PACE: Partial-state Amortized Constraint Editing for Neural Combinatorial Optimization

Bohao Li[1] **Chenhao Yuan**[1] **Pei He**[1] **Ying Li**[1,2] **Yangming Guo**[1†]
[1]Northwestern Polytechnical University
[2]Ningbo Institute of Northwestern Polytechnical University
{bh_li, yuanchenhao, hepei002, liying_npu}@mail.nwpu.edu.cn
yangming_g@nwpu.edu.cn

Abstract

We propose **PACE**, **P**artial-state **A**mortized **C**onstraint **E**diting, which treats task-native feasible partial states as the common semantic object for neural combinatorial optimization, unifying edit learning, constraint-preserving editing, anytime closure, and test-time scaling across routing and graph combinatorial optimization. Existing neural combinatorial optimization methods expose complementary strengths: constructive and adaptive expansion solvers keep meaningful partial solutions but couple them to serial or method-specific growth, while global prediction, diffusion, masked reconstruction, and complete-solution refinement provide scalable guidance yet often refine heatmaps, noisy solutions, reconstruction targets, or perturbed full outputs. These semantics make learning, search, and test-time scaling act on the same state object. PACE learns amortized constraint editing by ranking candidate edits conditioned on the current feasible partial state; a constraint-preserving transition Γ_I commits only edits that preserve legal continuation and extendability; and task-native closure Comp_I maps any extendable intermediate feasible partial state to a task-terminal feasible output. Budgeted refinement then spends extra inference-time compute through deeper edit steps, additional refinement rounds, or broader candidate sets along the same trajectory. We instantiate these semantics across TSP, ATSP, CVRP, MIS, MVC, MCL, and MCut, covering edge-oriented routing and node-oriented graph combinatorial optimization. Theoretical guarantees are structural rather than optimality claims, covering task-wise soundness, constructive completion, state-space closure in the extendable subset, terminal feasible completion at any editing depth, and elite-pool nondegradation. Empirically, PACE is competitive and budget-controllable, including ATSP-500 at 0.247% Drop, MIS RB-[800-1200] at 2.00%, and MVC at 0.07%. The codes are publicly available at `https://github.com/53mins/PACE`.

1 Introduction

Combinatorial optimization lies behind routing [1, 2, 3], allocation [4, 5, 6], scheduling [7, 8], and graph reasoning [9, 10, 11], and neural combinatorial optimization has progressed from learned constructive heuristics to global prediction and diffusion-based solvers [1, 2, 12]. Yet a basic representational question remains unresolved: what intermediate object should a solver inhabit while additional inference-time compute is being spent? Local construction preserves feasibility and clear action semantics, but it ties the solver state to a serial decoding order. Global prediction offers broader parallel guidance, yet its intermediate objects are often heatmaps or noisy full-solution trajectories whose constraint meaning is less stable under discrete refinement.

