Meta-Reinforcement Learning for Dynamic Parameter Adaptation in Multi-Round Vehicle Routing

Bojan Dimovski
Student 4th year, Computer Technologies and Engineering
Faculty of electrical engineering and information technologies
Skopje, Macedonia
kti252022@feit.ukim.edu.mk

***Abstract*—This research investigates the optimization of the Vehicle Routing Problem (VRP) by implementing an Adaptive Genetic Algorithm (GA) controlled by a Reinforcement Learning (RL) "Meta-Strategist." Traditional Genetic Algorithms often rely on static parameters, such as mutation rates and offspring sizes, which fail to adapt to the evolving state of a population during the search process. To address this, we developed a hierarchical framework where a Proximal Policy Optimization (PPO) agent, implemented via the Stable-Baselines3 library, dynamically adjusts GA parameters in real-time. The agent observes an 8-dimensional state vector, including population diversity and fitness improvement rates, to select optimal configurations for mutation rates (0.1 to 0.5) and offspring sizes (200 to 600) every 20 generations.**

Experimental results demonstrate that the RL-powered Adaptive GA significantly outperforms a Vanilla GA baseline, achieving superior fitness scores in 92.59% of tested instances and maintaining a 100% instance-level feasibility rate. Statistical analysis via the Wilcoxon Signed-Rank test confirmed that these improvements are significant, with a median fitness score of 92.65 compared to the baseline. However, the study also highlights substantial challenges, particularly the high computational overhead and training time required for the RL agent. While the RL agent effectively manages the exploration-exploitation trade-off by responding to population stagnation, the complexity of the neural network policy may be excessive for smaller problem instances. We conclude by suggesting that future research focus on lightweight, rule-based adaptive mechanisms that replicate the RL agent's successful heuristics, such as increasing mutation during diversity loss, to provide the benefits of dynamic adaptation without the prohibitive temporal costs of reinforcement learning.

I. Introduction

To establish the framework of this study, we first define the Static Capacitated Vehicle Routing Problem (SCVRP). We represent an instance as a septuple $\mathcal{P} = \langle V, E, c, d, Q, K, \mathcal{C} \rangle$, where:

- $V = \{v_0, v_1, ..., v_n\}$ is the set of vertices, where v_0 denotes the central depot and $V \setminus \{v_0\}$ represents the customers.
- $E \subseteq V \times V$ is the set of edges connecting vertices.
- $c : E \to \mathbb{R}^+$ defines the edge weights, representing travel costs or distances.
- $d : V \setminus \{v_0\} \to \mathbb{R}^+$ maps each customer to their specific demand.
- Q is the uniform capacity constraint for each vehicle.
- K is the number of available vehicles.
- $\mathcal{C}$ is the objective function to be minimized, typically representing total distance or operational cost.

While traditional benchmarks, such as those in [1], assume the number of required routes is less than or equal to K, we extend this model by introducing Dispatch Rounds. We define the number of rounds as $\left\lceil \frac{\#\text{routes}}{\#\text{vehicles}} \right\rceil$. This modification allows vehicles to perform multiple trips, transforming the optimization challenge into a multi-phase scheduling problem.

II. Metrics

Since the benchmark instances in [1] provide known optimal solutions, candidate solutions can be evaluated relative to these reference values. The evaluation framework is constructed so that all fitness values are bounded above by 100, which simplifies comparison across instances and experimental settings. Here are the key components of which the fitness function which we aim to optimize is a linear combination of a few components.

A. Relative cost gap

Let $C(\sigma)$ denote the total routing cost of a candidate solution σ, and let C^* denote the known optimal cost of the corresponding instance. The relative cost gap is defined as the percentage increase over the optimal solution:

$$\text{gap\_cost}(\sigma) = \text{gc}(\sigma) = \left(\frac{C(\sigma) - C^*}{C^*} \right) \cdot 100 \qquad (1)$$

This way, the relative cost gap is a number in the interval $[0, \infty)$ how much worse (percentage-wise) the cost of the tuple of routes found by σ is, compared to the optimal solution.

