ML-Enhanced Large Neighborhood Search Project Report

Machine Learning for Optimization, WS 2025

Daniel Levin (12433760)

1 Problem: SCF-PDP

The **Selective Capacitated Fair Pickup and Delivery Problem (SCF-PDP)** is a vehicle routing problem where the goal is to design fair and feasible routes for a subset of customer requests. Each customer needs transportation of goods from a pickup location to a corresponding drop-off location.

The problem is modeled on a complete directed graph $G = (V, A)$ where:

- V contains the vehicle depot plus all pickup/drop-off locations
- $A = \{(u, v) : u, v \in V, u \neq v\}$ represents travel routes
- Arc distances $a_{u,v} = \lceil \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2} \rceil$ (rounded Euclidean)

There are n customer requests $CR = \{1, \ldots, n\}$, each defined by pickup location $v_i^{\uparrow}$ and drop-off location $v_i^{\downarrow}$ with demand c_i. A fleet K of n_K identical vehicles with capacity C serves requests starting from the depot.

A feasible solution assigns routes R_k to each vehicle $k \in K$ such that:

- Vehicle capacity never exceeded along any route
- Each served request handled entirely by one vehicle
- At least γ requests served across all vehicles

The objective minimizes total distance plus fairness penalty:

$$\sum_{k \in K} d(R_k) + \rho \cdot (1 - J(R))$$

where $d(R_k)$ is route k's total distance and $J(R)$ is the Jain fairness index:

$$J(R) = \frac{\left(\sum_{k \in K} d(R_k)\right)^2}{n_K \cdot \sum_{k \in K} d(R_k)^2}$$

The fairness index $J(R) \in (0, 1]$ equals 1 when all routes have equal distance. Parameter ρ controls the distance-fairness trade-off.

2 Baseline: ALNS

The baseline implementation uses Adaptive Large Neighborhood Search (ALNS) with simulated annealing acceptance, developed and tuned in the Heuristic Optimization Techniques course. This section summarizes the key components.

2.1 Operators

The implementation uses 3 destroy and 3 repair operators, yielding 9 combinations. Each destroy operator removes q requests sampled uniformly from $[10\%, 40\%]$ of served requests.

Destroy Operators:

- **Random**: Removes q randomly selected requests (unbiased exploration)
- **WorstCost**: Removes requests with highest distance contribution
- **LongestRoute**: Removes from longest routes (targets fairness)

Repair Operators:

- **Greedy**: Reinserts using flexible pickup-dropoff construction heuristic
- **RandomGreedy**: Reinserts at random feasible positions
- **ObjectiveAware**: Minimizes full objective including fairness penalty

2.2 Adaptive Weight Mechanism

Operators are selected via roulette wheel with adaptive weights. Scores: 10 for new best, 1 for accepted, 0 for rejected. Weights update every 100 iterations using:

$$\rho_i \leftarrow \rho_i \cdot (1 - \gamma) + \gamma \cdot \frac{s_i}{a_i}$$

where γ is the reaction factor, s_i total score, and a_i applications in the period.

2.3 Parameters

Parameter	Description	Default
`max_iterations`	Maximum iterations	10000
`max_time_seconds`	Runtime limit	300s
`weight_update_period`	Weight update frequency	100
`reaction_factor` (γ)	Adaptation speed	0.1
`min/max_removal_pct`	Removal range	10%-40%
`initial_temperature`	SA start temperature	100.0
`cooling_rate`	Temperature decay	0.99

Table 1: ALNS default parameters

3 Multi-Armed Bandit Operator Selection

The adaptive weight mechanism in baseline ALNS has several limitations: periodic updates introduce delayed feedback, purely proportional selection offers minimal exploration, and all operators are treated equally regardless of how well-tested they are. Multi-armed bandit (MAB) algorithms address these issues by providing explicit exploration mechanisms and immediate learning from each iteration [1, 2].

3.1 Method

Applying MAB to ALNS operator selection requires three key design decisions [1]: the definition of arms, the reward mechanism, and the learning policy.

3.1.1 Arms Definition

Each ALNS iteration requires selecting both a destroy and a repair operator. Three alternatives were considered for modeling this as a MAB problem:

1. **Destroy operators only**: Model only the 3 destroy operators as arms and use a fixed repair strategy. This approach appears in the BALANS paper [1], where repair is done via an exact MIP solver. However, we have 3 diverse repair operators that provide complementary search behavior, and fixing repair would waste this diversity. Additionally for the sake of clean comparison with the existing adaptive approach, it is beneficial to use the same operators and focus puerly on their selection strategies.

2. **Operator pairs**: Model all $3 \times 3 = 9$ destroy-repair combinations as individual arms. This captures potential synergies between specific operators (e.g., WorstCost + Greedy may work differently than Random + Greedy). However, 9 arms significantly slows learning compared to 6 arms, and interpreting which combinations work well becomes less clear.

3. **Independent selection** (chosen): Maintain two independent MAB instances—one for destroy operators (3 arms) and one for repair operators (3 arms). Each iteration selects from both independently. This explores the full 3×3 space with only 6 total arms, accelerating learning while matching baseline ALNS behavior.