†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

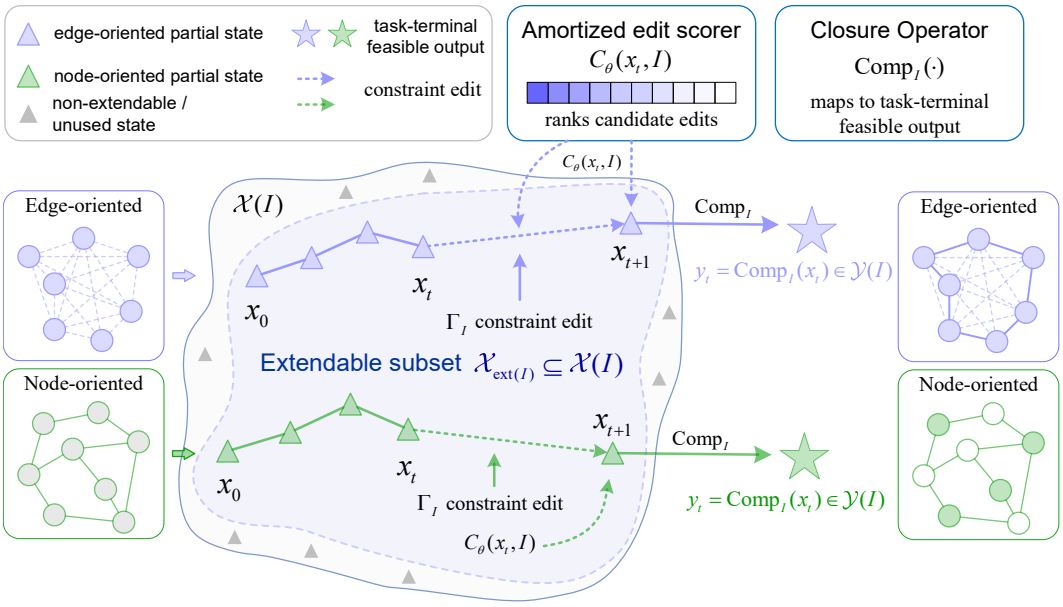


Figure 1: Overview of PACE. A feasible partial state is edited by an amortized scorer, filtered by a constraint-preserving transition, and closed into a task-terminal feasible output across edge-oriented routing tasks and node-oriented graph combinatorial optimization tasks.

Recent work has advanced this frontier from several directions. Search-based test-time solvers [13, 14, 15] refine complete candidates through learned search. Masked reconstruction [16] extracts dense local supervision from reference solutions and can also drive iterative improvement. Adaptive expansion [17, 18] shows that partial decisions can be grown under structural control. Together these results show that neural CO increasingly depends on how learning and extra computation interact during solving. What remains missing are shared feasible partial-state semantics that stay task-native across training, search, and controlled test-time scaling.

PACE starts from a simple thesis: The right semantic object for neural combinatorial optimization is not a complete solution, but a feasible partial state. PACE, **P**artial-state **A**mortized **C**onstraint **E**diting, operationalizes this idea by defining for each task a feasible partial state that records committed structure while leaving unresolved regions open. The solver state is therefore organized around extendability to at least one terminal feasible output, giving learning and search a common object whose meaning is determined by the combinatorial structure itself.

On top of these semantics, PACE formulates solving as amortized constraint editing. Conditioned on the current feasible partial state, the model scores candidate edits according to their value if committed next. A task-native constraint-preserving transition then turns high-scoring edits into legal state updates and keeps the trajectory inside the extendable region. Because every such state remains closable, a lightweight closure routine can return a terminal feasible output at any editing depth.

This organization also makes budgeted refinement the native mechanism for inference-time scaling. More outer rounds, deeper inner refinement, or broader candidate exploration all spend extra compute within the same state process. The theoretical center of PACE is therefore structural: extendability of partial states, soundness of accepted edits, constructive completion, and anytime closing. These guarantees explain why continued refinement can remain aligned with task semantics instead of switching to a separate search logic.

The same semantic contract applies across edge-oriented routing and node-oriented graph combinatorial optimization tasks. In TSP, ATSP, and CVRP, a feasible partial state records a route skeleton whose committed edges, arcs, or chains preserve the conditions required for later closure. In MIS, MVC, MCL, and MCut, it records selected vertices or partial assignments that remain consistent while leaving the rest of the instance open. These tasks differ in constraint geometry, but each instantiates the same state-editing framework: score edits on the current state, accept only legal commitments, and close the state into a feasible terminal output when needed. Fig. 1 summarizes

this state-editing loop and its instantiations across edge-oriented routing and node-oriented graph combinatorial optimization tasks.

Based on this viewpoint, PACE asks whether coherent feasible partial-state semantics can organize training, search, and refinement across seven representative combinatorial optimization tasks while benefiting from larger inference-time budgets. Our contributions are threefold: (1) we identify feasible partial states as the common semantic object for edge-oriented routing and node-oriented graph combinatorial optimization; (2) we formulate Partial-state Amortized Constraint Editing, where edit ranking, the constraint-preserving transition Γ_I, and closure Comp_I turn scored commits into legal updates and terminal outputs; and (3) we develop budgeted refinement as an inference-time scaling mechanism, instantiate it on TSP, ATSP, CVRP, MIS, MVC, MCL, and MCut, and support it with structural guarantees and empirical evaluation.