- If $\text{gap\_cost}(\sigma) = 0$, then the solution (represented by the list of routes that the GA found) has the same cost (or distance travelled) as the optimal solution.
- If $\text{gap\_cost}(\sigma) \in (0, 100)$ then the solution found by the GA is suboptimal

- If $\text{gap\_cost}(\sigma) = 100$, then the cost (sum of weights of all the routes in the list of routes) is twice the cost of the optimal solution
- If $\text{gap\_cost}(\sigma) > 100$, the sum of the total lengths of the routes is more than double the cost of the optimal solution

B. Relative dispatch round increase

Let $R(\sigma)$ denote the number of dispatch rounds required by solution σ, as defined in the previous section. If the instance provides the optimal set of routes, the corresponding optimal number of dispatch rounds R^* is computed using the same definition. The relative increase in dispatch rounds is given by

$$\text{gap\_dispatch}(\sigma) = \text{gd}(\sigma) = \left(\frac{R(\sigma) - R^*}{R^*}\right) \cdot 100 \quad (2)$$

The interpretation of the values of gap_dispatch is similar to gap_cost:

- $\text{gap\_dispatch}(\sigma) = 0$ means σ represents a solution that takes as many dispatch rounds as the optimal (so the least amount of dispatch rounds possible)
- $\text{gap\_dispatch}(\sigma) = 100$ means that σ represents a solution that takes twice as many dispatch rounds as the optimal solution would.

C. Penalties due to constraint violations

Here, we treat the optimization problem at hand as a problem with s.c. *soft* constraints, meaning that infeasible solutions are **not** discarded, as they can mutate or recombine with other candidate solutions to provide more optimal ones, but rather, penalized with a certain penalty. This means that if a constraint is violated, i.e. a vehicle picks up more than it can carry, the total value of the fitness function will decrease by a penalty proportional to the number of violations.

$$\text{penalty}(\sigma) = [\#\text{violations}] \cdot \text{penalty\_factor} \quad (3)$$

This means that not all infeasible solutions are penalized equally. A solution in which only one route has an accumulated pick-up weight more than Q won't be penalized as much as a solution in which all routes violate the uniform capacity constraint of the vehicles. For our algorithm, we define a penalty factor of 200.

D. Fitness function

Combining (1), (2), and (3), even though the `metrics` module reports all metrics, the final fitness function which the GA aims to optimize is calculated according to the following formula:

$$\text{fit}(\sigma) = 100 - [w_c \ \text{gc}(\sigma) + (1 - w_c)\text{gd}(\sigma) + \text{p}(\sigma)] \quad (4)$$

Where $w_c \in (0, 1)$ is the weight of the relative cost gap in the final fitness function. For our purposes, we have defined $w_c = 0.7$. Defined as such, $\text{fit}(\sigma)$ can have values in the range $(-\infty, 100]$. Since it is a metric that can be unintuitive, an analogy can be made with $R^2\%$.

- If $\text{fit}(\sigma) = 100$, then σ is the optimal solution, just like $R^2 = 100\%$ represents a perfect fit of the curve.
- If $\text{fit}(\sigma) \in (0, 100)$ it means that σ is suboptimal, but the solution is not naive.
- If $\text{fit}(\sigma) = 0$, it means the solution σ leads to a solution which costs twice as much as the cost of the optimal solution and the vehicles travel twice as many dispatch rounds, i.e. σ is twice as worse as the optimal solution σ^*, i.e. a naive solution analogous to $R^2 = 0$ which would be tantamount to a dummy regressor predicting the mean for any input.
- If $\text{fit}(\sigma) < 0$ it means the solution σ is not feasible or is so bad that it performs worse than the naive solution which is twice as worse, analogous to $R^2 < 0$ meaning a regressor performing worse than a naive dummy regressor with a mean strategy.

III. Defining the Vanilla Genetic Algorithm

The Genetic Algorithm (GA) serves as the fundamental search engine for solving the Vehicle Routing Problem (VRP). It operates by mimicking biological evolution, where a population of candidate solutions is iteratively improved through selection and variation.

A. Evolutionary cycle

The EA follows a structured loop that continues for a set number of generations or until early stopping is triggered:

1) Initialization: A population of μ individuals is created. Each individual represents a potential delivery route sequence.
2) Parent Selection: Two parents are chosen from the population using tournament selection, where a small group is compared and the best performer wins the right to reproduce.
3) Variation (Crossover & Mutation):

3.1. Crossover: The parents exchange genetic material via ordered crossover to produce two offspring.

3.2. Mutation: Each offspring has a chance (defined by the mutation_rate) to undergo a random structural change to maintain diversity.