The independent selection approach balances exploration efficiency with operator diversity.

3.1.2 Reward Mechanism

The reward function defines what constitutes "success" for an operator. Three options were considered:

1. **Binary (0/1)**: Reward of 1 if the iteration improves the current solution, 0 otherwise. This is simple and aligns naturally with Thompson Sampling's Beta-Bernoulli conjugate prior [2], but ignores the magnitude of improvement.

2. **Continuous (Δ objective)**: Reward equals the actual objective improvement ($f_{\text{old}} - f_{\text{new}}$). This captures improvement magnitude but requires different priors for Thompson Sampling (e.g., Normal-Gamma) and can be sensitive to problem scaling.

3. **Tiered (10/1/0)** (chosen): Reward of 10 if the iteration finds a new global best solution, 1 if it improves the current solution (accepted by simulated annealing), and 0 otherwise. This matches the baseline scoring system and emphasizes discovering high-quality solutions, not just local improvements.

For Thompson Sampling, which requires binary feedback for Beta distributions, tiered rewards are converted to binary: any positive reward counts as success.

3.1.3 Learning Policies

Two MAB algorithms are compared against the adaptive weight baseline:

UCB1 (Upper Confidence Bound): Selects the operator with the highest upper confidence bound:

$$\text{UCB}_i(t) = \bar{r}_i + c\sqrt{\frac{\ln t}{n_i}}$$

where $\bar{r}_i$ is the average reward for operator i, t is the total number of selections, n_i is the number of times operator i has been selected, and c is an exploration constant (default 1.0). The second term provides an explicit exploration bonus: operators that have been selected less frequently get higher confidence bounds, encouraging balanced exploration.

Thompson Sampling: Maintains a Beta distribution $\text{Beta}(\alpha_i, \beta_i)$ for each operator i, representing the posterior belief about its success probability. At each iteration, a sample $\theta_i \sim \text{Beta}(\alpha_i, \beta_i)$ is drawn for each operator, and the operator with the highest sample is selected. After observing the outcome, the distribution is updated: $\alpha_i \leftarrow \alpha_i + 1$ if the reward was positive (success), otherwise $\beta_i \leftarrow \beta_i + 1$. Thompson Sampling naturally balances exploration and exploitation through probabilistic sampling, and is known to perform well in non-stationary environments where operator effectiveness may change during the search [2].

Both algorithms update immediately after each iteration, providing faster adaptation than the baseline's periodic updates.

3.2 Research Questions

The experiments aim to answer:

- **RQ1**: Does MAB-based selection outperform adaptive weights in terms of solution quality?
- **RQ2**: Which MAB algorithm (UCB1 or Thompson Sampling) performs best?
- **RQ3**: How does operator selection evolve over time (exploration to exploitation)?

3.3 Results

Ran 120 instances (30 per size: 50, 100, 200, 500 requests) with 10k iterations each. Head-to-head comparisons on identical instances.

Adaptive vs Thompson (Table 2, Figure 1): Thompson wins 67-53 with 0.63% better total objective. Mixed results by size: Thompson wins at $n = 50$ ($p = 0.036$), Adaptive at $n = 100$ ($p = 0.045$), no significant difference for larger sizes. Slight edge to Thompson but the baseline stays competitive.

Table 2: Adaptive Weights vs Thompson Sampling by instance size

Size	Adaptive Obj	Thompson Obj	% Diff	Wins (A/T)	p-value
50	1510 ± 136	1483 ± 114	+1.6%	10/20	0.036*
100	3867 ± 272	3914 ± 261	−1.3%	20/10	0.045*
200	10732 ± 871	10659 ± 923	+0.7%	12/18	0.299
500	42024 ± 2572	41712 ± 2380	+0.7%	11/19	0.164
Total	1,743,960	1,733,022	+0.63%	53/67	—