2 Related Work

2.1 Neural Constructive and Global-Prediction Solvers

Autoregressive and other constructive neural solvers model solving as committed decision sequences, making the intermediate state explicit and task-aware but tied to serial decoding. Attention Model [1] and POMO [2] are canonical edge-oriented routing examples; BQ-NCO [19] and PolyNet [20] revisit construction through symmetry-aware state reduction and diverse policies; and Li et al. [3] and UDC [21] keep learned components in larger decomposition or reunification loops. In contrast, global-prediction methods such as NeuroLKH [22] and DIFUSCO [12] provide broad edge or graph-variable guidance before hard decoding, heuristics, or diffusion search. These paradigms expose a tradeoff between local feasibility semantics and parallel full-solution guidance. PACE keeps the former without using a decoder prefix, subproblem queue, or soft full-solution proxy: the feasible partial state is the semantic object itself.

2.2 Partial-Solution Refinement, Masked Reconstruction, and Test-Time Search

Another thread treats neural CO as iterative refinement, either by editing feasible complete candidates or reconstructing selected solution parts. L2I [23] and DACT [24] learn improvement operators over complete solutions. T2T [13] and Fast T2T [14] pair generative modeling with test-time gradient search, while GenSCO [15] studies test-time scaling for diffusion-based solvers. MaskCO [16] turns one reference solution into many masked local training signals and also supports iterative inference. These works show that neural CO quality increasingly depends on how learned models use additional solve-time compute. PACE shares that emphasis, but its core state is not a noisy complete candidate or a reconstruction target. Instead, amortized constraint editing, the constraint-preserving transition, budgeted refinement, and closure all act on the same feasible partial state, which is what gives PACE a native anytime mechanism through state closing.

2.3 Unified, Scalable, and Benchmark-Driven ML4CO

Recent ML4CO work has also pushed toward unification in models, evaluation, and scale. CO-Expander [17] and NEXCO [18] are especially close to PACE because they give partial decisions substantive algorithmic meaning and grow them under structural control rather than fixed one-shot or strictly autoregressive decoding; their mechanisms, however, remain coupled to particular predictors, diffusion processes, or expansion schedules. In parallel, GOAL [25] studies a generalist backbone across heterogeneous combinatorial problems, while ML4CO-Bench-101 [26] and FrontierCO [27] emphasize unified taxonomies, reproducible comparisons, and broader-scale evaluation. PACE builds on this movement by elevating the feasible partial state into the common semantic object, so learning, search, budgeted refinement, and test-time scaling share one state semantics across edge-oriented routing and node-oriented graph combinatorial optimization.

3 Preliminaries

Combinatorial Optimization on Graphs. We consider graph-structured instances I from seven combinatorial optimization tasks: TSP, ATSP, CVRP, MIS, MVC, MCL, and MCut, following

standard ML4CO formulations on edge-oriented and node-oriented graph combinatorial optimization [11, 17, 18]. Each instance is specified by a graph $G_I = (V_I, E_I)$, possibly directed, together with task-dependent attributes such as edge costs, a depot, and customer demands. A terminal decision y is task-dependent: a Hamiltonian cycle for TSP, a Hamiltonian directed cycle for ATSP, a depot-rooted capacity-feasible route set covering all customers for CVRP, an independent set for MIS, a vertex cover for MVC, a clique for MCL, and a complete binary cut assignment for MCut.

Let Ω_I denote the classical feasible set of the underlying combinatorial optimization task. Edge-oriented problems use edge or arc incidence variables, MIS, MVC, and MCL use node indicators, and MCut uses binary side assignments. We minimize cost for TSP, ATSP, CVRP, and MVC, and maximize size or cut value for MIS, MCL, and MCut. For uniform notation, $\tilde{f}_I$ denotes the minimized objective, namely the original objective for TSP, ATSP, CVRP, and MVC, $-|y|$ for MIS and MCL, and $-\mathrm{cut}(y)$ for MCut. The classical problem is

$$y^\star \in \arg\min_{y \in \Omega_I} \tilde{f}_I(y). \tag{1}$$

Task-Native Partial States and Extendability. PACE separates the classical feasible set from the task-terminal output semantics used by the solver. For an instance I, Ω_I remains the classical feasible set in the optimization problem above, whereas $\mathcal{Y}(I)$ denotes the task-terminal feasible output set used by closure and theory. We have $\mathcal{Y}(I) = \Omega_I$ for TSP, ATSP, CVRP, MVC, and MCut. For MIS and MCL, however, $\mathcal{Y}(I)$ consists of maximal independent sets and maximal cliques. Hence a classically feasible independent set or clique need not be task-terminal.