4) Fitness Evaluation: The cost of each new solution is calculated. If a solution violates vehicle capacity constraints, it receives a heavy fitness penalty.
5) Environmental Selection: The algorithm determines which individuals survive to the next generation. Under "Plus Selection," the best individuals are chosen from the combined pool of parents and offspring $(\mu + \lambda)$ in total, to ensure the best solutions are never lost.

B. Mutation operators

To prevent the algorithm from getting stuck in local optima, we employ four distinct mutation types. Each mutation has a specific probability of being chosen: 0.4 for inversion, 0.3 for swap, 0.2 for scramble, and 0.1 for displacement.

- Inversion Mutation (40%): A segment of the route is selected and its order is completely reversed. This is highly effective at untangling "crossed" paths in a route.
- Swap Mutation (30%): Two random delivery points (customers) in the sequence are swapped. This allows for small, surgical adjustments to the schedule.

- Scramble Mutation (20%): A random segment of the route is chosen, and the customers within that segment are shuffled into a completely random order.
- Displacement Mutation (10%): A sub-sequence of the route is removed and re-inserted into a different random position. This allows the algorithm to move entire blocks of deliveries to different times in the day.

IV. Performance of Vanilla Genetic Algorithm

Given the difficult nature of the problem, the Vanilla GA with fixed heuristics performed quite well. The experiment consisted of 5 runs, each with different seedings, for each of the 27 VRP instances. Parameters were configured globally. A $(\mu + \lambda)$ strategy was deployed and the following values were chosen: $\mu = 800, \lambda = 400, \text{n\_gen} = 1000, \text{mut\_rate=}0.3$. As expected, for *simpler* VRPs (less than 40 locations and no more than 6 vehicles), the Vanilla GA performs quite well. Solutions found for the more challenging VRPs, however, tend to be suboptimal. In one instance, the Vanilla GA even failed to find a feasible solution.

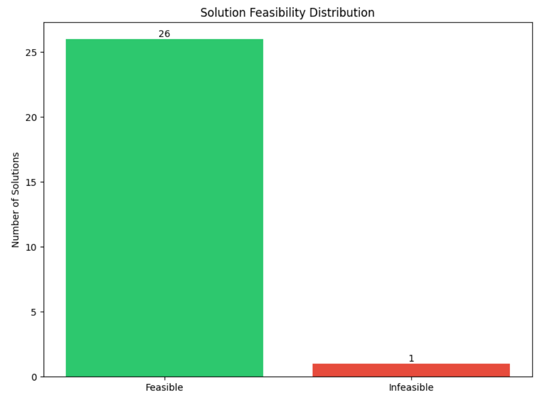

Fig. 1. Number of feasible solutions found across all instances by Vanillla GA

Fig. 1 shows the feasibility distribution of the best solutions found for each of the instances by the Vanilla GA. As explained before, for one instance, specifically the instance `A-n61-k9` (meaning the instance consisted of 61 locations and 9 vehicles), the GA did not manage to find a feasible solution resulting in a negative value for the fitness function. For all other instances the GA found solutions, most of which were suboptimal, but all of which are better than a "naive" solution (with fitness 0 - twice as worse as the optimal one).

