DYN-LNS: ONLINE LEARNING-BASED ADAPTIVE CONTROL FOR LARGE NEIGHBORHOOD SEARCH WITH EVOLVING SEARCH DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

The performance of Large Neighborhood Search (LNS) depends critically on a series of interdependent control decisions, including the configuration of destroy or repair operators, and the solution acceptance rule. However, the effectiveness of these decisions may change substantially as the search progresses, whereas existing online selection methods typically rely on globally stationary performance estimates and remain state-agnostic. Another class of methods that account for such dynamic evolution generally depends on policies pretrained offline, which incurs substantial training costs and often leads to poor generalization ability. To address this gap, we propose Dyn-LNS, a general online learning-based control framework that dynamically adapts LNS decisions to the evolving search dynamics. The proposed method comprises three key techniques: 1) a state characterization mechanism based on bucket partitioning, capable of capturing search progress and stagnation without offline pre-training; 2) heterogeneous credit tables that enable independent adaptive adjustment of destroy operators, destruction severity, and acceptance temperature at a state-specific decision resolution; and 3) a recency-weighted credit update strategy that allows operator credit values to dynamically increase or decay over the course of the search process. We evaluate Dyn-LNS on representative combinatorial optimization (CO) and mixed integer programming (MIP) problems, using primal gap (PG) and primal integral (PI) as metrics. Experiments demonstrate that Dyn-LNS generally matches or outperforms existing adaptive LNS strategies across the tested benchmarks. Ablation studies further analyze the individual contributions of state conditioning, non-stationary credit estimation, and heterogeneous control.

1 INTRODUCTION

Combinatorial optimization problems (COPs), such as vehicle routing (Toth & Vigo, 2002), scheduling (Manne, 1960), and facility layout (Heragu & Kusiak, 1991), arise widely in real-world decision-making scenarios, and most of them are NP-hard, which makes designing efficient and effective algorithms both important and challenging. A large class of COPs can be flexibly encoded and solved as mixed-integer programs (MIPs), and the academic and industrial communities have invested substantial research and engineering effort into building MIP solvers such as SCIP (Achterberg, 2009), Gurobi (Gurobi Optimization, LLC, 2025), and CPLEX (IBM ILOG, 2025). At the core of these solvers lie tree-search algorithms such as branch-and-bound (BnB) (Land & Doig, 1960), augmented with a range of heuristics. However, due to its exhaustive search nature, BnB often struggles to scale to large, difficult instances; metaheuristics such as Large Neighborhood Search (LNS) offer an appealing alternative, capable of quickly finding high-quality solutions and improving anytime scalability (Pisinger & Ropke, 2018).

A growing body of work combines learning with these two solving paradigms in complementary ways. Within branch-and-bound, researchers have trained learned policies to imitate strong branching decisions (Khalil et al., 2016; Gasse et al., 2019), and to rank candidate nodes so that the search dives more quickly toward better incumbents (Labassi et al., 2022). Beyond the exact search tree, other work explores whether learning can directly replace hand-designed heuristics: for example, predicting a partial variable assignment and searching a small neighborhood around it (Nair et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2020; Han et al., 2023), learning which neighborhood an LNS-style local search should explore next (Wu et al., 2021; Sonnerat et al., 2021; Huang et al., 2023), and learning when to trigger internal primal heuristics during the search (Khalil et al., 2017; Chmiela et al., 2021).

Unlike the offline-trained approaches above, another line of research selects LNS destroy heuristics or neighborhoods based entirely on online feedback, either by adaptively reweighting a fixed portfolio of operators or by casting the choice as a multi-armed bandit problem, an idea also used to drive LNS in adjacent settings such as multi-agent path finding (Phan et al., 2024). However, these online methods typically treat every destroy-repair combination as a context-free arm that is unaware of the actual state of the search, and implicitly assume that each arm's payoff is stationary throughout the run, even though an operator that is effective early on often ceases to be effective later. State-aware alternatives exist, but only by training a policy in advance on a large corpus of other instances and freezing it before deployment, reintroducing the offline training cost that online methods are meant to avoid.

In this paper, we investigate how to make purely online LNS control aware of the evolving search dynamics within a single solve, without relying on any offline-trained model. We propose Dyn-LNS, which characterizes the current search state using a stagnation signal together with the fraction of the iteration budget consumed so far, and uses this state to modulate three independent, heterogeneously updated credit tables, governing the destroy operator, the destroy severity, and the acceptance temperature, respectively; entries in each table are updated via a recency-weighted averaging scheme, so that an operator's estimated credit can rise or decay as the search progresses instead of converging to a fixed long-run average.

Our main findings and contributions are as follows:

- We propose Dyn-LNS, a bandit-based LNS control method, to the best of our knowledge, it's the first online method that adapts LNS decisions to be aware of the search dynamics.
- We demonstrate experimentally that, on synthetic families of combinatorial optimization problems as well as on MIPLIB2017 instances, Dyn-LNS consistently matches or exceeds the performance of existing LNS baselines, in terms of both primal gap and primal integral.
- We conduct ablation experiments that isolate the individual contributions of state conditioning, non-stationary credit estimation, and heterogeneous control.

2 Related work

2.1 Classical destroy heuristics for LNS

A large body of classical, non-learned neighborhoods has been proposed to improve LNS inside a MIP solver, including local branching (Fischetti & Lodi, 2003), relaxation induced neighborhoods (Danna et al., 2005), RENS (Berthold, 2014), DINS (Ghosh, 2007), proximity search (Fischetti & Monaci, 2014), and evolutionary solution polishing (Rothberg, 2007). These neighborhoods are effective individually, but which one to apply, and when, is typically left to hand-tuned rules or a fixed schedule.

2.2 Learning-augmented LNS

A separate line of work instead learns this choice from data, offline, prior to deployment. Song et al. (2020) show that even a randomized destroy heuristic can outperform BnB, and learn a decomposition-based destroy policy with reinforcement and imitation learning; Sonnerat et al. (2021) and Wu et al. (2021) instead imitate or reinforce variable-selection policies over a bipartite-graph encoding of the MIP; and Huang et al. (2023) use contrastive learning to the same end. Going further, Reijnen et al. (2024), Wang et al. (2025), Yu et al. (2026), Xie & Li (2025), and Xu et al. (2025) condition the destroy/repair choice on features of the search state itself, via a deep reinforcement learning policy trained offline and frozen before deployment. A related line of work applies the same offline-training paradigm to a different decision, namely when to trigger which primal heuristic inside the branch-and-bound tree (Khalil et al., 2017; Chmiela et al., 2021), rather than which neighborhood an LNS-style search should explore. All of these methods require a training phase over a representative sample of instances prior to deployment.

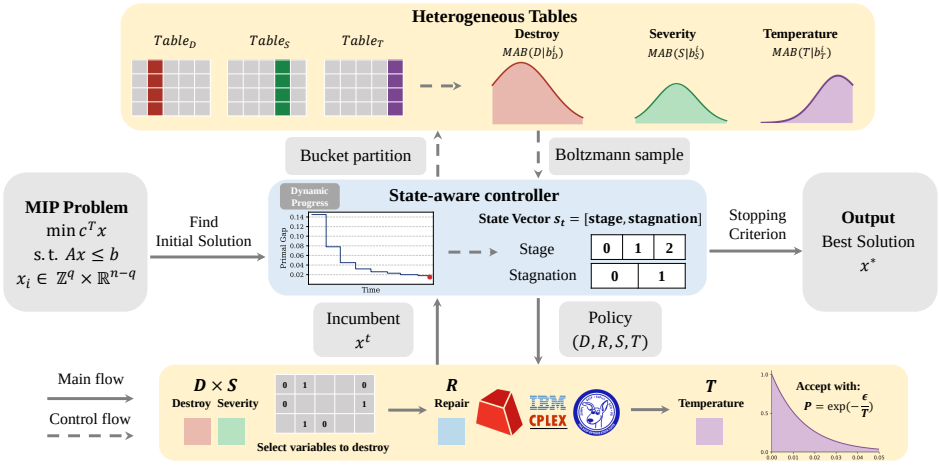


Figure 1: Overview of the Dyn-LNS iteration loop. At each step, the controller characterizes the current search state via two signals, progress stage and stagnation level, forming a fine-grained bucket b_t (for the destroy decision) or a coarse bucket $\bar{b}_t$ (for severity and temperature). Three independent heterogeneous credit tables are row-indexed by the appropriate bucket and sampled via a Boltzmann policy to produce three control decisions. The selected destroy operator frees a subset of variables, the resulting sub-MIP is reoptimized, and only the visited table entries are updated via the recency-weighted EWMA rule (Eq. (3)) using the observed reward.