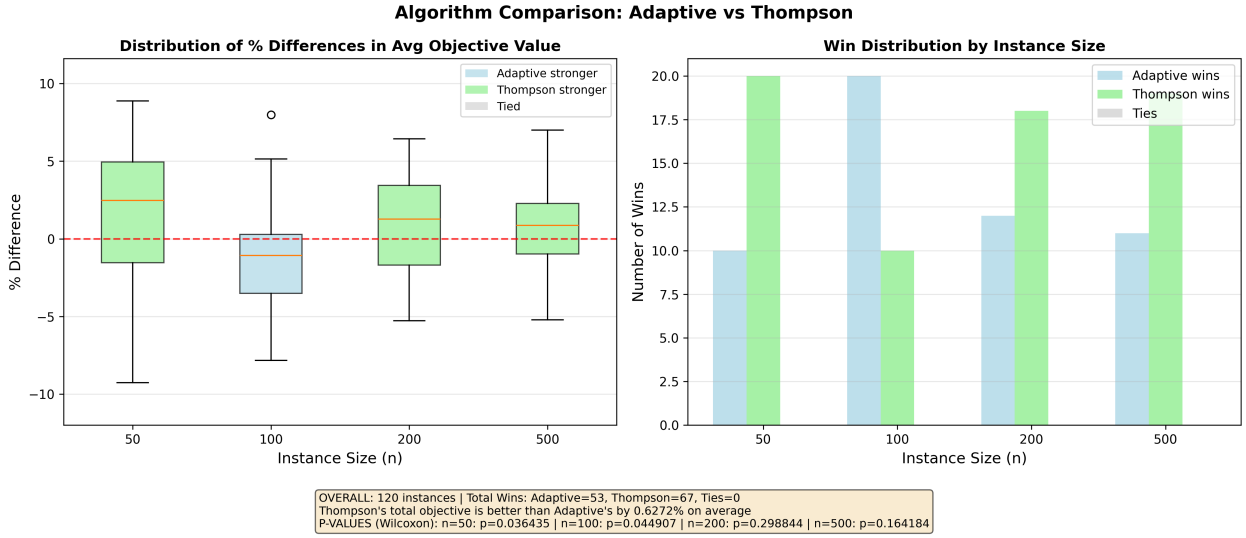

Figure 1: Adaptive vs Thompson: distribution of % differences (positive = Thompson better) and win counts

Adaptive Weights vs UCB1 (Figure 2): Adaptive dominates with 102 wins vs 18, achieving 7.6% lower total objective. The gap widens with instance size: from -0.9% at $n = 50$ (not significant) to -8.6% at $n = 500$ ($p < 10^{-9}$). UCB1's deterministic selection appears to over-exploit early successes, converging prematurely on suboptimal operators.

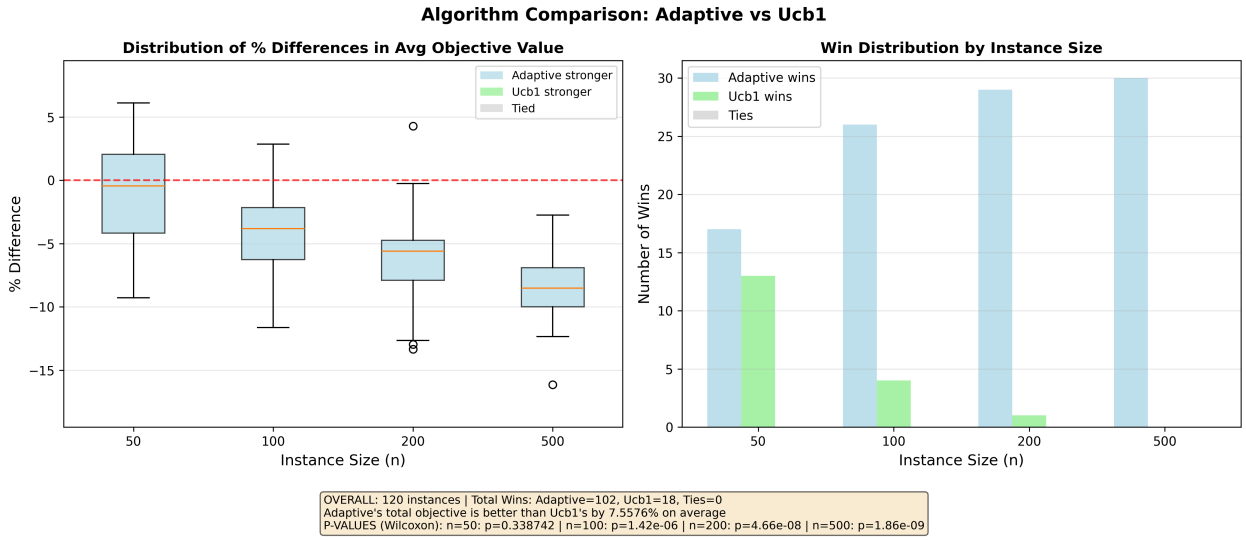

Figure 2: Adaptive vs UCB1: Adaptive significantly outperforms, especially on larger instances

UCB1 vs Thompson Sampling (Figure 3): Thompson wins 108 out of 120 instances with 7.5% better total objective. The advantage is significant at all sizes ($p < 0.007$) and grows from $+2.7\%$ at $n = 50$ to $+8.0\%$ at $n = 500$.