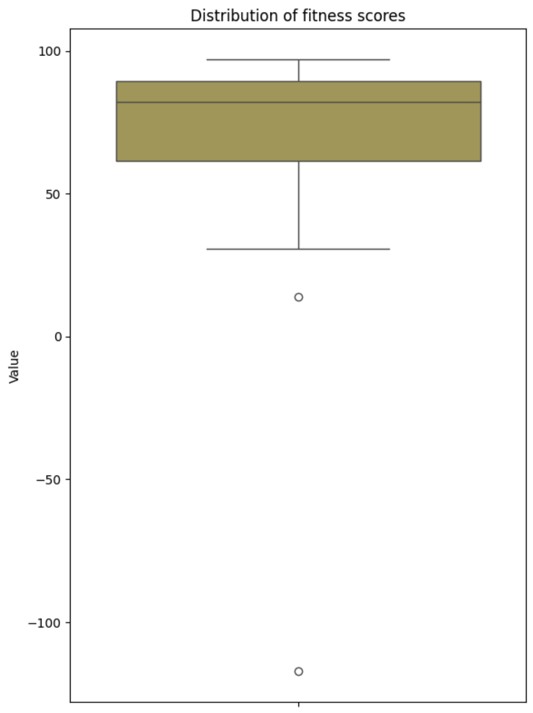

Fig. 2. Distribution of best fitnesses found by Vanilla GA across all instances

Fig. 2 shows the distribution of the best candidate solutions' fitness scores for the 27 solutions. Only one solution is infeasible, hence outside the range $(0, 100)$ - so it is only natural that it's the outlier. Another outlier is solution that's quite suboptimal with a fitness score just over 0. The mean value of all the fitness scores is 67.09. However, the instances vary in difficulty and the fitness scores are not normally distributed whatsoever, so a more reliable metric would be the median best fitness score per instance, which has a satisfactory value of 82.02. Another advantage of using the median score here is that it represents a specific instance with a specific candidate solution, meaning the solution can be visualized. Here, the instance whose best found fitness score was 82.02 was actually the instance `A-n38-k5`. Surprisingly, the GA scored better than the median for some VRPs more complex than the one that the GA didn't find a feasible solution for.

Plotted below is graph with all the routes found by the GA (each route is a different shade of red)

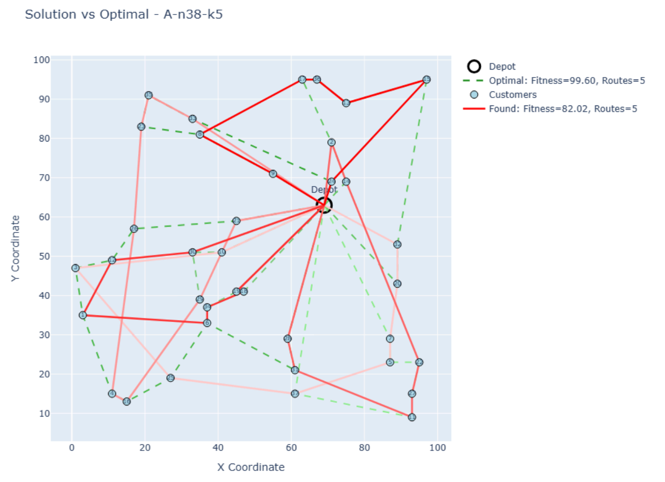

Fig. 3. Routes found by GA vs. routes from optimal solution of `A-n38-k5`

Fig. 3 clearly demonstrates that even though the routes found by the GA are not the optimal ones, the GA has still figured out routes that are not entangled, meaning that it performs quite well for this instance. This shouldn't be a surprise since it's a 'simple' VRP. Here, 'simple' refers to the property that the VRP has less than 40 locations and no more than 6 vehicles - since GA's, like almost any meta-heuristics, suffer from the curse of dimensionality. A quite interesting observation is that when we focus on the solutions of the Vanilla GA on the problems that we dubbed 'simple', performance drastically increases, with the average best-found fitness score across all instances being 90.04.

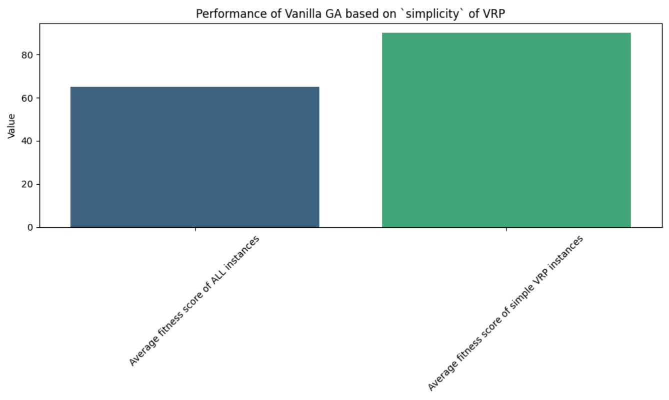

Fig. 4. Performance comparison of GA on all problems vs. 'Simple' problems

Fig. 4 compares the average best-found fitness score across all instances vs. the average best-found fitness score across the 'simple' instances.