This terminal restriction does not change the underlying optimization problem for MIS or MCL. Every feasible independent set or clique can be greedily extended to a maximal one, and the normalized objective $\tilde{f}_I(y) = -|y|$ cannot increase under such an extension. Hence, for I corresponding to MIS or MCL,

$$\min_{y \in \mathcal{Y}(I)} \tilde{f}_I(y) = \min_{y \in \Omega_I} \tilde{f}_I(y). \tag{2}$$

Thus maximality is used only to define task-terminal solver outputs and closure behavior, not to replace maximum independent set or maximum clique with a different objective. PACE evaluates returned outputs with the same $\tilde{f}_I$, now restricted to $\mathcal{Y}(I)$.

For each task, PACE defines an ambient task-native partial-state space $\mathcal{X}(I)$. Its elements record committed structure while leaving unresolved decisions open: route skeletons for TSP, ATSP, and CVRP, selected-set states for MIS, MVC, and MCL, and partial side assignments for MCut. We write $x \preceq_I y$ when a task-terminal output $y \in \mathcal{Y}(I)$ preserves every commitment already stored in x. For edge-oriented problems, this means preserving committed edges, arcs, or customer-chain edges; for MIS, MVC, and MCL, it means containing all committed selected vertices; for MCut, it means agreeing with every committed side assignment. The extendable subset is

$$\mathcal{X}_{\mathrm{ext}}(I) = \{x \in \mathcal{X}(I) : \exists y \in \mathcal{Y}(I),\ x \preceq_I y\}. \tag{3}$$

We refer to elements of $\mathcal{X}_{\mathrm{ext}}(I)$ as feasible partial states. The ambient space $\mathcal{X}(I)$ may contain non-extendable states, but PACE is defined on feasible partial states.

Constraint-Preserving Transition, Closure, and Budgeted Refinement. PACE formulates solving as amortized constraint editing over feasible partial states. Given $x \in \mathcal{X}_{\mathrm{ext}}(I)$, the model produces a scored candidate list $C_\theta(x, I)$ over task-native edits. The constraint-preserving transition Γ_I scans this list in priority order and commits only edits that satisfy the task-native feasibility predicate at the current scan state. With commit budget m_t,

$$x_{t+1} = \Gamma_I(x_t, C_\theta(x_t, I), m_t). \tag{4}$$

The full scan-state recursion is task-specific and stated later; later sections give task-wise conditions under which repeated application of Γ_I keeps the trajectory inside $\mathcal{X}_{\mathrm{ext}}(I)$.

Closure is given by a task-native completion map

$$\mathrm{Comp}_I : \mathcal{X}_{\mathrm{ext}}(I) \to \mathcal{Y}(I), \qquad x \preceq_I \mathrm{Comp}_I(x). \tag{5}$$

For edge-oriented routing tasks, Comp_I closes the current skeleton into a full tour or route set; for MIS and MCL, it saturates the current set to maximality; for MVC and MCut, it completes the

remaining uncovered or unassigned part. In particular, task-terminal states are fixed points of closure. Budgeted refinement refers to iterating Γ_I under finite solve-time compute and invoking closure at any depth. Starting from $x_0 \in \mathcal{X}_{\text{ext}}(I)$, PACE can return an anytime task-terminal output

$$y_t = \text{Comp}_I(x_t) \tag{6}$$

after any number of edits. Later sections instantiate this budget through inner edit steps and outer refinement rounds, but extra compute always extends the same trajectory of feasible partial states.

4 The PACE Framework: Partial-State Amortized Constraint Editing

With the objects of Sec. 3 fixed, PACE combines an edit scorer trained on feasible partial states, Γ_I as the legal commit rule, and Comp_I for anytime closing. This section describes the training objective, the solve-time transition, and the budgeted solver.

4.1 Amortized Edit Learning from Feasible Partial States

Training starts from reference terminal outputs $y^{\text{ref}} \in \mathcal{Y}(I)$ and derives many feasible partial states $x \in \mathcal{X}_{\text{ext}}(I)$ satisfying $x \preceq_I y^{\text{ref}}$. Each such state records the currently committed structure together with an open frontier of admissible edits. Conditioned on this feasible partial state, PACE scores candidate commits and learns a ranking rule for the next edit.