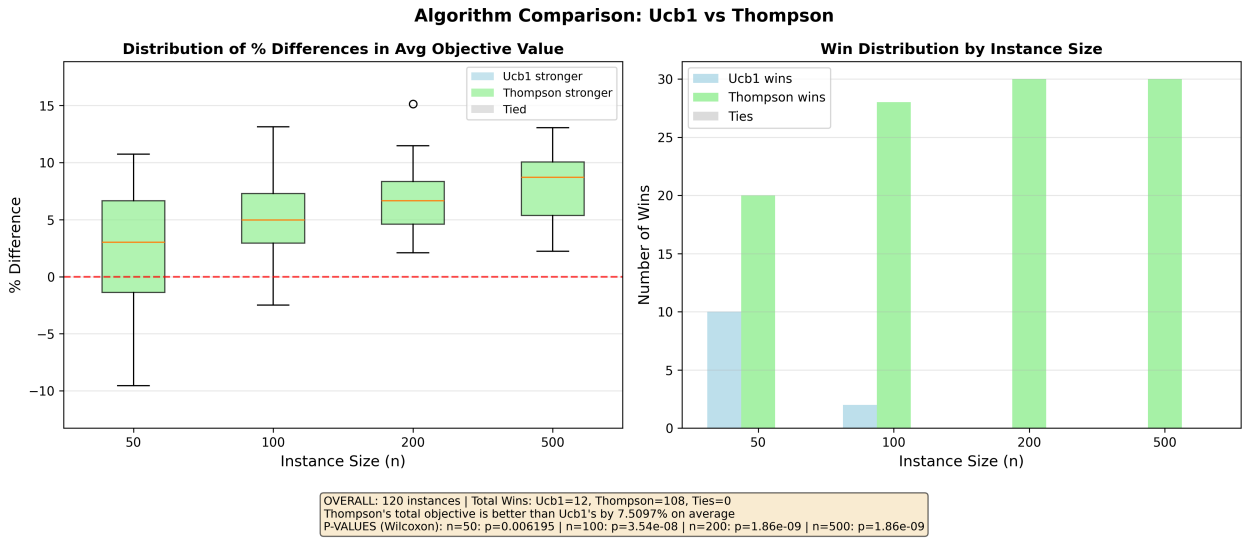

Figure 3: UCB1 vs Thompson: Thompson dominates across all instance sizes

Statistical significance assessed using Wilcoxon signed-rank test ($\alpha = 0.05$).

3.4 Case Study: Detailed Run Analysis

Looking at a single instance (`instance61_nreq100_nveh2_gamma91`, 500 iterations) reveals the behavioral differences.

Convergence (Figure 4): All three start at ∼4900 but diverge quickly. Thompson achieves the best result (4184.4, 15.0% improvement), Adaptive is close behind (4202.5, 14.6%), and UCB1 is last (4394.0, only 9.8%). All three seem to reach a plateau around 200th iteration, however both UCB1 and Adaptive manage to break out after 450th iteration which indicates the exploration is not exhausted yet. With more iterations we would probably see a similar picture where from time to time each operator selector manages to break out of plateaus. But in the end of the day Thompson seems to be the strongest one here as it converges to lower objective value early.

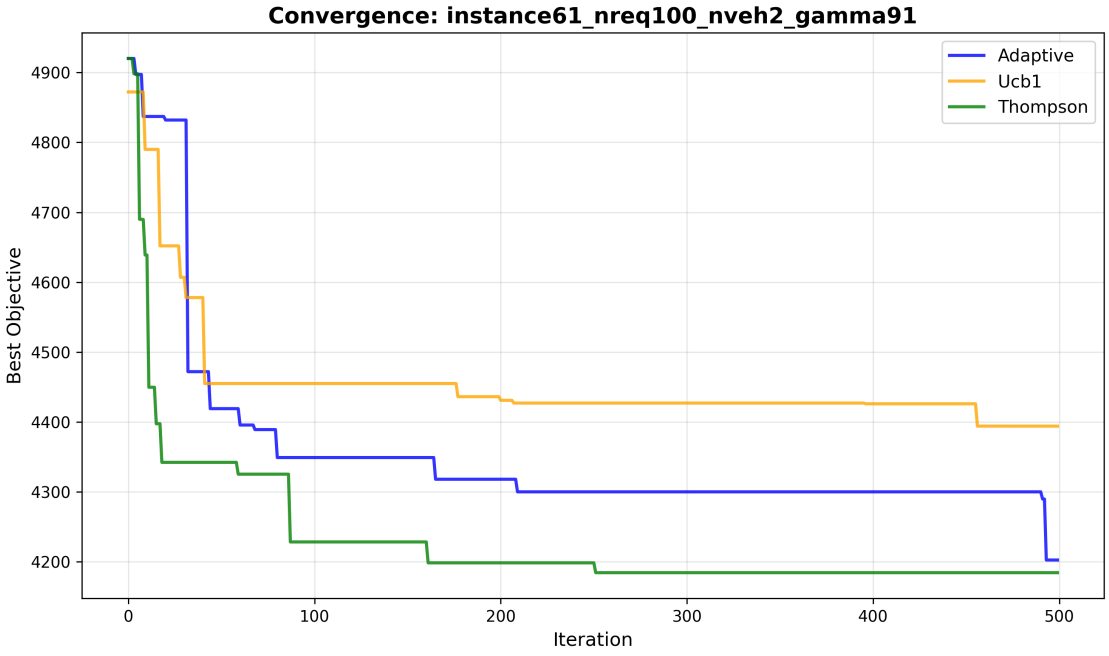

Figure 4: Convergence on instance61: Thompson continues improving while UCB1 plateaus early

Operator selection patterns:

Thompson (Figures 5, 6): Identifies the winning combination early and sticks with it: WorstCost (72%) + Greedy (87%). Frequencies stay stable throughout—no dramatic exploration-to-exploitation shift, just steady probabilistic sampling.

UCB1 (Figures 7, 8): Locks onto Random destroy (87%) while barely using WorstCost (6%)—seems to be the wrong choice for this problem. Thompson proves WorstCost is better, but UCB1's deterministic selection locked in after early luck. Gets repair right (87% Greedy) but the bad destroy choice kills performance.