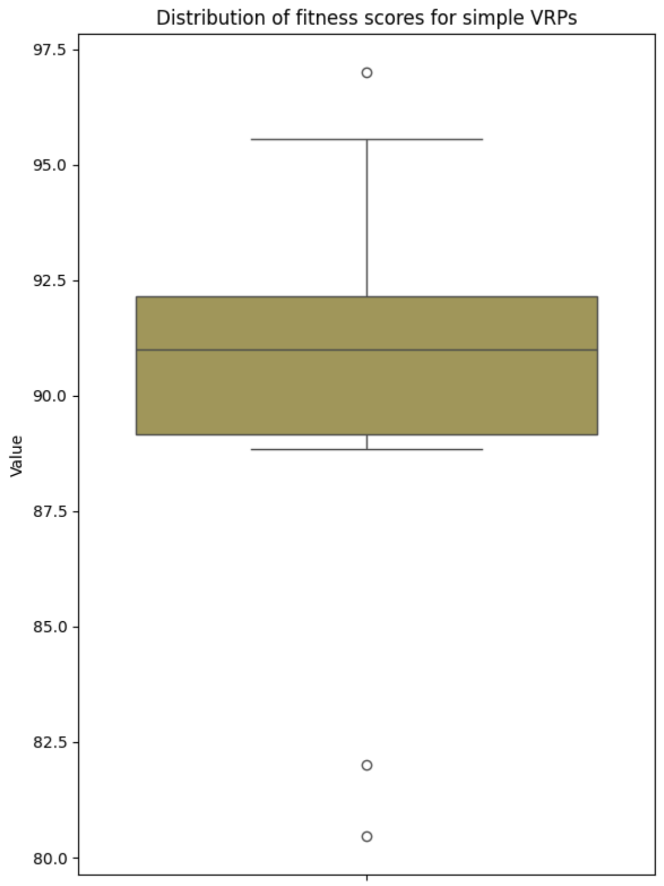

Fig. 5. Performance of Vanilla Genetic Algorithm on 'Simple' problems only

Fig. 5 represents a box-plot of the distribution of the fitness scores for the simple VRPs.

V. RL Agent as a meta-strategist

While the Vanilla GA utilizes static parameters, the introduction of a Reinforcement Learning (RL) agent transforms the algorithm into an adaptive system [2]. The RL agent acts as a "Meta-Strategist," monitoring the state of the evolutionary search in real-time and dynamically adjusting the genetic operators to optimize performance.

A. The Adaptive Control Loop and State Space

The RL agent operates within a specialized Gymnasium environment (`AdaptiveEAEnv`) that wraps the core Evolutionary Algorithm. Every 20 generations (defined as `n_check_gen`), the EA pauses to allow the Meta-Strategist to observe the current population and make adjustments.

The agent observes a state vector $s_t \in \mathbb{R}^8$:

$$s_t = \left[f_{\text{best}}, f_{\text{avg}}, \Delta f, \text{div}, \text{prog}, \mu, \lambda_{\texttt{norm}}, \text{step}_{\text{check}}\right] \quad (5)$$

Where:

- f_{best} and f_{avg}: Normalized best and average fitness.
- Δf: Recent improvement rate $\frac{f_t - f_{t-1}}{100}$.
- div: Population diversity measured as the standard deviation of fitness.
- prog: Temporal progress $\frac{\text{gen}}{\text{gen}_{\text{max}}}$.

B. *Policy Optimization via Stable-Baselines3*

To manage this environment, we utilize the Proximal Policy Optimization (PPO) algorithm as implemented in the Stable-Baselines3 library [3]. Using a high-level framework like Stable-Baselines3 ensures a robust, peer-reviewed implementation of PPO's complex clipping and advantage estimation mechanisms, allowing focus on the VRP-specific environment logic rather than low-level gradient handling.

The agent optimizes a surrogate objective function to prevent updates from being too large, which maintains training stability:

$$L_C(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big)\Big] \quad (6)$$

Where:

- $r_t(\theta)$ is the probability ratio between the new and old policy.
- $\hat{A}_t$ is the estimated advantage, calculated using Generalized Advantage Estimation (GAE) with $\lambda = 0.95$ and $\gamma = 0.99$.
- $\varepsilon = 0.2$ is the clipping parameter that restricts the policy change.

The implementation uses an `MlpPolicy` with two separate networks (`Actor` and `Critic`), each consisting of two hidden layers of 256 neurons and `ReLU` activation functions.