2.3 Online learning-based LNS

A smaller body of methods instead selects the neighborhood or destroy heuristic online, without any offline training, treating the choice as an adaptive decision made from feedback observed during the search itself. Hendel (2022) reweights a fixed portfolio of neighborhoods according to their performance observed so far; Chmiela et al. (2023) use a multi-armed bandit to jointly schedule LNS and diving heuristics within the branch-and-bound tree; and Cai et al. (2024) propose BALANS, a meta-solver that runs a bandit over a portfolio of LNS neighborhoods on top of an MIP solver, later extended to run in parallel by Yilmaz et al. (2025).

3 Method

We consider MIPs of the form $\min_x\{c^\top x : Ax \le b, x_i \in \mathbb{Z}\ \forall i \in \mathcal{I}\}$, where $\mathcal{I}$ indexes the integer-constrained variables. Dyn-LNS follows the standard Large Neighborhood Search (LNS) loop (Pisinger & Ropke, 2018), shown in Figure 1: starting from an initial solution x^0, at each iteration t, given the incumbent x^t, a subset $\mathcal{X}^t \subseteq \mathcal{I}$ is freed and reoptimized while all other variables remain fixed at their values in x^t, until a stopping condition is met. Dyn-LNS makes three decisions at each iteration: a destroy operator d_t, a destruction severity ρ_t, and an acceptance temperature τ_t. The severity ρ_t sets $k^t = \lceil \rho_t \cdot |\mathcal{I}| \rceil$, the number of integer variables to free; the destroy operator d_t determines how $\mathcal{X}^t$ (of size k^t) is chosen from $\mathcal{I}$; the resulting sub-MIP is reoptimized by a MIP solver to produce a candidate solution $\hat{x}$; and the temperature τ_t parameterizes how likely a $\hat{x}$ that degrades the incumbent is nonetheless accepted as x^{t+1}. The outcome is scored with a reward r_t, which is used to update whichever credit estimates the three decisions were drawn from, so that the next iteration's decisions reflect everything observed so far. This repeats until a wall-clock time budget T is exhausted. The rest of this section motivates and describes how Dyn-LNS actually makes these three decisions.

Reward. We score the outcome of each iteration using a four-tier scheme adapted from the adaptive LNS literature (Pisinger & Ropke, 2018): a rejected candidate scores 0; a candidate accepted without improving on x^t scores 0.1; a candidate that improves on x^t but does not improve on the best solution found so far scores 0.4; and a candidate that establishes a new best-known solution

scores

$$r_t = \frac{1}{2} + \frac{1}{2}\,\mathrm{clip}\left(\frac{\Delta_t}{c},\, 0,\, 1\right), \quad \Delta_t = \frac{f(x^\star) - f(x^{t+1})}{\max(|f(x^\star)|, \epsilon)}, \tag{1}$$

where $f(x^\star)$ is the objective value of the best solution found prior to iteration t, $\epsilon > 0$ avoids division by zero, and $c > 0$ is a saturation scale below which larger relative improvements are rewarded more. Since $\Delta_t \geq 0$ whenever a new best solution is found, r_t always falls in $(0.5, 1]$, strictly exceeding the fixed scores of the other three tiers, while still rewarding larger improvements more than marginal.

3.1 Hypotheses

Dyn-LNS is built around two hypotheses about how a destroy-repair LNS search actually behaves, neither of which is exploited by existing online learning LNS methods.

Hypothesis 1 (Non-stationary operator value). How much a given destroy operator is worth, i.e., how much reward it is likely to earn if invoked right now, changes over the course of a single solve rather than staying fixed. An aggressive destroy that frees a large fraction of the variables is well suited to escaping a poor starting incumbent or a long stagnation stretch, but that same destroy is more likely to discard useful structure once the search is already close to a strong solution and improving steadily on its own. Consequently, an operator that was reliably useful earlier in the search need not remain useful later, and one number meant to summarize how good an operator is, averaged over the whole run, is chasing a moving target.

Hypothesis 2 (Separable destroy type and severity). Which destroy-operator type to apply and how large a neighborhood to destroy can be decided independently of one another. The best-performing destroy family for a given MIP is mostly governed by its constraint structure (e.g., whether the instance rewards operators that exploit graph-like neighborhoods over generic variable subsets), while the best severity is more a function of how much of the resulting sub-MIP can be reoptimized within the available time and how confident the search already is that a small, local move suffices. Because these two questions appear to be driven by largely different considerations, it is reasonable to infer that good choices along each axis may be learned separately.

3.2 State conditioning and non-stationary credit estimation

Hypothesis 1 implies that a credit estimate which ignores where the search currently stands, and which is updated as a plain long-run average, is fundamentally mismatched to how an operator's credit actually evolves. Dyn-LNS addresses this along two complementary axes: it conditions every credit estimate on a summary of the current search state, so the same operator can carry a different estimated credit depending on when it is invoked, and it updates every estimate with a recency-weighted rule, so that even a single table entry can rise or decay as new evidence arrives.

Search state characterization. Dyn-LNS characterizes the current search state using two signals computed directly from the solver's own progress log. The first is a stagnation signal, defined as the number of iterations elapsed since the incumbent objective was last improved, where $t_{\text{last_improve}}(t)$ is the most recent iteration index strictly before t at which the incumbent improved, normalized by a fixed window W_{stagn} and clipped to $[0, 1]$:

$$\mathrm{stagn}(t) = \mathrm{clip}\left(\frac{t - t_{\text{last_improve}}(t)}{W_{\text{stagn}}},\, 0,\, 1\right). \tag{2}$$

The second is the fraction of the wall-clock time budget consumed so far, $\mathrm{elapsed}(t)/T$, where $\mathrm{elapsed}(t)$ is the wall-clock time elapsed since the start of the solve as of iteration t; we partition this fraction into three equal-width stages, early ($\mathrm{elapsed}(t)/T < 1/3$), mid ($1/3 \leq \mathrm{elapsed}(t)/T < 2/3$), and late ($\mathrm{elapsed}(t)/T \geq 2/3$).

We binarize $\mathrm{stagn}(t)$ at its midpoint into a stalled/not-stalled label, and cross it with the three progress stages to obtain a 6-way state bucket b_t, used to index the destroy table. The severity and temperature tables use a coarser, 2-way bucket $\bar{b}_t$; since severity and temperature govern how aggressively the search perturbs the incumbent and how willing it is to accept a worse solution,

both of which are classical responses to a search being stuck, we set $\bar{b}_t$ to the stalled/not-stalled label alone, discarding the progress-stage component. This state-conditioning is what lets the destroy/severity/temperature credit tables introduced in Section 3.3 hold a different estimate per state instead of one number per operator.

Recency-weighted credit update. At the end of iteration t, only the table entries actually read at iteration t are updated, via an exponentially weighted moving average (EWMA) of the reward r_t:

$$T[\cdot] \leftarrow (1-\lambda)\, T[\cdot] + \lambda\, r_t, \tag{3}$$

with its own recency weight λ_d, λ_ρ, or λ_τ depending on whether the destroy, severity, or temperature table was read; all other, unvisited entries are left unchanged. This replaces the implicit $\lambda = 1/n_{\text{visits}}$ long-run average, under which an entry's estimate converges to a fixed value and becomes increasingly insensitive to new evidence as n_{visits} grows. Under Eq. (3), by contrast, an operator's estimated credit can rise or decay throughout the search, tracking its most recently observed payoff in its current state, directly addressing Hypothesis 1.

3.3 Heterogeneous control

Hypothesis 2 implies that the three control decisions need not be forced into a single joint choice learned over the product of all destroy operator, severity, and temperature combinations. Dyn-LNS operationalizes this via heterogeneous control: three independent, heterogeneously updated credit tables T_d, T_ρ, T_τ govern the destroy operator, severity, and temperature decisions respectively, each read and updated entirely on its own.

Heterogeneous credit tables. All three tables are initialized to zero at the start of a solve. The destroy table has one row per 6-way bucket b_t; the severity and temperature tables have one row per 2-way bucket $\bar{b}_t$, reflecting the coarser state resolution. The destroy operators, and the discrete severity and temperature levels, are adapted from the same portfolios used in prior adaptive LNS work (Pisinger & Ropke, 2018; Cai et al., 2024). Maintaining three separate tables rather than one joint table over all combinations, is what allows destroy type, severity, and temperature to be estimated independently, and at the bucket resolution that best matches how often each is actually revisited.