Adaptive (Figures 9, 10): Maintains balanced diversity: destroy stays at 36%/33%/30%, shifting gradually over time. Repair adapts more: Greedy grows from 35% to 45%. This diversity explains why Adaptive is competitive—no catastrophic over-commitment like UCB1.

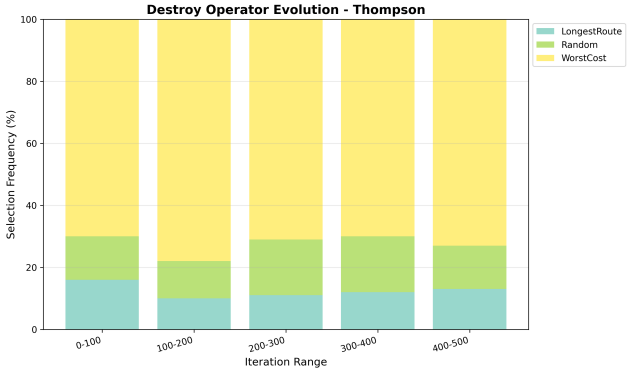

Figure 5: Thompson destroy operators

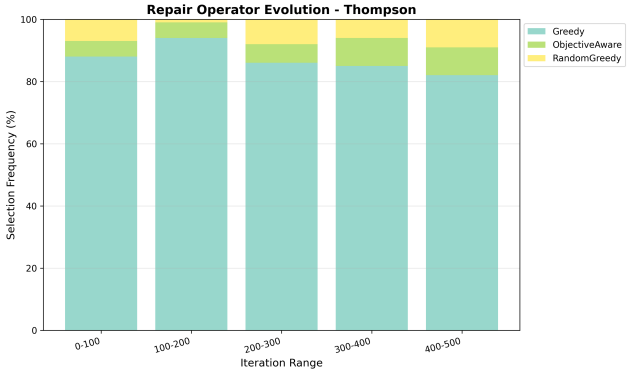

Figure 6: Thompson repair operators

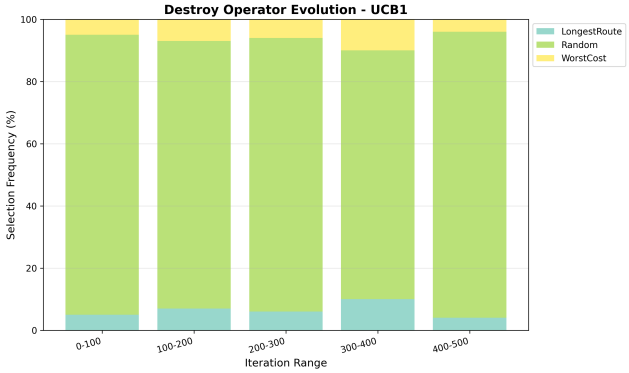

Figure 7: UCB1 destroy: locks onto Random

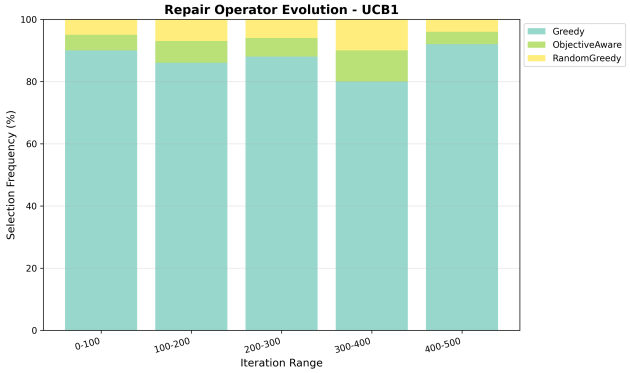

Figure 8: UCB1 repair operators

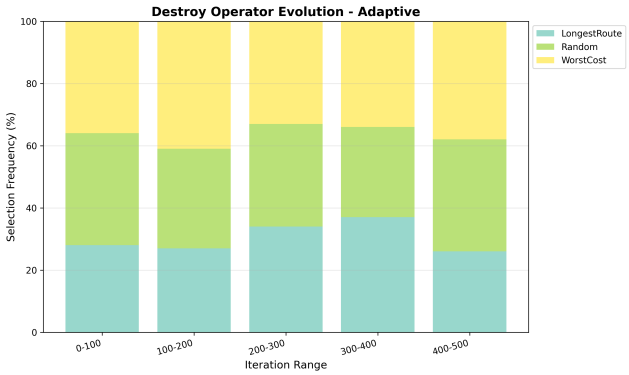

Figure 9: Adaptive destroy: balanced selection

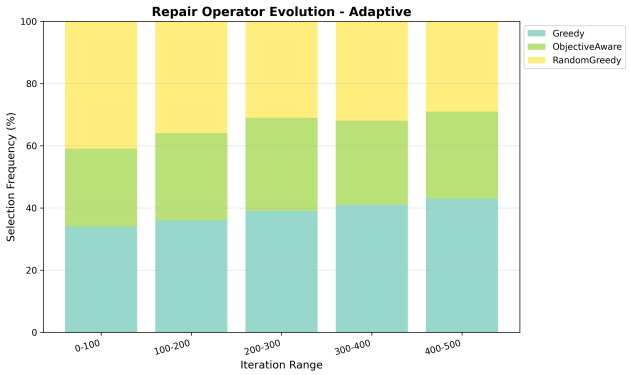

Figure 10: Adaptive repair: gradual shift

3.5 Discussion

Thompson Sampling wins overall (67-53 vs Adaptive, 108-12 vs UCB1) but the margin over the baseline is small (+0.6%). UCB1 performs significantly worse (-7.6% vs Adaptive) due to premature convergence—it locks onto Random destroy operator which happens to be the wrong choice. The baseline Adaptive Weights are surprisingly competitive because they maintain operator diversity (30-40% for each), avoiding UCB1's catastrophic over-exploitation.