C. *Reward Engineering and Strategy*

The agent is trained to maximize a cumulative reward signal R_t. The reward function is designed to penalize stagnation and favor efficiency:

$$R_t = \frac{5 \cdot \Delta f}{100 - f_{t-1}} + \frac{f_{\text{best}}}{100} + \text{Bonuses} - \text{Penalties} \quad (7)$$

- Improvement Bonus: Scaled by the remaining gap to the optimal solution.
- Feasibility Penalty: A −1.0 deduction if more than 50% of the population is infeasible, forcing the agent to learn "safe" parameter ranges.
- Diversity Penalty: A −0.5 deduction if diversity drops below 0.05, preventing the agent from collapsing the population into a local optimum.This mathematical framework enables the RL Meta-Strategist to handle the "exploration-exploitation" trade-off more effectively than any static configuration.

VI. Performance of RL-powered Evolutionary Algorithm

The integration of the PPO-based adaptive controller yielded some improvements in the robustness and convergence of the Genetic Algorithm. The following analysis evaluates the performance of the RL-powered GA across the problem suite, focusing on its ability to navigate complex search spaces through dynamic parameter modulation.

The global performance metrics indicate a high level of consistency in finding high-quality solutions. As illustrated in the fitness distribution below, the RL-tuned approach maintains a tight interquartile range, suggesting that the "Meta-Strategist" successfully mitigates the risk of poor runs by intervening during population stagnation.

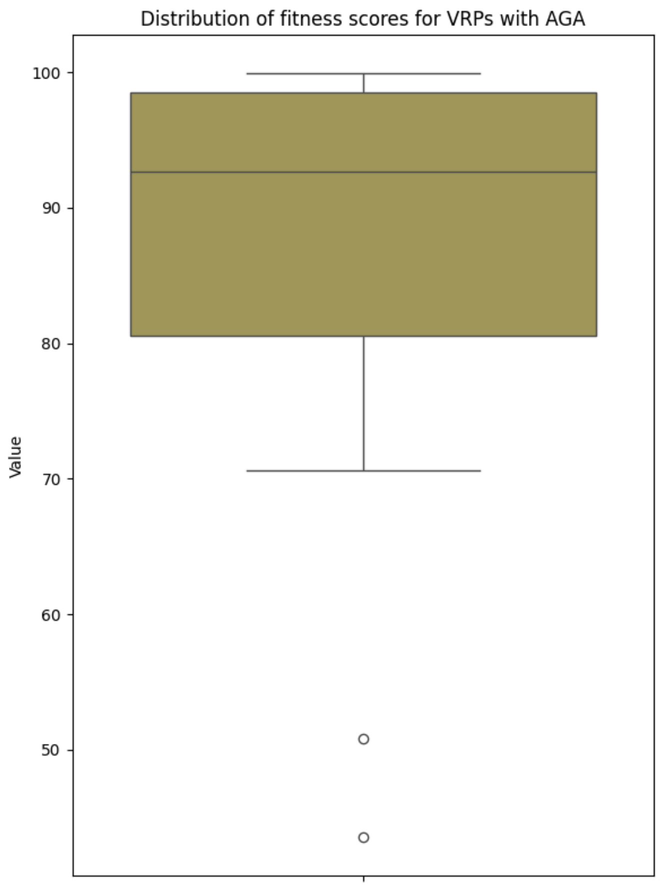

Fig. 6. Fitness distribution for solutions of GenAlg with adaptive parameters

Key statistical observations from the experimental results include:

- Mean Performance: The mean best-found fitness score across all problem instances is 86.85. This high average reflects the agent's ability to maintain performance even on instances where traditional static GAs struggle.
- Median Performance: The median best-found fitness score is 92.65. The fact that the median is significantly higher than the mean indicates a distribution skewed toward high-quality solutions, with the majority of runs achieving near-optimal fitness.
- Diversity Management: The policy's adherence to the diversity penalty (rewarded for staying above 0.05) prevented the premature convergence that typically lowers the mean in static GAs.

One of the primary objectives of the RL Meta-Strategist was to maximize the feasibility rate of the final solutions, particularly given the constrained nature of the VRP.

Instance Success Rate: The adaptive GA successfully found at least one feasible solution for every instance in the dataset, achieving a 100% (27/27) instance-level feasibility rate.

Reliability Metrics: While the algorithm consistently solved every instance, the reliability across individual runs provided deeper insight into the problem's difficulty. In the most challenging instance - the same instance for which the Vanilla GA failed to find any feasible solutions - the RL-powered GA achieved a feasibility rate of 80.0%.