Writing the scalar score inside $C_\theta(x, I)$ as $s_\theta(c; x, I)$, we convert scores to a softmax distribution over a task-dependent normalization set $\mathcal{M}_I(x)$. For a reference y^{ref}, let $\mathcal{F}_I^+(x; y^{\text{ref}}) \subseteq \mathcal{M}_I(x)$ denote the feasible candidate edits consistent with y^{ref}. The core ranking term is

$$p_\theta(c \mid x, I) = \frac{\exp(s_\theta(c; x, I))}{\sum_{c' \in \mathcal{M}_I(x)} \exp(s_\theta(c'; x, I))}. \tag{7}$$

$$\mathcal{L}_{\text{edit}}(x, y^{\text{ref}}) = -\frac{1}{|\mathcal{F}_I^+(x; y^{\text{ref}})|} \sum_{c \in \mathcal{F}_I^+(x; y^{\text{ref}})} \log p_\theta(c \mid x, I). \tag{8}$$

Across many feasible partial states derived from references, this objective amortizes online edit evaluation into a reusable ranking function. At test time, the model applies that ranking function to score candidate commits under the current feasible partial state. Task-specific calibration losses may be added, but the shared training signal remains supervision on feasible partial states for edit priority.

4.2 Constraint-Preserving Editing and State Closing

At solve time, $C_\theta(x_t, I)$ is sorted into an ordered list $C_t = (c_{t,1}, \ldots, c_{t,L_t})$. The constraint-preserving transition Γ_I scans candidates in priority order, maintains the current scan state, and accepts at most m_t edits that satisfy the task-native feasibility predicate when they are examined. Each accepted edit is committed immediately before the next candidate is checked. This sequential screening makes feasibility depend on the evolving state and places the task-specific constraint geometry inside the transition itself.

Because acceptance is evaluated after every committed edit, Γ_I converts a scored list into a legal state update while keeping the trajectory inside the extendable subset $\mathcal{X}_{\text{ext}}(I)$. After any number of accepted edits, closure maps the current feasible partial state to a task-terminal feasible output through Comp_I and preserves all committed structure as in Eq. (5). Task-terminal states are fixed points of closure. The resulting task-wise soundness, constructive completion, and anytime closing guarantees are formalized in Secs. B.2 to B.4.

4.3 Budgeted Refinement Solver

Budgeted refinement is the solver used at inference in PACE. The inner loop starts from one or more feasible partial states, applies scored editing for E_s steps, and closes each resulting state into a task-terminal feasible output. The outer loop retains an elite pool of closed outputs, reopens selected elites into feasible partial states, and feeds them through the same inner editing dynamics. Algorithm 1 summarizes the full procedure.

Increasing E_s consolidates more committed structure before closing, while increasing R adds more rounds of elite reopening and refinement. In both cases, extra compute continues the same feasible partial-state process. The reopening policy only needs to return states in $\mathcal{X}_{\mathrm{ext}}(I)$ consistent with selected elites, so empty starts and reopened starts share the same semantics. With elite retention, the best task-terminal feasible output in the pool is nondegrading across outer rounds; the corresponding proof is given in Sec. B.6.

Algorithm 1: PACE budgeted refinement solver.

Input: instance I, outer rounds R, inner edit steps E_s, commit budgets $\{m_t\}_{t=0}^{E_s-1}$, pool size K, starts B
Initialize: a multistart collection $\mathcal{S}$ of B empty feasible partial states and $\mathcal{P} \leftarrow \emptyset$
for $r = 1, \ldots, R$ **do**
  $\widetilde{\mathcal{P}} \leftarrow \emptyset$
  for all $x_0 \in \mathcal{S}$ **do**
    $x \leftarrow x_0$
    for $t = 0, \ldots, E_s - 1$ **do**
      form a candidate list C_t ordered by score from $C_\theta(x, I)$
      $x \leftarrow \Gamma_I(x, C_t, m_t)$
    end for
    $y \leftarrow \mathrm{Comp}_I(x)$
    $\widetilde{\mathcal{P}} \leftarrow \widetilde{\mathcal{P}} \cup \{y\}$
  end for
  retain the best K outputs under $\tilde{f}_I$
  $\mathcal{P} \leftarrow \mathrm{TopK}(\mathcal{P} \cup \widetilde{\mathcal{P}}, K)$
  if $r < R$ **then**
    build a new multistart collection $\mathcal{S}$ by reopening elites in $\mathcal{P}$
  end if
end for
Output: the best $y \in \mathcal{P}$ under $\tilde{f}_I$