Thompson works well because probabilistic sampling naturally balances exploration and exploitation. Even after identifying WorstCost as the best operator (72% selection), it keeps sampling alternatives (15% Random, 12% LongestRoute). This prevents UCB1-style lock-in while still exploiting knowledge.

For this problem, the simple baseline is hard to beat. Thompson provides marginal gains but adds complexity. If switching to MAB, only Thompson is viable—UCB1's deterministic selection is too greedy for long ALNS runs.

4 Reinforcement Learning Operator Selection

While MAB methods treat operator selection as a stateless multi-armed bandit, reinforcement learning can leverage search state to make context-aware decisions. Following Reijnen et al. [3], we implement Deep Q-Networks (DQN) to control both operator selection and ALNS parameters jointly.

4.1 Method

The DQN selector extends MAB by (1) incorporating search state and (2) jointly controlling destroy/repair operators plus severity and temperature multipliers [3]. The state representation captures 7 features: normalized objectives (current/initial, best/initial, current/best), search stagnation (iterations_without_improvement / max_iterations), temperature decay (current_temp / initial_temp), and recent performance metrics (acceptance rate, improvement rate over last 100 iterations).

The action space is discrete with 144 actions encoding all combinations: 3 destroy operators $\times$ 3 repair operators $\times$ 4 severity levels [0.1, 0.2, 0.3, 0.4] $\times$ 4 temperature multipliers [0.5, 1.0, 1.5, 2.0]. This joint control allows DQN to adapt neighborhood size and acceptance criteria dynamically based on search progress—something MAB cannot do.

The Q-network is a simple 2-layer MLP ($7 \to 64 \to 64 \to 144$) trained with standard DQN. We use DQN rather than PPO (the algorithm in [3]) for simplicity: DQN's discrete action handling and experience replay align well with our 144-action space and limited training data, while requiring less implementation complexity than actor-critic methods. Training uses experience replay (10k buffer, batch size 64), target network updated every 100 steps, discount $\gamma = 0.99$, and Adam optimizer (lr=0.001). Epsilon-greedy exploration decays from 1.0 to 0.1 over training. The reward signal is inherited from ALNS: solution improvement normalized by initial objective.

Training uses sequential episodes across all instances from sizes 50 and 100 (training set). Each instance runs for 500 iterations (max 100s), constituting one episode. The shared DQN accumulates experience across all episodes, learning a general policy rather than per-instance strategies.

4.2 Results

The policy was trained on 60 episodes (30 instances each from sizes 50 and 100) with 500 iterations per episode. Training achieved 19.8% average improvement with epsilon decaying smoothly to 0.1. Performance remained consistent across episode sizes: 20.8% for $n = 50$ and 18.8% for $n = 100$ (Figure 11), suggesting the network learned generalizable patterns rather than overfitting to specific instances.

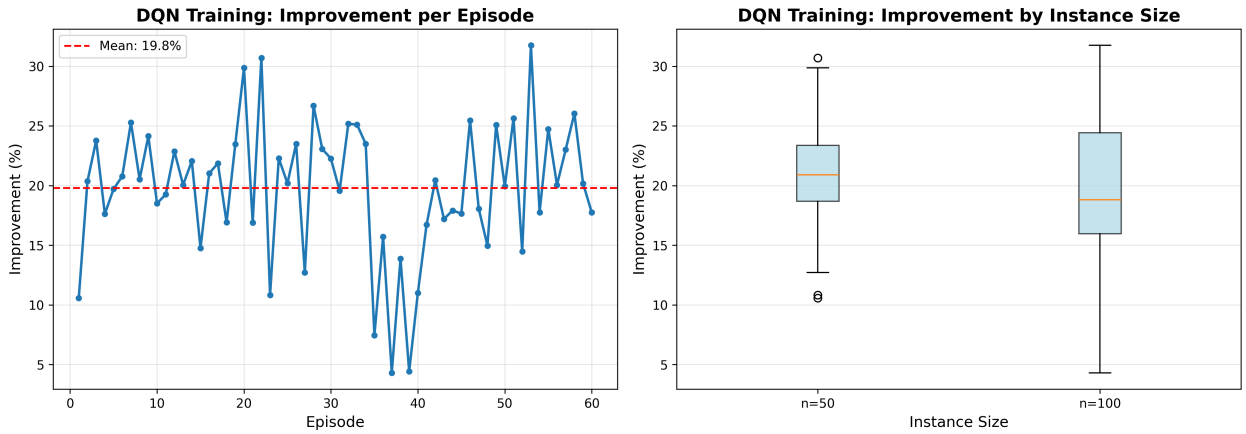

Figure 11: DQN training progress: improvement per episode with mean (left) and improvement distribution by instance size (right)

Parameter evolution on a sample instance (Figure 13) reveals how the learned policy operates. Both severity and temperature multipliers go up and down throughout rather than converging to fixed values. The overlaid stagnation periods (red shading) show that parameter choices correlate with search progress—DQN adjusts its strategy based on whether the search is making progress or plateauing. The objective convergence confirms the policy successfully guides the search toward better solutions.