Boltzmann-categorical sampling. Each of the three decisions is drawn independently from a Boltzmann distribution over its own table, row-indexed by its own bucket resolution. For the destroy operator,

$$P(d \mid b_t) \propto \exp\big(\beta_t\, T_d[b_t, d]\big), \tag{4}$$

and the severity and temperature decisions are drawn from the same functional form, reading $T_\rho[\bar{b}_t, \cdot]$ and $T_\tau[\bar{b}_t, \cdot]$ respectively. All three distributions share a single, linearly annealed inverse-temperature schedule,

$$\beta_t = \beta_0 + (\beta_1 - \beta_0)\frac{\text{elapsed}(t)}{T}, \quad \beta_0 < \beta_1, \tag{5}$$

so that operator selection starts close to uniform, when table entries are still noisy, and sharpens toward the entries with the highest estimated credit as evidence accumulates over the course of the search. To make sure every table entry holds a reward-based estimate before this sharpening begins, the first N_{cold} iterations instead select d_t, ρ_t, τ_t uniformly at random, independent of the still-mostly-empty tables; the recency-weighted update rule still applies during this phase, so that by iteration $N_{\text{cold}} + 1$ every visited entry already reflects observed evidence.

Acceptance rule. Given a candidate $\hat{x}$, Dyn-LNS decides x^{t+1} through the criterion below:

$$x^{t+1} = \begin{cases} \hat{x} & \text{if } f(\hat{x}) \le f(x^t), \\ \hat{x} & \text{w.p. } p_t, \text{ if } f(\hat{x}) > f(x^t), \\ x^t & \text{otherwise (w.p. } 1 - p_t), \end{cases} \tag{6}$$

$$p_t = \exp\left(-\frac{f(\hat{x}) - f(x^t)}{\tau_t \max(|f(x^t)|, \epsilon)}\right), \tag{7}$$

Algorithm 1 Dyn-LNS

Require: initial solution x^0, wall-clock time budget T, cold-start length N_{cold}, annealing bounds β_0, β_1, recency weights $\lambda_d, \lambda_\rho, \lambda_\tau$
Ensure: best solution $x^\star$

```
Initialize T_d, T_ρ, T_τ ← 0; x* ← x^0; t ← 0
while elapsed(t) < T do
    Compute stagn(t) (Eq. (2)) and progress stage from elapsed(t)/T; form buckets b_t, b̄_t
    if t < N_cold then
        Sample d_t, ρ_t, τ_t uniformly at random
    else
        Compute β_t (Eq. (5))
        Sample d_t ~ Boltzmann(T_d[b_t, ·]; β_t)
        Sample ρ_t ~ Boltzmann(T_ρ[b̄_t, ·]; β_t), τ_t ~ Boltzmann(T_τ[b̄_t, ·]; β_t)
    end if
    X^t ← free k^t = ⌈ρ_t|I|⌉ variables via destroy operator d_t
    x̂ ← reoptimize the sub-MIP induced by X^t
    x^{t+1} ← accept or reject x̂ via Eqs. (6)–(7)
    Compute reward r_t (Eq. (1))
    Update the visited entries of T_d, T_ρ, T_τ via EWMA (Eq. (3))
    if f(x^{t+1}) improves on f(x*) then
        x* ← x^{t+1}
    end if
    t ← t + 1
end while
return x*
```

with $p_t := 0$ when $\tau_t = 0$. A candidate that does not degrade the incumbent is therefore always accepted, independent of τ_t; a degrading candidate is accepted with a probability that decreases exponentially in its relative degradation and increases in τ_t, so τ_t governs how tolerant the search currently is of temporarily worse solutions. τ_t is itself one of the three heterogeneous decisions, drawn each iteration from its own credit table T_τ conditioned on the coarse stagnation bucket $\bar{b}_t$, from a discrete level set that includes $\tau = 0$, which collapses Eq. (6) to strict hill-climbing. This lets the search learn, separately for each stagnation state, how much tolerance for worsening moves is actually useful, rather than committing to one global annealing curve.

Given the three sampled decisions, the destroy operator d_t selects $\mathcal{X}^t$ of size $k^t = \lceil \rho_t |\mathcal{I}| \rceil$ from $\mathcal{I}$; the resulting sub-MIP is solved to produce a candidate $\hat{x}$; and $\hat{x}$ is accepted or rejected as x^{t+1} following Eq. (6). We summarize the overall procedure as Algorithm 1.

4 Experiments

In this section, we design experiments to validate the effectiveness of our method and conduct ablation studies to verify the contribution of each component. Hyperparameter sensitivity analysis is provided in Appendix E.

4.1 Setup

Evaluation metrics. We report Primal Gap (PG) and Primal Integral (PI) (Berthold, 2013). The primal gap at time t is

$$\gamma(x^t) = \begin{cases} 0 & \text{if } |f(x^t)| = |f^\star| = 0, \\ 1 & \text{if } f(x^t) \cdot f^\star < 0, \\ \dfrac{|f(x^t) - f^\star|}{\max(|f(x^t)|, |f^\star|)} & \text{otherwise,} \end{cases} \tag{8}$$

where $f^\star$ is the best objective value found for the instance across all compared methods in the main comparison, and across the four ablation variants in the ablation studies (Section 4.3 and

Appendix D); $\gamma(x^t) = 1$ if no feasible incumbent has yet been found. PG is $\gamma(x^t)$ at the end of the time budget T, and PI is its integral over the run,

$$\mathrm{PG} = \gamma(x^T), \quad \mathrm{PI} = \int_0^T \gamma(x^t)\,\mathrm{d}t. \tag{9}$$

Datasets. We evaluate the performance on:

1. Synthetic CO instances: four NP-hard problem benchmarks that are widely used in existing studies (Wu et al., 2021; Song et al., 2020; Scavuzzo et al., 2022; Huang et al., 2023), which consist of two graph optimization problems, namely the minimum vertex cover (MVC) and maximum independent set (MIS) problems, and two non-graph optimization problems, namely the combinatorial auction (CA) and set covering (SC) problems. Table 4 (Appendix B.2) reports the average number of variables, constraints, and nonzero coefficients for each instance type.
2. MIPLIB2017 (Gleixner et al., 2021) instances: 35 real-world MIPs drawn from the MIPLIB2017 hard collection, after filtering out instances on which SCIP, BALANS, or any other baseline failed to produce a result (see Appendix B.3 for the full selection).

Baselines. We compare against five SCIP-backed methods: **SCIP-DEFAULT** (default heuristic schedule), **SCIP-AGGR** (aggressive heuristic emphasis), **RANDOM** (random 50% destroy with no selection), **ALNS** (Hendel, 2022) (portfolio reweighting by observed success rate), and **BALANS** (Cai et al., 2024) (bandit over destroy–repair combinations). We additionally include **Gurobi-DEFAULT** and **Gurobi-HEUR** as commercial-solver references.

Hyperparameters. Destroy families: {mutation, RINS, RENS, local branching, proximity}. Sub-MIPs are reoptimized by SCIP. Severity levels $\rho \in \{25\%, 50\%, 75\%\}$. Temperature levels $\tau \in \{0, 0.0015, 0.005, 0.015, 0.04\}$. $N_{\text{cold}} = 40$, $W_{\text{stagn}} = 100$, $\lambda_d = 0.15$, $\lambda_\rho = \lambda_\tau = 0.05$, $\beta_0 = 1.0$, $\beta_1 = 8.0$, $c = 0.01$. Sub-MIP solves are capped at 20s for the first incumbent and 60s per subsequent iteration. Baseline-method configurations are provided in Appendix B.4.

4.2 Performance evaluation

Synthetic instances. Figure 2 reports, for each of the eight synthetic family-size combinations, the relative rank of every method against all others, aggregated over the corresponding test instances. Table 1 reports the corresponding Primal Gap and Primal Integral statistics, at the 3,600s time budget. We observe that: 1) Figure 2 shows Dyn-LNS's octagon is the largest and most uniformly filled among the eight methods, indicating its comprehensive advantage. 2) Table 1 demonstrates that, among the six SCIP-backed methods, Dyn-LNS ranks first on 9 of the 16 metrics and second on 6, placing in the top two on 15 of 16 metrics. The remaining seven first-place results are split among RANDOM (three), SCIP-AGGR (two), and BALANS (two). 3) Gurobi performs well mainly on CA, yielding the lowest PG and PI on both CA sizes, but it is less competitive on SC and MVC, and notably, degrates severely on MIS.

MIPLIB2017. To assess whether Dyn-LNS generalizes beyond the synthetic instance families studied above, we additionally evaluate on 35 instances drawn from the MIPLIB 2017 (Gleixner et al., 2021) hard collection (see Appendix B.3 for full instance list and filtering criteria).