4.4 Task-wise State, Edit, and Closure Rules

Across all seven tasks, differences enter only through the state representation in $\mathcal{X}(I)$, the edit type scored by $C_\theta(x, I)$, the task-native feasibility predicate inside Γ_I, and the closing rule Comp_I. Edge-oriented routing tasks use path or customer-chain skeletons with degree, cycle, direction, and capacity checks; node-oriented graph tasks use selected-set or partial assignment states with independence, monotonicity, clique-consistency, or assignment-consistency checks. The solver loop remains unchanged: score task-native edits on the current feasible partial state, commit only legal edits, and close the state whenever a task-terminal output is needed. Task-wise choices, checks, and closures are detailed in Sec. A, with soundness and closure arguments in Secs. B.2 and B.3.

4.5 Scorer Architecture

PACE keeps the neural component limited to edit scoring. For a feasible partial state x, the scorer returns logits over task native candidates,

$$C_\theta(x, I) = \{s_\theta(c; x, I) : c \in \mathcal{M}_I(x)\}. \tag{9}$$

These logits define the ranking loss in Eq. (8) and the candidate order used by Γ_I. Feasibility checking and terminal completion are still handled by the task operators Γ_I and Comp_I.

We use Transformer scorers that combine instance features with an encoding of the current feasible partial state. Their attention layers use additive biases for task structure and committed state information:

$$\mathrm{Attn}(Q, K, V; x, I) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}} + B_I(x)\right) V. \tag{10}$$

Routing tasks produce pairwise edge or arc logits, with CVRP scoring customer edges and leaving depot attachments mainly to closure. Graph tasks produce node logits for MIS, MVC, and MCL, and label logits for MCut. Detailed state encodings, edit heads, and model settings are given in Sec. C.3.

5 Experiments

5.1 Datasets and Metrics

We evaluate PACE on seven tasks spanning edge-oriented routing and node-oriented graph combinatorial optimization. The edge-oriented benchmarks include uniform Euclidean TSP-50/100/500/1000, full directed cost-matrix ATSP-50/100/200/500, and uniform CVRP-50/100/200/500. The node-oriented benchmarks include RB and ER instances for MIS, SATLIB for MIS transfer evaluation, RB instances for MCL and MVC, and BA instances for MCut, following ML4CO-Bench-101 [26]. Reference objectives are produced by Concorde for TSP, LKH for ATSP, HGS for CVRP, KaMIS for MIS, and Gurobi for MCL, MVC, and MCut.

Table 1: Results on TSP across problem sizes. * denotes reported results from prior work.

Method	TSP-50			TSP-100			TSP-500			TSP-1000		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
Classical Reference Solvers												
Concorde* [28]	5.69	–	0.06s	7.76	–	0.24s	16.55	–	18.67s	23.12	–	84.41s
LKH (500)* [29]	5.69	0.00%	0.06s	7.76	0.00%	0.18s	16.55	0.00%	1.85s	23.12	0.01%	4.64s
Learning-based Solvers with Direct Decoding												
GCN* [30]	5.69	0.07%	0.01s	7.78	0.24%	0.01s	16.77	1.35%	0.06s	23.53	1.77%	0.23s
DIMES* [31]	5.89	3.58%	0.01s	8.11	4.54%	0.01s	17.66	6.71%	0.31s	24.91	7.74%	0.66s
LEHD PRC 100* [32]	–	–	–	7.76	0.01%	0.64s	16.61	0.34%	3.75s	23.44	1.22%	20.16s
RL4CO (SymNCO)* [33]	5.73	0.68%	0.17s	7.89	1.75%	0.32s	–	–	–	–	–	–
BQ-NCO + Beam-16* [19]	5.79	1.72%	0.24s	7.87	1.48%	0.90s	16.77	1.33%	4.02s	23.51	1.71%	10.34s
Learning-based Solvers with Test-Time Scaling												
DIFUSCO (T_s=50) [12]	5.69	0.13%	0.36s	7.78	0.28%	0.47s	16.82	1.64%	1.14s	23.57	1.94%	4.01s
T2T (T_s=50,T_g=30) [13]	5.69	0.02%	1.09s	7.76	0.07%	1.43s	16.68	0.82%	3.10s	23.44	1.40%	9.38s
Fast T2T (T_s=5,T_g=5) [14]	5.69	0.01%	0.29s	7.76	0.03%	0.33s	16.61	0.39%	1.50s	23.25	0.58%	6.18s
StruDiCO (T_s=3/5,T_g=3/5)* [34]	5.69	0.01%	0.06s	7.76	0.02%	0.08s	16.60	0.33%	0.67s	23.24	0.54%	2.61s
COExpander (D_s=3,T_s=5) [17]	5.69	0.02%	0.11s	7.76	0.04%	0.20s	16.63	0.52%	0.62s	23.34	0.95%	2.30s
NEXCO (D_s=3)* [18]	–	–	–	7.76	0.04%	0.05s	16.61	0.39%	0.23s	23.31	0.85%	0.91s
NEXCO (D_s=7)* [18]	–	–	–	7.76	0.02%	0.11s	16.59	0.25%	0.43s	23.24	0.52%	1.68s
PACE (E_s=10,R=1)	5.69	0.001%	0.01s	7.76	0.005%	0.02s	16.63	0.525%	0.20s	23.25	0.566%	0.63s
PACE (E_s=20,R=1)	5.69	0.001%	0.02s	7.76	0.004%	0.02s	16.62	0.459%	0.30s	23.23	0.482%	1.14s
PACE (E_s=50,R=1)	5.69	0.001%	0.03s	7.76	0.003%	0.04s	16.61	0.406%	0.60s	23.22	0.429%	2.66s
PACE (E_s=20,R=5)	5.69	0.001%	0.15s	7.76	0.003%	0.32s	16.59	0.251%	5.93s	23.18	0.274%	22.68s