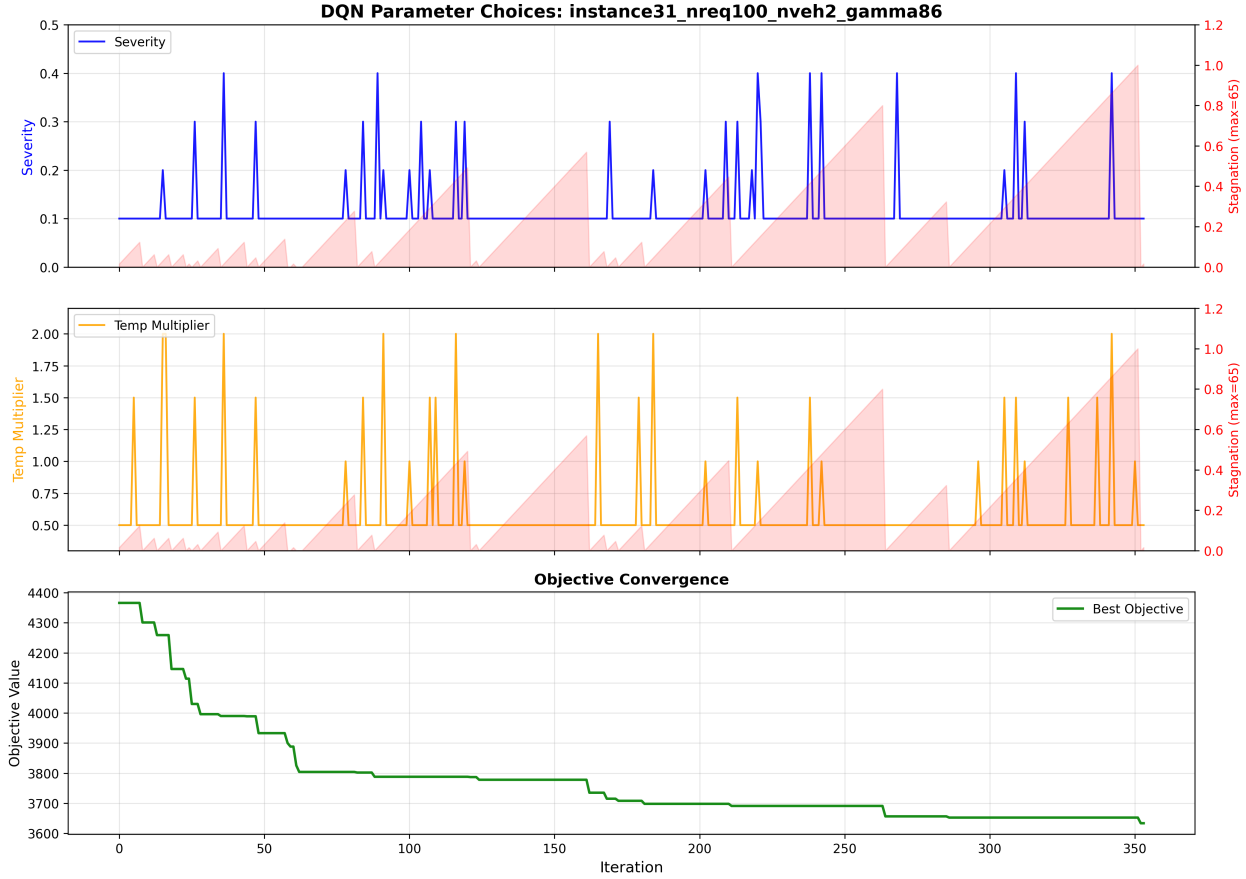

Figure 12: DQN parameter evolution with context: severity (top) and temperature multiplier (middle) choices overlaid with search stagnation (red shading), plus objective convergence (bottom)

Evaluation on 120 test instances (sizes 50, 100, 200, 500) reveals a size-dependent pattern: DQN performs better on smaller instances but Thompson dominates on larger ones (Table 3, Figure 12). DQN wins decisively at $n = 100$ (29-1, -4.4%, $p < 0.001$) and marginally at $n = 50$ (31-29, -1.5%, $p = 0.007$).

However, Thompson wins strongly at $n = 500$ (28-2, +5.1%, $p < 0.001$) and at $n = 200$ (20-10, +2.0%, $p = 0.007$). Overall, Thompson achieves 3.4% lower total objective across all instances.