Failure Analysis: In this specific "hard" instance, the model successfully found a feasible best solution in 4 out of 5 runs. This 80% success rate represents a substantial improvement over the baseline, as the RL agent learned to increase the offspring size (λ) and mutation rate to explore the feasible region more aggressively when the population moved toward a constrained boundary.

Plotted below is the routes found by this RL-powered adaptive GA for the same instance from Fig. 3, i.e. instance `A-n38-k5` for which the best-found fitness score improved from 82.02 to 82.33. Interestingly enough, the only difference in the routes is one swap of two vertices (specifically locations numbered 18 and 19) in a single route from all the routes.

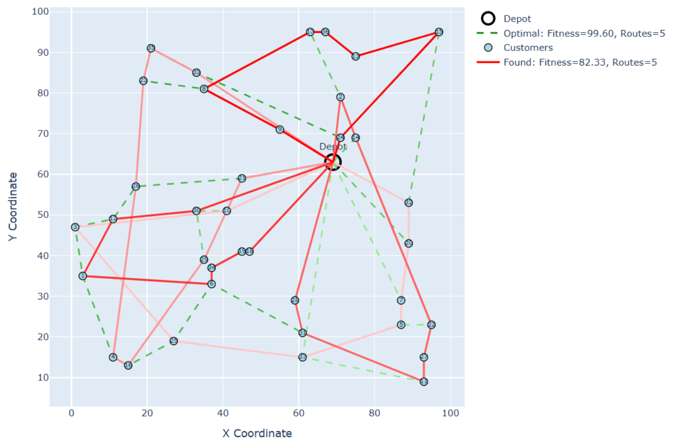

Fig. 7. Routes found by RL-GA vs. optimal routes for instance `A-n38-k5`

VII. Comparison

The performance comparison between the Vanilla Genetic Algorithm (VGA) and the RL-powered Adaptive Genetic Algorithm (RLGA) reveals a clear dominance of the meta-strategy approach across the majority of the problem suite. As indicated by the data, the RLGA achieved superior fitness scores in 25 out of 27 instances, representing a 92.59% success rate in outperforming the static baseline. The two instances in which the vanilla GA performed better are A-n55-k9 and A-n80-k10, which are high-dimensional complex problems. It is hypothesized that in these specific scenarios, the RL agent may have encountered "policy chatter" or sub-optimal adaptation when faced with the extreme state-space complexity of these larger instances, leading to less efficient parameter tuning compared to the stable, albeit rigid, baseline. However, as illustrated in Fig. 8, even in instances where the VGA outperforms the RL-powered Adaptive GA, it does so only slightly, suggesting that the RLGA's learned policy remains competitive even when not optimal.

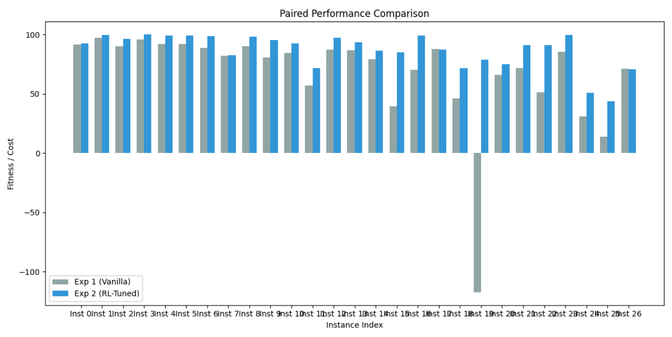

Fig. 8. Bar-plot of pairs of best-found fitness scores per Genetic Algorithm

This marginal difference in the few cases of VGA superiority is overshadowed by the substantial gains the RLGA provides across the rest of the dataset. This is further confirmed in Fig. 9, where the per-instance fitness difference is plotted, showing a consistent upward trend in solution quality for the adaptive variant. The RL agent's ability to observe the normalized best fitness, average fitness, and population diversity every 20 generations allows it to proactively "rescue" the GA from local optima that typically trap a static configuration. By dynamically adjusting the mutation rate between 0.1 and 0.5 and the offspring size (λ) between 200 and 600, the RL agent effectively manages the exploration-exploitation trade-off in real-time. For example, when the agent detects a drop in diversity - defined in the environment as the standard deviation of fitnesses - it receives a penalty and subsequently learns to select actions that increase the mutation rate to force the population into new regions of the search space.