Table 2: Results on ATSP and CVRP. * denotes reported results from prior work.

Method	ATSP-50			ATSP-100			ATSP-200			ATSP-500		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
LKH (1000)* [29]	1.55	0.00%	0.10s	1.57	0.00%	0.24s	1.56	0.00%	0.72s	1.57	0.00%	4.38s
MatNet ($\times$16)* [35]	1.56	0.30%	0.04s	1.59	1.58%	0.07s	3.73	138.40%	0.16s	–	–	–
GOAL + Beam-16* [25]	1.63	4.74%	0.35s	1.62	3.53%	0.91s	1.61	2.86%	4.69s	1.70	8.13%	32.24s
COExpander (S=4,D_s=3,T_s=5) [17]	1.56	0.17%	0.20s	1.58	0.95%	0.71s	1.59	1.50%	2.64s	1.60	1.57%	16.48s
PACE (E_s=15,R=1)	1.558	0.245%	0.05s	1.573	0.438%	0.19s	1.593	1.794%	0.92s	1.578	0.313%	4.63s
PACE (E_s=50,R=1)	1.558	0.194%	0.15s	1.572	0.360%	0.57s	1.590	1.626%	2.47s	1.577	0.247%	14.13s
Method	**CVRP-50**			**CVRP-100**			**CVRP-200**			**CVRP-500**		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
HGS* [36]	10.37	0.00%	1.01s	15.56	0.00%	20.03s	19.63	0.00%	60.02s	37.15	0.00%	360.38s
RL4CO (SymNCO)* [33]	10.56	1.91%	0.09s	15.93	2.38%	0.17s	20.19	2.88%	0.34s	38.70	4.17%	0.88s
COExpander (D_s=3) [17]	10.77	3.90%	0.04s	16.22	4.25%	0.06s	20.59	4.89%	0.15s	39.12	5.34%	0.76s
NEXCO (D_s=3)* [18]	10.48	1.12%	0.04s	15.83	1.73%	0.08s	20.18	2.76%	0.27s	–	–	–
NEXCO (D_s=5)* [18]	10.46	0.85%	0.06s	15.78	1.40%	0.12s	20.11	2.45%	0.39s	–	–	–
PACE (E_s=20,R=1)	10.497	1.226%	0.03s	15.772	1.343%	0.09s	20.146	2.631%	0.58s	38.508	3.642%	2.52s
PACE (E_s=40,R=1)	10.494	1.199%	0.04s	15.744	1.162%	0.11s	20.116	2.476%	0.61s	38.339	3.189%	2.68s

We report three metrics: Obj. is the average task objective, Drop is the relative deviation from the reference objective, and Time is the average inference time per instance. For minimization tasks, Drop is computed from excess objective; for maximization tasks, Drop is computed from the loss in objective, so negative Drop indicates that the method exceeds the time-limited reference. Unless otherwise stated, for TSP and ATSP, learning-based entries are compared after the standard 2-Opt post-processing step. Training settings, model configurations, and test instance counts are summarized in Secs. C.1 and C.2 and Tables 9 and 10.