Table 2 reports Primal Gap and Primal Integral at the 3,600s budget. Among the SCIP-backed methods, Dyn-LNS achieves the lowest mean PG (7.63%) and PI (622.1), outperforming SCIP-DEFAULT (11.83%, 857.7), ALNS (12.01%, 920.9), BALANS (8.21%, 698.2), SCIP-AGGR (8.58%, 695.0), and RANDOM (25.82%, 1559.9). Gurobi-DEFAULT and Gurobi-HEUR are included as a commercial-solver reference; as expected, Gurobi remains the strongest solver overall (PG 1.53% and 3.07%, respectively).

4.3 Ablation study

Here we test three key mechanisms directly, by disabling each of them inside the Dyn-LNS pipeline and measuring the resulting change in solution quality. Each ablation modifies exactly one mechanism, holds the destroy operator pool, reward, acceptance rule, and cold-start phase fixed, and is run

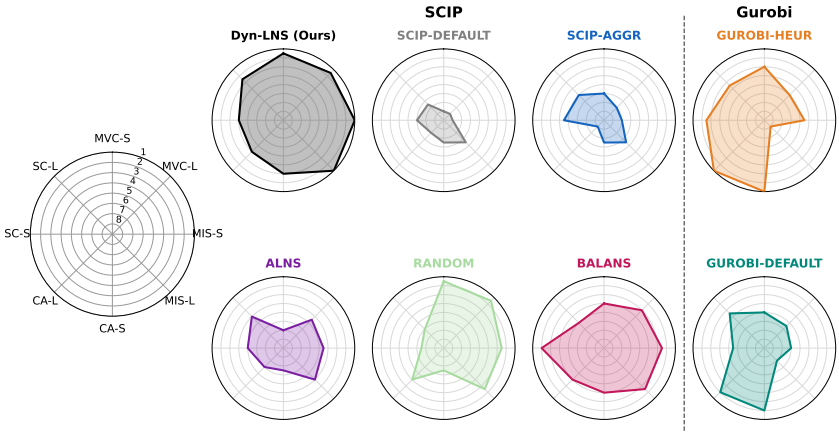


Figure 2: Per-family rank comparison across the eight synthetic family-size combinations (MVC-S/L, MIS-S/L, CA-S/L, SC-S/L), with one octagon panel per method. Each axis is a family-size combination, and the plotted value is computed as the arithmetic mean of the ranks of primal gap (PG) and primal integral (PI). A larger, more filled octagon indicates a method that ranks consistently better across families.

Table 1: PG (%, lower is better) and PI (lower is better) at a 3,600s (60-minute) cutoff, mean ± std over 30 test instances per family-size combination. Bold values mark the best (lowest) result in each column among the SCIP-backed methods and underline marks the second-best; separately, bold also marks the best result among the Gurobi-backed methods when it improves upon every SCIP-backed method.

	MVC-S		MIS-S		CA-S		SC-S	
Method	PG (%)↓	PI↓	PG (%)↓	PI↓	PG (%)↓	PI↓	PG (%)↓	PI↓
Dyn-LNS	**0.15±0.37**	14.2±20.9	**0.01±0.04**	**20.3±9.3**	**1.52±0.89**	**138.0±49.4**	0.76±1.00	71.4±53.6
SCIP-DEFAULT	1.14±0.30	70.8±17.0	5.15±1.62	394.5±60.4	2.95±0.77	298.1±56.6	0.51±0.70	314.3±66.3
SCIP-AGGR	0.35±0.22	36.5±15.4	4.57±0.55	342.4±37.7	2.51±0.82	389.7±126.4	**0.33±0.57**	306.0±53.9
RANDOM	0.18±0.36	**13.7±21.3**	0.63±0.29	45.4±18.4	3.44±0.75	229.0±47.7	1.34±1.24	103.3±80.5
ALNS	0.46±0.36	42.3±18.0	1.38±0.24	213.5±19.8	2.54±1.07	301.2±65.7	1.00±0.96	75.7±56.2
BALANS	0.25±0.17	23.4±10.2	0.45±0.34	108.0±28.5	1.63±0.85	220.2±65.1	0.49±0.73	**69.5±53.1**
Gurobi-DEFAULT	0.33±0.22	27.0±12.5	1.76±0.97	337.1±39.4	0.25±0.27	35.6±18.2	0.60±0.83	133.5±49.0
Gurobi-HEUR	0.23±0.20	22.9±11.3	0.80±0.45	292.9±34.9	**0.20±0.38**	**33.7±20.8**	**0.14±0.33**	89.2±33.6

	MVC-L		MIS-L		CA-L		SC-L	
Method	PG (%)↓	PI↓	PG (%)↓	PI↓	PG (%)↓	PI↓	PG (%)↓	PI↓
Dyn-LNS	**0.10±0.13**	10.5±7.1	**0.01±0.03**	**30.5±8.1**	2.59±1.16	249.8±59.2	0.95±0.90	**88.6±60.5**
SCIP-DEFAULT	1.17±0.20	71.1±12.2	7.89±1.19	501.3±31.8	8.26±5.59	962.4±117.3	0.99±1.14	1949.0±225.9
SCIP-AGGR	0.75±0.19	67.8±11.6	7.59±0.85	505.0±23.9	10.20±6.05	1014.4±52.4	**0.59±0.83**	1656.8±389.2
RANDOM	0.10±0.11	**8.1±6.2**	0.72±0.27	52.5±14.0	3.61±0.66	**238.3±35.9**	1.83±1.33	133.6±75.6
ALNS	0.29±0.19	35.8±8.9	1.95±0.29	256.8±15.9	6.27±1.56	645.2±96.0	1.24±1.02	105.3±58.0
BALANS	0.22±0.17	25.0±10.0	0.38±0.30	136.2±35.8	**2.14±1.15**	397.0±100.1	1.52±1.18	132.1±74.0
Gurobi-DEFAULT	0.32±0.19	46.5±10.9	9.99±0.84	819.5±41.4	0.48±0.37	90.2±23.6	0.57±0.68	407.1±98.0
Gurobi-HEUR	0.23±0.14	54.1±8.6	10.68±1.67	882.1±34.3	**0.03±0.11**	**82.1±12.5**	**0.20±0.38**	414.8±52.7

on the same eight synthetic family-size combinations, 30 test instances per combination, and 3,600s budget:

- **Dyn-LNS-S** collapses the 6-way bucket b_t and 2-way bucket $\bar{b}_t$ to a single state shared by all three credit tables, so every table holds only one estimate per operator.
- **Dyn-LNS-R** replaces the recency-weighted update (Eq. (3)) with the implicit $\lambda = 1/n_{\text{visits}}$ long-run average.
- **Dyn-LNS-H** merges the destroy-type table T_d and severity table T_ρ into a single 6×15 table indexed by the 6-way bucket b_t.

Table 2: PG (%, lower is better) and PI (lower is better) at a 3,600s cutoff on the MIPLIB-Hard subset, mean ± std over all test instances. Bold values mark the best (lowest) result in each column among the SCIP-backed methods and underline marks the second-best; separately, bold also marks the best result among the Gurobi-backed methods when it improves upon every SCIP-backed method.

Solver	Method	PG (%)↓	PI↓
SCIP	**Dyn-LNS**	**7.63±15.34**	**622.1±1000.3**
	SCIP-DEFAULT	11.83±20.95	857.7±1285.0
	SCIP-AGGR	8.58±19.18	<u>695.0±1198.2</u>
	RANDOM	25.82±28.57	1559.9±1711.3
	ALNS	12.01±18.66	920.9±1224.6
	BALANS	<u>8.21±13.81</u>	698.2±999.1
Gurobi	Gurobi-DEFAULT	**1.53±3.64**	**302.1±521.4**
	Gurobi-HEUR	3.07±13.04	337.6±821.0

Table 3: Ablation study on L-size families: PG (%, lower is better) and PI (lower is better) at a 3,600s cutoff, mean ± std over 30 test instances. $f^\star$ is the best value across the four variants, so values are not comparable with Table 1. Bold marks the best (lowest) value in each column per metric and underline marks the second-best.