Table 3: Thompson vs DQN on test instances

Size	Thompson Obj	DQN Obj	% Diff	Wins (T/D)	p-value
50	22444 ± 21087	23452 ± 22168	-1.5%	29/31	0.007*
100	3872 ± 284	3698 ± 267	-4.4%	1/29	<0.001***
200	10780 ± 956	10983 ± 879	$+2.0\%$	20/10	0.007*
500	43386 ± 3498	45522 ± 2941	$+5.1\%$	28/2	<0.001***
Total	1,786,215	1,847,584	$+3.4\%$	78/72	—

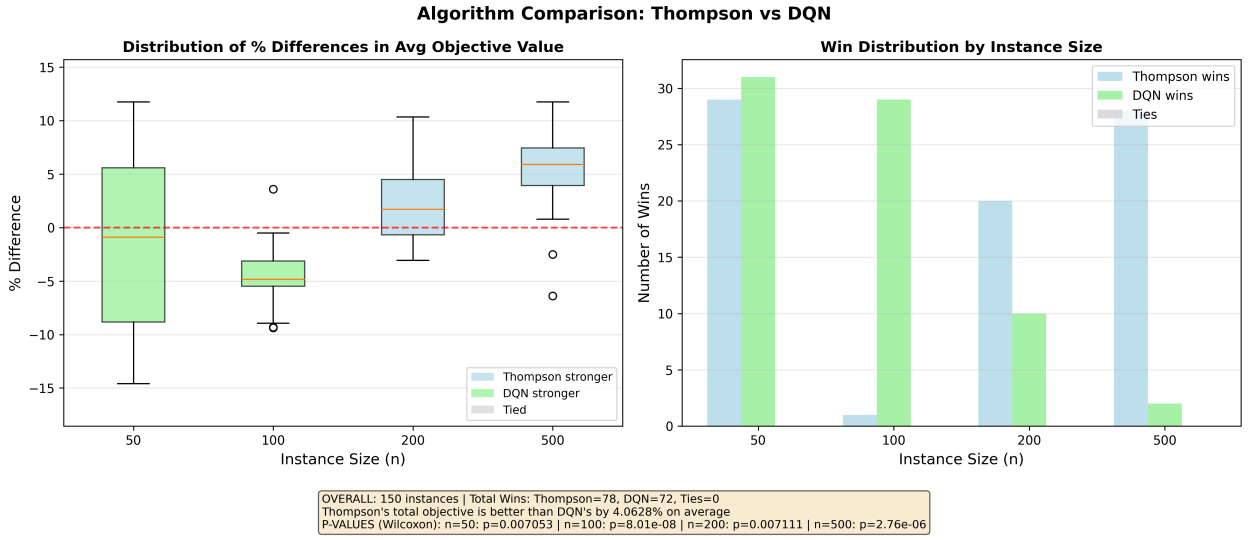

Figure 13: Thompson vs DQN: distribution of percentage differences and win counts by instance size. Positive values favor Thompson.

4.3 Discussion

The results reveal a critical limitation: DQN's performance deteriorates with instance size. While it excels on small instances ($n = 100$: -4.4%), it fails dramatically on large ones ($n = 500$: $+5.1\%$). This pattern suggests the learned policy doesn't generalize beyond its training distribution, creating a fundamental mismatch between training and deployment.

DQN trained exclusively on sizes 50-100, then deployed on sizes up to 500. The learned policy apparently doesn't generalize beyond its training distribution. Larger instances have different characteristics (longer runtimes allow more iterations, different neighborhood structures, altered acceptance dynamics), and the frozen policy cannot adapt. Thompson Sampling, learning online from scratch on each instance, adapts naturally regardless of size.

The training approach compounds this issue. With only 60 training episodes and ∼5k network parameters, DQN may have overfit to small-instance patterns. The 7-feature state representation captures normalized metrics (objective ratios, temperature decay, stagnation), which should theoretically scale across sizes. However, the absolute search dynamics clearly differ enough that the learned Q-values fail to transfer.

The joint action space (144 discrete actions) creates another challenge. While controlling severity and temperature offers flexibility, it increases sample complexity. With limited training data, the network

may not learn robust associations between states and parameter choices. MAB's simpler operator-only selection requires less data to identify good strategies.

Most fundamentally, the problem may not benefit from offline learned policies. If optimal operator choice varies primarily with instance characteristics (size, structure) rather than search state, then per-instance online adaptation (Thompson) beats pre-trained policies (DQN). The size-dependent performance strongly supports this: what works for $n = 100$ differs from $n = 500$, and no single frozen policy handles both well.

References

[1] Cai, J., Kadıoğlu, S., & Dilkina, B. (2024). *BALANS: Multi-Armed Bandits-based Adaptive Large Neighborhood Search for Mixed-Integer Programming Problems.* arXiv preprint arXiv:2410.07393.

[2] Phan, T., Huang, T., Dilkina, B., & Koenig, S. (2024). *Adaptive Anytime Multi-Agent Path Finding Using Bandit-Based Large Neighborhood Search.* In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).

[3] Reijnen, R., Zhang, Y., Lau, H. C., & Bukhsh, Z. (2024). *Online Control of Adaptive Large Neighborhood Search Using Deep Reinforcement Learning.* In Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS).