world dynamic scenarios exhibit varying dynamic characteristics, where the introduction of customers may be dynamic, and other variables may also change, such as route lengths and capacity. The framework focused on a single variant of the VRP, where other variants with additional constraints, such as time windows or multiple depots, may present differing algorithmic performance.

7 FUTURE WORK

Future work should be centred around evaluating the efficiency of pre-established algorithms that achieve near-optimal results. This research should be conducted with the computational resources necessary to solve problems of the complexity of the VRP and its variants. A more thorough analysis of these algorithms' capabilities in solving dynamic VRP instances is required, as most research conducted with these approaches has focused on the CVRP rather than problems that mimic real-life applications. These experiments should detail performance across differing variants of the VRP, as well as differing dynamic variables.

REFERENCES

[1] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. 2012. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* 218, 1 (2012), 1–6. https://doi.org/10.1016/j.ejor.2011.07.037

[2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2020. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. arXiv:1811.06128 [cs.LG] https://arxiv.org/abs/1811.06128

[3] Zuzana Borcinova. 2017. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review* 8 (12 2017), 463–469. https://doi.org/10.17535/crorr.2017.0029

[4] G Clarke and J W Wright. [n. d.]. *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points.* Technical Report. 568–581 pages. Issue 4. https://www.jstor.org/stable/167703

[5] Kassem Danach. 2024. *Reinforcement Learning for Dynamic Vehicle Routing Problem: A Case Study with Real-World Scenarios.* Technical Report. 6703 pages. Issue 3. https://ijcnis.org/

[6] G B Dantzig and J H Ramser. 1959. The Truck Dispatching Problem. *Management Science* 6 (10 1959), 80–91. Issue 1.

[7] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. (2 2018). http://arxiv.org/abs/1802.09477

[8] Fred Glover. 1989. Tabu Search—Part I. *ORSA Journal on Computing* 1 (1989). Issue 3. https://doi.org/10.1287/ijoc.1.3.190

[9] Bruce L Golden, S Raghavan, and Edward A Wasil. 2008. *The vehicle routing problem* (2008 ed.). Springer, New York, NY.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs.LG] https://arxiv.org/abs/1801.01290

[11] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. 2019. Soft Actor-Critic Algorithms and Applications. arXiv:1812.05905 [cs.LG] https://arxiv.org/abs/1812.05905

[12] Zangir Iklassov, Ikboljon Sobirov, Ruben Solozabal, Martin Takáč, Abu-Dhabi Editors, Berrin Yanıko, and Wray Buntine. 2023. *Reinforcement Learning for Solving Stochastic Vehicle Routing Problem.* Technical Report. 2023–2023 pages. https://github.com/Zangir/SVRP

[13] Anna Konovalenko and Lars Magnus Hvattum. 2024. Optimizing a Dynamic Vehicle Routing Problem with Deep Reinforcement Learning: Analyzing State-Space Components. *Logistics* 8 (12 2024). Issue 4. https://doi.org/10.3390/logistics8040096

[14] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Attention, Learn to Solve Routing Problems! arXiv:1803.08475 [stat.ML] https://arxiv.org/abs/1803.08475

[15] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. 2021. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. (7 2021). http://arxiv.org/abs/2010.16011

[16] Gilbert Laporte, Paolo Toth, and Daniele Vigo. 2013. Vehicle routing: Historical perspective and recent contributions. *EURO Journal on Transportation and Logistics* 2 (05 2013). https://doi.org/10.1007/s13676-013-0020-6

[17] Allan Larsen. 2000. *The dynamic vehicle routing problem.* Technical Report.

[18] Wanjing Ma, Lin Zeng, and Kun An. 2023. Dynamic vehicle routing problem for flexible buses considering stochastic requests. *Transportation Research Part C: Emerging Technologies* 148 (3 2023), 104030. https://doi.org/10.1016/J.TRC.2023.104030

[19] Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. 2022. A survey of adaptive large neighborhood search algorithms and applications. *Computers and Operations Research* 146 (10 2022). https://doi.org/10.1016/j.cor.2022.105903

[20] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takáč. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. (2 2018). http://arxiv.org/abs/1802.04240

[21] Ibrahim Hassan Osman. 1991. *Metastrategy: simulated annealing and tabu search for combinatorial optimization problems.* Ph. D. Dissertation. Imperial College London (University of London).

[22] D ; Pisinger and S Røpke. 2010. *Large Neighborhood Search.* Technical Report. 399–420 pages.

[23] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. 2016. Dynamic vehicle routing problems: Three decades and counting. *Networks* 67 (1 2016), 3–31. Issue 1. https://doi.org/10.1002/net.21628

[24] Stefan Ropke and David Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40 (2006), 455–472. Issue 4. https://doi.org/10.1287/trsc.1050.0135

[25] Marius M. Solomon. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research* 35, 2 (April 1987), 254–265. https://doi.org/10.1287/opre.35.2.254

[26] Milan Stanojević, Bogdana Stanojević, and Mirko Vujošević. 2013. Enhanced savings calculation and its applications for solving capacitated vehicle routing problem. *Appl. Math. Comput.* 219 (6 2013), 10302–10312. Issue 20. https://doi.org/10.1016/J.AMC.2013.04.002

[27] E. Taillard. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 2 (1993), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M Project Management anf Scheduling.

[28] Paolo. Toth and Daniele. Vigo. 2015. *Vehicle routing : problems, methods, and applications.* Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104). 463 pages.

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML] https://arxiv.org/abs/1710.10903

[30] Zhikang T. Wang and Masahito Ueda. 2021. Convergent and Efficient Deep Q Network Algorithm. (6 2021). http://arxiv.org/abs/2106.15419

[31] Niels A. Wouda and Leon Lan. 2023. ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic. *Journal of Open Source Software* 8, 81 (2023), 5028. https://doi.org/10.21105/joss.05028

[32] Niels A. Wouda, Leon Lan, and Wouter Kool. 2024. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing* 36, 4 (2024), 943–955. https://doi.org/10.1287/ijoc.2023.0055

[33] Jian Zhang, Kelin Luo, Alexandre M. Florio, and Tom Van Woensel. 2023. Solving large-scale dynamic vehicle routing problems with stochastic requests. *European Journal of Operational Research* 306 (4 2023), 596–614. Issue 2. https://doi.org/10.1016/j.ejor.2022.07.015

A MATHEMATICAL FORMULATION

A.1 DVRP Constraints

The DVRP is subject to the following constraints:

$$\sum_{k \in K} \sum_{j \in V, j \neq i} x_{ij}^k(t) = 1, \qquad \forall i \in V \setminus \{0\}, \forall t \tag{A.1.1}$$

$$\sum_{j \in V \setminus \{0\}} x_{0j}^k(t) = 1, \qquad \forall k \in K, \forall t \tag{A.1.2}$$

$$\sum_{i \in V \setminus \{0\}} x_{i0}^k(t) = 1, \qquad \forall k \in K, \forall t \tag{A.1.3}$$

$$q_i^k(t) \leq Q_k, \qquad \forall k \in K, \forall i \in V, \forall t \tag{A.1.4}$$

$$q_j^k(t) = q_i^k(t) - d_j(t) \cdot x_{ij}^k(t), \qquad \forall k \in K, \forall (i,j) \in E, \forall t \tag{A.1.5}$$

Constraint Descriptions:

- **Equation** (A.1.1): Each customer must be visited exactly once by exactly one vehicle
- **Equation** (A.1.2): Each vehicle must depart from the depot exactly once per time period
- **Equation** (A.1.3): Each vehicle must return to the depot exactly once per time period
- **Equation** (A.1.4): Vehicle loads cannot exceed capacity at any time
- **Equation** (A.1.5): Load balance equation tracking demand satisfaction

B ALGORITHM SPECIFICATIONS

B.1 Clarke-Wright Savings Algorithm

Algorithm 1 Clarke-Wright Savings Algorithm for DVRP

Require: Customers $V = \{1, 2, \ldots, n\}$, distances c_{ij}, capacity Q, demands d_i
Ensure: Set of routes R

```
Initialize routes: R_i = {0, i, 0} for all i ∈ V
Compute savings: S_ij = c_0i + c_0j − c_ij for all i, j ∈ V, i ≠ j
Sort savings in descending order
for each (i, j) in sorted savings list do
    if routes R_i and R_j can be merged then
        if Σ_{k∈R_i∪R_j} d_k ≤ Q then
            Merge routes R_i and R_j
        end if
    end if
end for
while dynamic customers arrive do
    Add new customers to unassigned list
    Recompute savings including new customers
    Apply savings procedure
end while
return Routes R
```

Table B.1.1: Notation for Clarke-Wright Savings Algorithm

Symbol	Description
V	Set of customer nodes $\{1, 2, \ldots, n\}$
c_{ij}	Euclidean distance between nodes i and j
Q	Vehicle capacity constraint (units)
d_i	Demand of customer i (units)
R_i	Route containing customer i
S_{ij}	Savings value for merging customers i and j into single route
c_{0i}	Distance from depot (node 0) to customer i
R	Final set of all feasible routes

B.2 Adaptive Large Neighborhood Search

Algorithm 2 Adaptive Large Neighborhood Search for DVRP

Require: Initial solution x, destroy operators D, repair operators R
Ensure: Best solution x^*

Initialize operator weights $w_d = w_r = 1$ for all $d \in D, r \in R$
$x^* \leftarrow x$, $x_{\text{current}} \leftarrow x$
while stopping criterion not met **do**
    Select destroy operator d using roulette wheel with weights w_d
    Select repair operator r using roulette wheel with weights w_r
    $x' \leftarrow \text{REPAIR}(d(\text{DESTROY}(x_{\text{current}})), r)$
    if ACCEPT(x', x_{current}) using RTT criterion **then**
        $x_{\text{current}} \leftarrow x'$
        if $f(x') < f(x^*)$ **then**
            $x^* \leftarrow x'$
        end if
    end if
    Update weights: $w \leftarrow \theta w + (1 - \theta) s_j$
    if dynamic customers arrive **then**
        Add customers to x_{current}
        Reset RTT threshold
    end if
end while
return x^*

Table B.2.1: Notation for Adaptive Large Neighborhood Search Algorithm

Symbol	Description
x	Current solution (set of routes)
x^*	Best solution found during search
x'	Candidate solution generated by destroy-repair cycle
D	Set of destroy operators {random removal, string removal}
R	Set of repair operators {greedy insertion}
d	Selected destroy operator from D
r	Selected repair operator from R
w_d, w_r	Adaptive weights for destroy and repair operators
$f(x)$	Objective function value (total cost) for solution x
θ	Weight decay parameter (0.8) controlling adaptation rate
s_j	Score assigned based on operator performance outcome
RTT	Record-to-Record Travel acceptance criterion

B.3 Soft Actor-Critic with Graph Attention Networks

Algorithm 3 Soft Actor-Critic with GAT for DVRP

Require: Environment $\mathcal{E}$, policy π_ϕ, Q-functions $Q_{\theta_1}, Q_{\theta_2}$
Ensure: Trained policy π_ϕ^*

Initialize replay buffer $\mathcal{B}$, target networks $\bar{Q}_{\theta_1}, \bar{Q}_{\theta_2}$
for each episode **do**
  Observe state s_0 from environment
  for each time step t **do**
    Extract features: $h = \text{GAT}(\text{embed}(s_t))$
    Sample action: $a_t \sim \pi_\phi(\cdot|h)$
    Map to discrete: $a_t^{\text{discrete}} = \text{ActionWrapper}(a_t)$
    Execute a_t^{discrete}, observe r_t, s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$
    if dynamic customer arrives **then**
      Update state s_{t+1} with new customer
    end if
    if ready to update **then**
      Sample batch from $\mathcal{B}$
      Update Q-functions using soft Bellman equation
      Update policy using reparameterization trick
      Update target networks: $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$
    end if
  end for
end for
return π_ϕ^*

Table B.3.1: Notation for Soft Actor-Critic with GAT

Symbol	Description
$\mathcal{E}$	DVRP environment implementing OpenAI Gym interface
π_ϕ	Actor network (policy) with parameters ϕ
$Q_{\theta_1}, Q_{\theta_2}$	Twin critic networks with parameters θ_1, θ_2
$\bar{Q}$	Target critic networks for stable training
$\mathcal{B}$	Experience replay buffer storing transitions
s_t	State at time step t (customer locations, demands, status)
a_t	Continuous action sampled from policy in range $[-1, 1]$
a_t^{discrete}	Discrete customer selection mapped from continuous action
r_t	Reward signal (negative tour length) at time step t
h	Feature representation extracted by GAT layers
GAT()	Graph Attention Network with multi-head attention
embed()	Linear embedding layer for state features
τ	Target network soft update rate (0.005)
π_ϕ^*	Optimal trained policy after convergence

C BENCHMARK SOLUTIONS

Table C.0.1: PyVRP Benchmark Solutions

Clustered (C)			Random (R)			Random-Clustered (RC)		
Instance	**Cost**	**Vehicles**	**Instance**	**Cost**	**Vehicles**	**Instance**	**Cost**	**Vehicles**
c1_25	330.00	3	r1_25	429.00	4	rc1_25	441.00	4
c1_50	595.00	5	r1_50	713.00	6	rc1_50	760.00	6
c1_100	1295.00	10	r1_100	1189.00	11	rc1_100	1401.00	11
c2_25	238.00	3	r2_25	358.00	3	rc2_25	271.00	3
c2_50	430.00	3	r2_50	500.00	4	rc2_50	413.00	4
c2_100	703.00	3	r2_100	718.00	4	rc2_100	734.00	4