Variant		MVC-L	MIS-L	CA-L	SC-L
Dyn-LNS	PG↓	**0.06±0.07**	0.13±0.13	**0.34±0.58**	**0.45±0.80**
	PI↓	**4.96±2.5**	**22.82±6.7**	**70.30±19.9**	**35.18±33.9**
Dyn-LNS-S	PG↓	<u>0.07±0.07</u>	**0.12±0.13**	1.34±0.96	0.75±0.99
	PI↓	<u>5.11±2.3</u>	24.54±6.6	98.87±35.0	50.64±39.3
Dyn-LNS-R	PG↓	0.08±0.09	0.12±0.12	1.23±1.05	<u>0.65±0.81</u>
	PI↓	5.75±3.0	25.57±5.9	101.74±34.6	<u>44.27±33.7</u>
Dyn-LNS-H	PG↓	0.07±0.10	<u>0.12±0.12</u>	<u>1.07±1.04</u>	0.71±1.00
	PI↓	5.82±3.3	<u>23.61±5.6</u>	<u>96.16±30.4</u>	44.84±37.6

Table 3 reports PG across all four L-size families, where Dyn-LNS is best or nearly best on every family. The advantage is most pronounced on CA-L and SC-L: ablated variants reach 2–4× higher PG on CA-L and 1.5–1.7× on SC-L. Separate learning helps since the joint table has 90 entries against 36 for T_d and T_ρ, so each entry gets far fewer updates, while the type–severity interaction is insignificant on MVC-L (Appendix F). Additional ablation study on MIPLIB2017 is provided in Appendix D.

5 Conclusion and Limitations

We introduced Dyn-LNS, a general online bandit-based control framework that adapts destroy operator, destruction severity, and acceptance temperature decisions in Large Neighborhood Search (LNS) to the evolving search dynamics without any offline training. On synthetic combinatorial optimization families and MIPLIB2017 instances, Dyn-LNS consistently matches or exceeds existing adaptive LNS baselines in both Primal Gap and Primal Integral, and our ablation studies confirm that state conditioning, non-stationary credit estimation, and heterogeneous control each contribute to this improvement.

Despite these results, Dyn-LNS has several limitations. First, Dyn-LNS does not achieve the best result on every family-size combination and metric, though Dyn-LNS achieves a comprehensively stronger performance. Finally, Dyn-LNS is intuitively motivated and empirically validated through experiments, although its superiority lacks a corresponding theoretical guarantee.

For future work, we plan to enrich the state characterization with additional solver-internal signals, to extend the same state-conditioned control to repair-operator selection, and to explore limited cross-instance transfer within the credit tables without reintroducing the offline training cost.

AI USE STATEMENT

In this work, we used generative AI tools to:

- **Aid or Polish Writing:** Correcting grammatical and typographical errors, improving sentence clarity and readability.
- **Retrieval and Discovery:** Assisting in identifying potentially relevant literature, related methods, and useful search directions.
- **Research Ideation or Execution:** Assisting with research execution by supporting code implementation, debugging, and refinement.
- **Draft Sections of the Paper:** Assisting in drafting preliminary formulations or portions of the manuscript.

We did not use generative AI tools to generate synthetic datasets or to prove mathematical claims, and other disclosure categories are not applicable to this work.

We reviewed all AI-assisted work before incorporating it into the paper. References and related works suggested or identified with the assistance of generative AI were independently checked against the original sources. AI-assisted code and implementation suggestions were reviewed, tested, and validated by the authors before use in the experiments. All technical claims, experimental results, analyses, and conclusions were independently verified by the authors.

ETHICS STATEMENT

This research does not involve any personally identifiable information. Our experiments use publicly available benchmark instances and synthetic instances generated using publicly documented, widely adopted procedures, and we have verified that their licenses permit research use. In accordance with the ICLR Code of Ethics(`https://iclr.cc/public/CodeOfEthics`),we ensure that our work adheres to principles of fairness, transparency, and responsible AI research.

REPRODUCIBILITY STATEMENT

We have provided, in as much detail as possible, all the information required to reproduce the results of this paper. The implementation details of Dyn-LNS are given in Section 3 and Algorithm 1, and its hyperparameter settings in Section 4.1. The benchmark instances are described in Appendix B.1–B.3. The configurations of all baselines, together with the hardware and software environment, are provided in Appendix B.4.

REFERENCES

Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Timo Berthold. Measuring the impact of primal heuristics. *Operations research letters*, 41(6): 611–614, 2013.

Timo Berthold. Rens: the optimal rounding. *Mathematical Programming Computation*, 6(1):33–54, 2014.

Junyang Cai, Serdar Kadioğlu, and Bistra N. Dilkina. Balans: Multi-armed bandits-based adaptive large neighborhood search for mixed-integer programming problem. In *International Joint Conference on Artificial Intelligence*, 2024.

Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

Antonia Chmiela, Ambros Gleixner, Pawel Lichocki, and Sebastian Pokutta. Online learning for scheduling mip heuristics. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 114–123. Springer, 2023.

Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

Paul Erd6s and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5(17-61):4, 1960.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Shubhashis Ghosh. Dins, a mip improvement heuristic. In *International Conference on Integer Programming and Combinatorial Optimization*, pp. 310–323. Springer, 2007.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2025. URL `https://www.gurobi.com`.

Qing-Yu Han, L. Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. In *International Conference on Learning Representations*, 2023.

Gregor Hendel. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 14(2):185–221, 2022.

Sunderesh S Heragu and Andrew Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1):1–13, 1991.

Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International conference on machine learning*, pp. 13869–13890. PMLR, 2023.

IBM ILOG. IBM ILOG CPLEX Optimization Studio, 2025. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, volume 2017, pp. 659–666, 2017.

Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*, 35: 32000–32010, 2022.

AH Land and AG Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 66–76, 2000.

Alan S Manne. On the job-shop scheduling problem. *Operations research*, 8(2):219–223, 1960.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Thomy Phan, Taoan Huang, Bistra Dilkina, and Sven Koenig. Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17514–17522, 2024.

David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pp. 99–127. Springer, 2018.

Robbert Reijnen, Yingqian Zhang, Hoong Chuin Lau, and Zaharah Bukhsh. Online control of adaptive large neighborhood search using deep reinforcement learning. In *Proceedings of the international conference on automated planning and scheduling*, volume 34, pp. 475–483, 2024.

Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

Lara Scavuzzo, Feng Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. *Advances in neural information processing systems*, 35:18514–18526, 2022.

Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33: 20012–20023, 2020.

Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pp. 437–448. Springer, 2005.

Zhilong Wang, Xiaoxuan Hu, Haiquan Sun, and Wei Xia. Adns: An adaptive dynamic neighborhood search method guided by joint learning heuristics and corresponding hyperparameters. *Applied Soft Computing*, 180:113280, 2025.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.

Lin Xie and Hanyi Li. Solving the pod repositioning problem with deep reinforced adaptive large neighborhood search. *arXiv preprint arXiv:2506.02746*, 2025.

Ning Xu, Junkai Zhang, Yang Wu, Huigen Ye, Hua Xu, Huiling Xu, and Yifan Zhang. Hyp-aso: A hybrid policy-based adaptive search optimization framework for large-scale integer linear programs. *arXiv preprint arXiv:2509.15828*, 2025.

Alican Yilmaz, Junyang Cai, Serdar Kadioglu, and Bistra Dilkina. Parbalans: Parallel multi-armed bandits-based adaptive large neighborhood search. *arXiv preprint arXiv:2508.06736*, 2025.

Shaohua Yu, Wenhao Mao, Zigao Wu, and Jakob Puchinger. New adaptive mechanism for large neighborhood search using dual actor-critic. *arXiv preprint arXiv:2601.11414*, 2026.

A Background

A.1 Mixed Integer Programming

A mixed integer (linear) program (MIP) is an optimization problem of the form

$$\min_x \; c^\top x \quad \text{s.t.} \quad Ax \le b,\; x_i \in \mathbb{Z} \;\; \forall i \in \mathcal{I}, \tag{10}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$ specify the objective and linear constraints, and $\mathcal{I} \subseteq \{1, \ldots, n\}$ indexes the subset of variables constrained to be integer (the remaining variables are continuous). A solution to the MIP is a feasible assignment of values to all variables.

A.2 Large Neighborhood Search for MIP

Large Neighborhood Search (LNS) is a heuristic algorithm that starts from an initial solution x^0 (e.g., obtained by running branch-and-bound for a short budget) and then iteratively destroys and reoptimizes part of the solution until a stopping condition is met (Pisinger & Ropke, 2018). In iteration $t \ge 0$, given the incumbent solution x^t (the best solution found so far), a destroy heuristic selects a subset of k^t variables $\mathcal{X}^t$ to reoptimize; LNS then solves the corresponding sub-MIP, in which the variables in $\mathcal{X}^t$ remain free while all other variables are fixed to their values in x^t, and the resulting solution becomes the new incumbent x^{t+1}. LNS often improves the objective faster than branch-and-bound on difficult instances (Song et al., 2020; Sonnerat et al., 2021; Wu et al., 2021), and exploring a large neighborhood at each step makes it less prone to getting stuck in local minima than other local search methods.