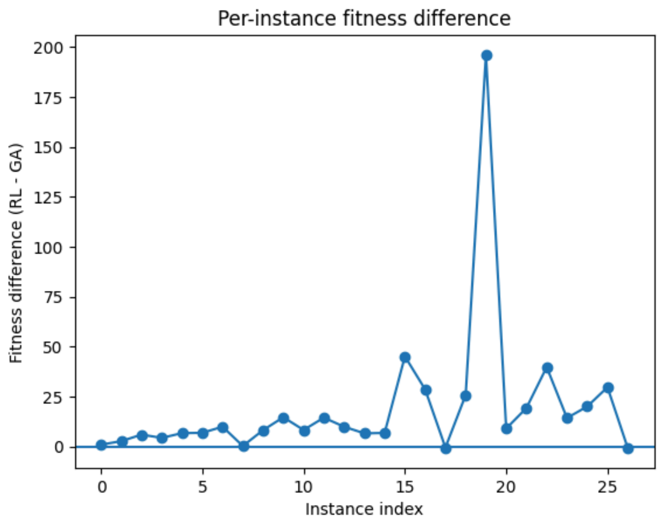

Fig. 9. Per-instance difference of best-found fitness scores by RLGA and VGA

To determine whether the difference between best-found fitness scores is statistically significant [4], a Wilcoxon Signed-Rank test was performed on the paired results from all 27 instances. This non-parametric test was chosen because it does not assume a normal distribution of fitness scores and is robust for comparing two related samples on a single set of test problems. The resulting p-value was significantly lower than the standard 0.05 threshold specifically it had a value of $p = 1.49 \cdot 10^{-7}$, providing strong statistical evidence that the RL-powered Adaptive GA's performance is not a product of random variation but a result of effective meta-strategy learning. The success of the RLGA is largely attributed to the stable training provided by the PPO implementation in Stable-Baselines3, which utilized a clipped surrogate objective and Generalized Advantage Estimation (GAE) to refine the policy over 10,000 timesteps. By treating GA parameters as a dynamic control problem rather than a static optimization problem, the Meta-Strategist allows the algorithm to adapt to the specific "topology" of each VRP instance, leading to the high median fitness of 92.65 and a 100% instance-level feasibility rate.

VIII. Concluding Remarks

Despite the statistically significant improvements demonstrated by the RL-powered variant, the integration of a Deep Reinforcement Learning agent as a meta-strategist presents substantial practical challenges, primarily regarding computational overhead and training complexity. The temporal cost of training the PPO agent over 10,000 timesteps is considerable, with the logs indicating that even a 30% progress mark requires significant real-world time. While the RL agent successfully learned to manage the exploration-exploitation trade-off by observing the population diversity and improvement rates, the performance gains - though measurable - do not always justify the resource intensity when compared to a well-tuned Vanilla GA. In high-dimensional instances such as A-n55-k9 and A-n80-k10, the added complexity of the RL policy occasionally resulted in sub-optimal parameter adaptation, suggesting that the agent may struggle to generalize across extremely varied search landscapes.Looking toward future implementations, a more pragmatic approach may lie in the development of an Adaptive GA that utilizes simple, deterministic rule-based logic rather than a heavy neural network-based controller. Such a system could replicate the RL agent's successful strategies through transparent heuristics: for example, if the population diversity falls below a critical threshold (e.g., 0.05), the mutation rate could be automatically incremented to 0.5 to force exploration. Similarly, if the improvement rate Δf stagnates over a set number of generations, the offspring size (λ) could be scaled to increase selection pressure. By stripping away the RL training phase, researchers can maintain the benefits of a dynamic parameter landscape - responding to the "health" of the population in real-time - without the prohibitive time-to-convergence associated with training a Meta-Strategist. This transition toward lightweight rule-based adaptation would preserve the flexibility required for the Vehicle Routing Problem while ensuring the algorithm remains computationally accessible for industrial applications.

References

[1] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, D. Naddef, and G. Rinaldi, "Computational results with a branch and cut code for the capacitated vehicle routing problem," no. 949-M. 1995.

[2] M. Abdelatti and M. Sodhi, "Using Reinforcement Learning for Tuning Genetic Algorithms." p. , 2021. doi: 10.1145/3449726.3463203.

[3] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021, [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[4] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.