A.3 Multi-Armed Bandits

The multi-armed bandit (MAB) problem is a class of online sequential decision-making problems (Vermorel & Mohri, 2005): at each round t, an agent selects ("pulls") one of K arms and observes the reward of the arm it pulled, with the goal of maximizing cumulative reward over a horizon of T rounds,

$$\max_{a_1, \ldots, a_T} \; \mathbb{E}\left[\sum_{t=1}^{T} r_t(a_t)\right], \quad a_t \in \{1, \ldots, K\}, \tag{11}$$

where $r_t(a_t)$ is the (possibly stochastic) reward obtained from arm a_t at round t. Because each arm's reward distribution is unknown in advance, the agent must decide, at every round, whether to pull the arm with the highest estimated reward ("exploit") or try another arm to learn more about it ("explore"), estimating the reward of each arm from past pulls via a learning policy (Kuleshov & Precup, 2014).

B Additional Experimental Details

B.1 Synthetic Instance Generation

For each of the four synthetic families studied in this work, we first sample a combinatorial structure (a random graph, a set of auction bids, or a random set-membership matrix) and then construct an equivalent MIP instance of the standard form $\min_x c^\top x$ s.t. $Ax \le b$, $x_i \in \mathbb{Z}\ \forall i \in \mathcal{I}$. Each family is generated at two sizes, S (small) and L (large).

Minimum Vertex Cover (MVC). We generate a random graph $G = (V, E)$ using the Barabási–Albert preferential-attachment model (Albert & Barabási, 2002): starting from a small initial graph, nodes are added one at a time, each connecting to m existing nodes with probability proportional to their current degree, until $|V| = n$ nodes are reached ($n = 1{,}000$ for S, $n = 2{,}000$ for L, and $m = 70$ in both cases, following Song et al. (2020)). The resulting MIP instance is

$$\min_{x \in \{0,1\}^{|V|}} \sum_{v \in V} x_v \quad \text{s.t.} \quad x_u + x_v \ge 1 \;\; \forall (u, v) \in E, \tag{12}$$

i.e., one binary variable per node and one covering constraint per edge.

Maximum Independent Set (MIS). We generate a random graph $G = (V, E)$ using the Erdős–Rényi model (Erd6s & Rényi, 1960): each of the $\binom{n}{2}$ possible edges is included independently with probability p ($n = 6{,}000$, $p = 8/5{,}999$ for S; $n = 12{,}000$, $p = 8/11{,}999$ for L, giving an expected average node degree of 8). The resulting MIP instance is

$$\max_{x \in \{0,1\}^{|V|}} \sum_{v \in V} x_v \quad \text{s.t.} \quad x_u + x_v \leq 1 \;\; \forall (u, v) \in E, \tag{13}$$

where edge constraints between mutually adjacent nodes are tightened into clique inequalities where possible, reducing the number of constraints below $|E|$.

Combinatorial Auction (CA). Following the arbitrary-relation bid generator of Leyton-Brown et al. (2000), we sample a set of items M and a set of bids K, where each bid $k \in K$ specifies a bundle $S_k \subseteq M$ and a price p_k, obtained by combining several sub-bids over randomly grown, overlapping item bundles ($|M| = 2{,}000$, $|K| = 4{,}000$ for S; $|M| = 4{,}000$, $|K| = 8{,}000$ for L). The winner-determination problem is

$$\max_{y \in \{0,1\}^{|K|}} \sum_{k \in K} p_k y_k \quad \text{s.t.} \quad \sum_{k \,:\, j \in S_k} y_k \leq 1 \;\; \forall j \in M, \tag{14}$$

i.e., one binary variable per bid, and one constraint per item ensuring that it is awarded to at most one winning bid.

Set Cover (SC). We sample a random binary membership matrix $A \in \{0, 1\}^{m \times n}$, with each entry drawn i.i.d. from $\mathrm{Bernoulli}(\rho)$ ($m = 5{,}000$ rows (elements) and density $\rho = 0.05$ in both sizes; $n = 1{,}000$ columns (candidate sets) for S, $n = 2{,}000$ for L), together with per-set costs c_j sampled uniformly at random. The resulting MIP instance is

$$\min_{x \in \{0,1\}^n} \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \quad Ax \geq \mathbf{1}_m, \tag{15}$$

i.e., one binary variable per candidate set, and one covering constraint per element requiring it to be covered by at least one selected set.

B.2 Synthetic Instance Statistics

Table 4: Names and the average numbers of variables, constraints, and nonzeros of the synthetic test instances.

	Small instances				Large instances			
Name	MVC-S	MIS-S	CA-S	SC-S	MVC-L	MIS-L	CA-L	SC-L
#Variables	1,000	6,000	4,000	1,000	2,000	12,000	8,000	2,000
#Constraints	65,100	24,864	2,745	5,000	135,100	49,841	5,483	5,000
#Nonzeros	130,200	48,834	19,708	250,000	270,200	97,870	39,196	500,000

B.3 MIPLIB2017 Test Instances

Table 5 lists the 35 MIPLIB2017 instances used to evaluate all baseline methods on real-world MIPs. We filter the instances according to the following criteria: (1) retain only instances labeled as "hard" on the MIPLIB website, indicating that they could not be solved within one hour; (2) exclude instances for which no initial incumbent solution can be found within 20 seconds; and (3) exclude instances that cannot be solved because of unidentified bugs in SCIP, BALANS, or other third-party packages; finally (4) exclude those that can actually be solved to optimality by SCIP or Gurobi within one hour. For each instance, we report the number of binary, general integer, and continuous variables, and the number of constraints, as read directly from the original MPS file (prior to any presolving).

Table 5: MIPLIB2017 instances used for the real-world baseline comparison, with the number of binary, general integer, and continuous variables, and the number of constraints.

Name	#Binary	#Integer	#Continuous	#Constraints
2club200v15p5scn	200	0	0	17,013
bab1	61,152	0	0	60,680
bg512142	240	0	552	1,307
bppc6-02	4,782	0	2	309
dc1c	8,380	0	1,659	1,649
dg012142	640	0	1,440	6,310
dolom1	9,720	0	1,892	1,803
ger50-17-ptp-pop-6t	0	540	4,352	545
ger50_17_trans	0	18,062	4,352	499
germany50-UUM	0	88	6,883	2,088
gmut-75-50	68,859	0	6	2,565
hgms-det	547	0	775	9,752
icir97_potential	1,235	422	455	3,314
milo-v13-4-3d-3-0	120	0	396	996
neos-3068746-nene	3,815	1,075	0	4,664
neos-3209462-rhin	383	0	56,070	170,226
neos-3352863-ancoa	20,045	0	1	169
neos-3372571-onahau	185	0	12,901	29,487
neos-5196530-nuhaka	9,792	0	4,752	42,124
neos-5261882-treska	2,730	0	170	2,971
probportfolio	300	0	20	302
queens-30	900	0	0	960
reblock354	3,540	0	0	19,906
rococoC12-010001	16,554	187	0	4,636
s1234	2,945	0	0	8,418
set3-10	1,424	0	2,595	3,747
set3-20	1,424	0	2,595	3,747
snip10x10-35r1budget17	63	0	47,548	213,801
sorrell4	2,048	0	0	504,451
swath	6,724	0	81	884
tbfp-bigm	2,404	0	2	35,999
tw-myciel4	759	1	0	8,146
usAbbrv-8-25_70	1,681	0	631	3,291
v150d30-2hopcds	150	0	0	7,822
van	192	0	12,289	27,331

B.4 Baseline Configurations

All methods are run single-threaded with the same wall-clock budget $T = 3{,}600$s. All runs use a single fixed random seed (seed=1), and each reported mean/std is taken across the 30 (or 35, for MIPLIB2017) independent test instances of a family-size combination, one run per instance. Experiments are run on 64-core CPU machines with 256GB of RAM, under Ubuntu 20.04, with SCIP 10.0.1 (via PySCIPOpt) and Gurobi 13.0.1.

SCIP-DEFAULT / SCIP-AGGR. SCIP-DEFAULT is plain PySCIPOpt `Model` with `limits/time` set to the budget and `parallel/maxnthreads`=1, with no other parameters changed from SCIP's own default heuristic schedule; SCIP-AGGR is identical, except heuristics are switched to SCIP's AGGRESSIVE parameter setting before solving.

RANDOM. Always use the mutation destroy operator at a fixed 50% destroy fraction.

ALNS (Hendel, 2022). A standalone reimplementation of Hendel's adaptive LNS bandit (reproducing the published method directly, since SCIP's own `heur_alns` plugin never leaves the root node in our setting). Reward mixing uses $\eta_1 = 0.5$, $\eta_2 = 0.8$; operator selection uses α-UCB with $\alpha = 0.0046$; the destroy-size control parameter φ is bounded in $[0.1, 0.9]$ with step 0.2 and decay

0.75, initialized at 0.9; the per-operator usage limit ν starts at 50, grows by a factor of 1.05 per use, and is capped at 5,000; RENS uses a fixed destroy fraction of 0.5.

BALANS (Cai et al., 2024). The balans python package, operator selection via `MABSelector` with Thompson Sampling and `SimulatedAnnealing` acceptance (as it is recommended by the author).

Gurobi-DEFAULT / Gurobi-HEUR. Gurobi-DEFAULT is plain gurobipy with `TimeLimit` set to the budget and `Threads`=1 and no other parameters changed; Gurobi-HEUR is identical, with `MIPFocus`=1 and `Heuristics`=0.5.

C Additional Figures

Figure 3 shows primal-gap-vs-runtime curves for Dyn-LNS and seven baselines on the eight synthetic family-size combinations.

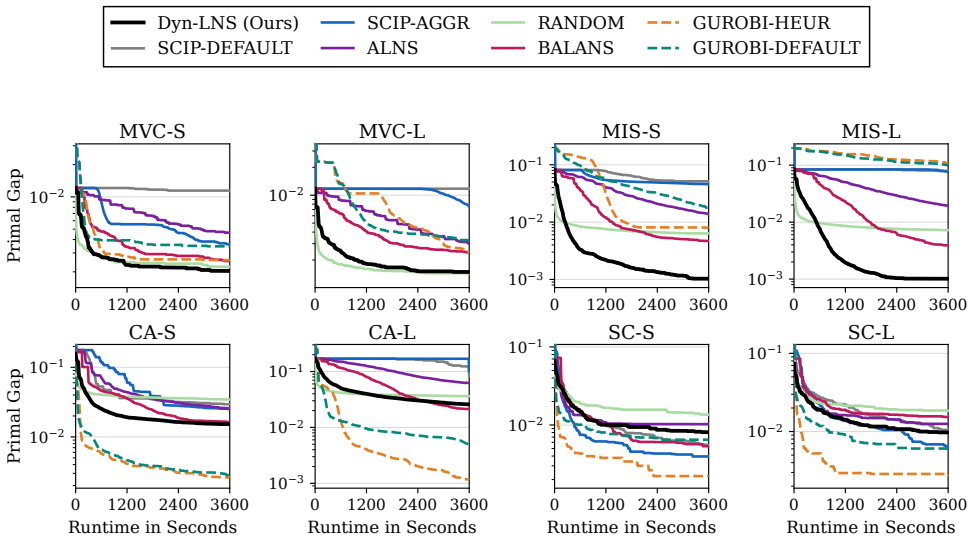


Figure 3: Mean relative primal gap vs. runtime across 30 instances per family. Gap is relative to a per-instance best-known-solution computed across all methods, floored at 10^{-3} for log-scale display.

Figure 4 shows the same style of curve for the 35 MIPLIB2017-Hard instances listed in Table 5.

D Additional Ablations on MIPLIB2017

Table 6 reports the same three ablation variants against Dyn-LNS, namely **Dyn-LNS-S**, **Dyn-LNS-R**, and **Dyn-LNS-H**, on the 35 MIPLIB2017-Hard instances, with the proxy best-known solution computed across the four arms per instance (3,600s budget). Dyn-LNS achieves the lowest mean Primal Gap and Primal Integral; Dyn-LNS-S shows the largest degradation (3.8$\times$ higher mean PG), confirming that the state-conditioned credit tables are the dominant contributor on heterogeneous real-world instances.

Figure 5 shows the corresponding primal-gap-vs-runtime curves, averaged over the 35 instances.

E Hyperparameter Sensitivity Analysis

To show that our gains stem from architectural design rather than tuning, Dyn-LNS was not subjected to deliberate hyperparameter-tuning procedures (e.g., grid search) in the main experiments; here we further test its sensitivity to its internal hyperparameters.

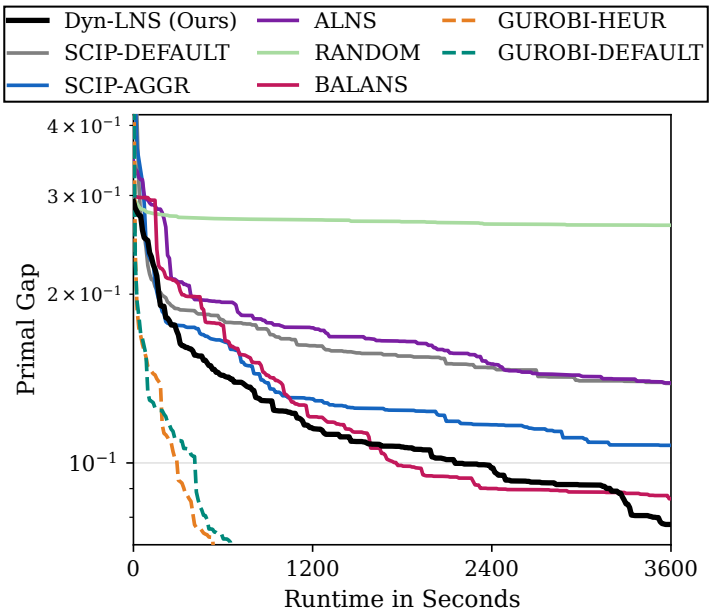


Figure 4: Mean relative primal gap vs. runtime on the 35 MIPLIB2017-Hard instances. Gap is relative to a per-instance best-known-solution computed across all methods, floored at 10^{-3} for log-scale display.

Table 6: Ablation study on MIPLIB-Hard (35 instances): Primal Gap (PG, %, lower is better) and Primal Integral (PI, lower is better) at a 3,600s cutoff, mean ± std. The proxy best-known solution is computed across the four ablation arms. Bold marks the best (lowest) value per column.

Variant	PG (%)↓	PI↓
Dyn-LNS	**1.23±3.25**	**352.4±964.8**
Dyn-LNS-S	4.63±14.90	593.6±1342.6
Dyn-LNS-R	1.64±3.89	531.7±1225.7
Dyn-LNS-H	3.98±7.37	509.9±1103.2

Setup. Both sweeps cover all four large synthetic families (CA-L, MIS-L, MVC-L, SC-L), all 30 held-out test instances per family, and the paper's full $T = 3{,}600$s budget.

Study A perturbs each of λ_d, $\lambda_\rho{=}\lambda_\tau$, N_{cold}, W_{stagn}, and β_1 independently by $\pm 50\%$ from its default value ($N_{\text{cold}}{=}40$, $W_{\text{stagn}}{=}100$, $\lambda_d{=}0.15$, $\lambda_\rho{=}\lambda_\tau{=}0.05$, $\beta_1{=}8.0$), holding all other hyperparameters fixed. **Study B** compares Dyn-LNS's default 3-stage progress / hard-binarized stagnation state against two finer alternatives, which we name analogously to the ablation variants above: **Dyn-LNS-P** (P for *progress*: 5 progress stages instead of 3) and **Dyn-LNS-C** (C for *continuous* stagnation: a soft credit update that splits weight across both stagnation-state rows in proportion to the continuous stagnation signal, instead of committing entirely to one row via hard 0/1 binarization).

Results. Table 7 reports PG and PI at each perturbation level for Study A, together with a spread indicator of PI across the three settings; Table 8 reports PG and PI for Study B. CA-L, MIS-L, and MVC-L are robust to all five perturbations. SC-L is the most sensitive family, and the default setting is not the best choice on its two most sensitive axes (λ_d, N_{cold}); even so, Dyn-LNS remains competitive on SC-L relative to other methods. The finer-granularity variants in Study B do not broadly improve on the baseline: SC-L is the only family with a clear, consistent gain, CA-L is slightly worse under both variants, and MIS-L/MVC-L are flat within noise.

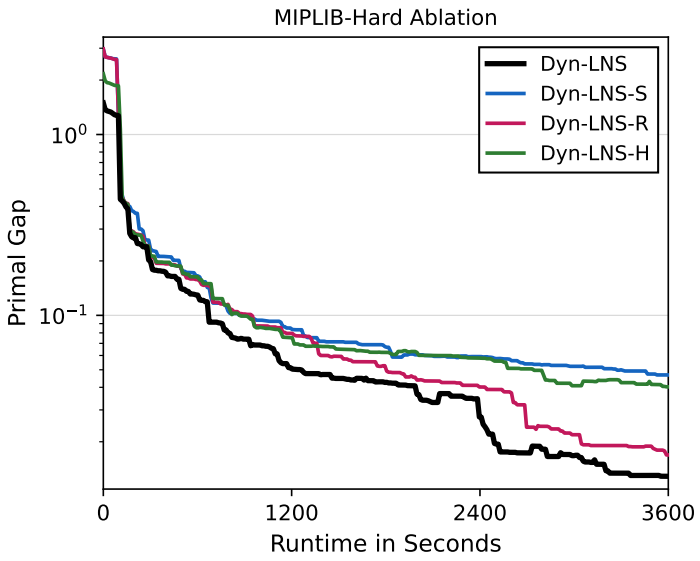


Figure 5: Mean relative primal gap vs. runtime on the 35 MIPLIB2017-Hard instances for Dyn-LNS and the three ablation variants. Gap is relative to a per-instance proxy best-known-solution computed across the four arms, floored at 10^{-3} for log-scale display.

F Intuition on Hypothesis 2

To provide empirical intuition for Hypothesis 2 (Section 3.1) that destroy type and severity can be chosen independently, we run a single-operator grid on the 30 MVC-L test instances: each instance is solved to the 3,600s budget with the destroy operator and severity level held fixed for the entire run, for every combination of the 5 destroy operators and 3 severity levels (15 fixed configurations per instance). Because (d, s) is held fixed for the whole run, we treat the end-of-budget objective under each fixed pair as a *proxy* for its quality, and ask whether this proxy decomposes additively across the destroy-type and severity axes. Let $f_{i,d,s}$ be this end-of-budget objective value of instance i under configuration (d, s). To remove the instance-scale confound, we min-max normalize within each instance across its 15 configurations,

$$y_{i,d,s} = \frac{f_{i,d,s} - \min_{d',s'} f_{i,d',s'}}{\max_{d',s'} f_{i,d',s'} - \min_{d',s'} f_{i,d',s'}} \in [0, 1], \tag{16}$$

and model $y_{i,d,s}$ with a two-way repeated-measures ANOVA,

$$y_{i,d,s} = \mu + \alpha_i + \beta_d + \gamma_s + (\beta\gamma)_{d,s} + \varepsilon_{i,d,s}, \tag{17}$$

where α_i absorbs per-instance difficulty, β_d and γ_s are the destroy-type and severity main effects, and $(\beta\gamma)_{d,s}$ is their interaction. Separate learning is justified when $H_0 \colon (\beta\gamma)_{d,s} = 0$ for all (d, s) cannot be rejected. Since the normalized objectives need not be normal, we also test the interaction with a permutation test that shuffles the 15 (d, s) labels within each instance (20,000 draws), and report the partial effect size $\eta^2_{\text{inter}} = SS_{(\beta\gamma)}/(SS_{(\beta\gamma)} + SS_{\varepsilon})$.

On MVC-L, destroy type has a strong main effect ($p_{\text{type}} < 10^{-4}$) and severity a weaker but significant one ($p_{\text{sev}} = 0.045$), while the interaction is not significant under either test ($p_{\text{inter}} = 0.824$, $p_{\text{perm}} = 0.470$) and explains a negligible share of the variation ($\eta^2_{\text{inter}} = 0.018$). Destroy type and severity thus affect solution quality largely additively, so learning them in two separate tables loses essentially no information while requiring far fewer samples per table entry.

Table 7: Study A: $\pm 50\%$ hyperparameter perturbation. Primal Gap (PG) and Primal Integral (PI) at each perturbation level (mean±std over all 30 instances). The spread column reports the relative spread of PI across the three settings, computed with $(\max - \min)/\max$.

Family	Axis		-50%	base	$+50\%$	spread (PI)
CA-L	λ_d	PG	0.0138±0.0093	0.0125±0.0092	0.0132±0.0094	
		PI	0.0291±0.0088	0.0275±0.0089	0.0289±0.0092	5.5%
	$\lambda_\rho{=}\lambda_\tau$	PG	0.0151±0.0089	0.0125±0.0092	0.0143±0.0072	
		PI	0.0297±0.0088	0.0275±0.0089	0.0294±0.0080	7.4%
	N_{cold}	PG	0.0165±0.0088	0.0125±0.0092	0.0143±0.0089	
		PI	0.0313±0.0092	0.0275±0.0089	0.0293±0.0088	12.0%
	W_{stagn}	PG	0.0148±0.0084	0.0125±0.0092	0.0148±0.0089	
		PI	0.0300±0.0089	0.0275±0.0089	0.0299±0.0089	8.1%
	β_1	PG	0.0144±0.0090	0.0125±0.0092	0.0134±0.0098	
		PI	0.0301±0.0087	0.0275±0.0089	0.0287±0.0089	8.4%
MIS-L	λ_d	PG	0.0022±0.0013	0.0023±0.0010	0.0028±0.0015	
		PI	0.0077±0.0018	0.0078±0.0016	0.0078±0.0019	1.9%
	$\lambda_\rho{=}\lambda_\tau$	PG	0.0023±0.0011	0.0023±0.0010	0.0022±0.0011	
		PI	0.0077±0.0012	0.0078±0.0016	0.0078±0.0016	1.6%
	N_{cold}	PG	0.0024±0.0013	0.0023±0.0010	0.0023±0.0010	
		PI	0.0077±0.0017	0.0078±0.0016	0.0079±0.0016	2.7%
	W_{stagn}	PG	0.0022±0.0011	0.0023±0.0010	0.0022±0.0012	
		PI	0.0080±0.0018	0.0078±0.0016	0.0079±0.0017	2.0%
	β_1	PG	0.0023±0.0012	0.0023±0.0010	0.0023±0.0013	
		PI	0.0080±0.0020	0.0078±0.0016	0.0078±0.0015	3.3%
MVC-L	λ_d	PG	0.0015±0.0009	0.0015±0.0009	0.0016±0.0010	
		PI	0.0022±0.0011	0.0022±0.0010	0.0023±0.0010	4.0%
	$\lambda_\rho{=}\lambda_\tau$	PG	0.0015±0.0007	0.0015±0.0009	0.0015±0.0008	
		PI	0.0022±0.0009	0.0022±0.0010	0.0021±0.0009	1.2%
	N_{cold}	PG	0.0016±0.0010	0.0015±0.0009	0.0017±0.0010	
		PI	0.0023±0.0011	0.0022±0.0010	0.0024±0.0011	9.2%
	W_{stagn}	PG	0.0016±0.0007	0.0015±0.0009	0.0015±0.0009	
		PI	0.0022±0.0009	0.0022±0.0010	0.0022±0.0010	1.8%
	β_1	PG	0.0016±0.0010	0.0015±0.0009	0.0017±0.0009	
		PI	0.0023±0.0011	0.0022±0.0010	0.0024±0.0012	8.6%
SC-L	λ_d	PG	0.0076±0.0073	0.0102±0.0095	0.0070±0.0079	
		PI	0.0132±0.0078	0.0157±0.0101	0.0139±0.0082	16.1%
	$\lambda_\rho{=}\lambda_\tau$	PG	0.0096±0.0077	0.0102±0.0095	0.0087±0.0091	
		PI	0.0148±0.0087	0.0157±0.0101	0.0147±0.0089	5.9%
	N_{cold}	PG	0.0109±0.0107	0.0102±0.0095	0.0085±0.0100	
		PI	0.0163±0.0115	0.0157±0.0101	0.0136±0.0097	16.3%
	W_{stagn}	PG	0.0114±0.0102	0.0102±0.0095	0.0108±0.0106	
		PI	0.0170±0.0106	0.0157±0.0101	0.0159±0.0106	7.9%
	β_1	PG	0.0092±0.0095	0.0102±0.0095	0.0091±0.0101	
		PI	0.0156±0.0100	0.0157±0.0101	0.0153±0.0104	2.7%

Table 8: Study B: state-granularity comparison. Primal Gap (PG) and Primal Integral (PI), mean±std over all 30 instances. Only SC-L shows a consistent improvement from finer granularity; CA-L is slightly worse under both variants, and MIS-L/MVC-L are flat.

Config		CA-L	MIS-L	MVC-L	SC-L
Dyn-LNS	PG	0.0125±0.0092	0.0023±0.0010	0.0015±0.0009	0.0102±0.0095
	PI	0.0275±0.0089	0.0078±0.0016	0.0022±0.0010	0.0157±0.0101
Dyn-LNS-P	PG	0.0136±0.0093	0.0021±0.0013	0.0015±0.0008	0.0070±0.0085
	PI	0.0296±0.0092	0.0080±0.0018	0.0022±0.0010	0.0153±0.0100
Dyn-LNS-C	PG	0.0145±0.0088	0.0020±0.0013	0.0014±0.0006	0.0072±0.0073
	PI	0.0292±0.0086	0.0078±0.0018	0.0022±0.0008	0.0137±0.0090