5.2 Main Results

TSP. As shown in Table 1, PACE reaches near-zero Drop on TSP-50/100 and improves TSP-1000 from 0.566% at E_s=10,R=1 to 0.274% with R=5. The E_s=50,R=1 setting already improves over Fast T2T and StruDiCO on TSP-1000 with 0.429% Drop, while larger R lowers Drop to 0.274% when more compute is available. These results support budgeted refinement rather than a fixed one-shot route decoder.

Table 3: Results on MIS. *: reported results from prior work; †: uses a larger training set.

Method	RB-[200-300]			RB-[800-1200]			ER-[700-800]		
	Obj.↑	Drop↓	Time↓	Obj.↑	Drop↓	Time↓	Obj.↑	Drop↓	Time↓
KaMIS* [37]	20.09	–	45.81s	43.00	–	56.97s	44.97	–	60.75s
Gurobi* [38]	20.09	0.00%	0.54s	42.19	1.83%	33.84s	38.78	13.75%	60.49s
GFlowNets* [39]	19.18	4.57%	0.06s	37.48	13.14%	0.52s	41.14	8.53%	0.35s
DIFUSCO (S=4,T_s=50)* [12]	–	–	–	–	–	–	40.97	8.89%	5.45s
Fast T2T (T_g=5,T_s=5)* [14]	19.50	2.89%	0.41s	–	–	–	40.69	9.51%	1.03s
StruDiCO (T_s=5,T_g=5)* [34]	19.75	1.71%	0.18s	–	–	–	42.13	6.32%	0.55s
COExpander (S=4,D_s=20) [17]	19.71	1.87%	0.35s	41.44	3.58%	3.82s	42.56	5.34%	2.05s
NEXCO (D_s=7)*† [18]	19.76	1.66%	0.14s	41.25	4.07%	1.00s	42.98	4.20%	0.56s
PACE (E_s=100,R=1)	19.85	1.14%	0.05s	41.97	2.36%	0.20s	43.24	3.83%	0.24s
PACE (E_s=500,R=1)	19.85	1.14%	0.05s	42.09	2.09%	0.29s	43.27	3.78%	0.31s
PACE (E_s=500,R=5)	19.86	1.09%	0.23s	42.13	2.00%	2.22s	43.34	3.61%	1.83s

ATSP and CVRP. Table 2 extends the same edge-oriented state semantics to asymmetric costs and capacity constraints. On ATSP-500, PACE reaches 0.247% Drop in 14.13s, compared with 1.57% in 16.48s for COExpander; on CVRP-500, PACE reports 3.189% Drop, compared with 5.34% for COExpander and 4.17% for RL4CO. This routing coverage indicates that amortized constraint editing is not tied to symmetric edge selection.

MIS. For MIS, Tables 3 and 4 show that PACE improves the neural quality-time tradeoff, with the full budget reaching 2.00% Drop on RB-[800-1200] versus 3.58% for COExpander and 4.07% for NEXCO. On SATLIB, PACE obtains 0.18% Drop in 0.16s, matching the best neural quality range while being substantially faster. The largest gains appear in the high-scale RB regime, where the selected-set partial state gives the scorer useful context while closure preserves maximal independent-set feasibility.

Table 4: Results on SATLIB. * denotes reported results from prior work.

Method	Obj.↑	Drop↓	Time↓
KaMIS* [37]	425.95	0.00%	24.37s
Gurobi* [38]	425.92	0.01%	3.95s
GFlowNets* [39]	423.54	0.57%	2.79s
DIFUSCO (S=4,T_s=50)* [12]	425.11	0.20%	2.96s
Fast T2T (S=4,T_g=5,T_s=5)* [14]	425.00	0.25%	2.67s
COExpander (D_s=20) [17]	425.05	0.22%	0.55s
PACE (E_s=10,R=1)	424.88	0.25%	0.10s
PACE (E_s=50,R=1)	425.18	0.18%	0.16s

Budget response. The heatmaps in Fig. 2 make the inference budget semantics explicit: increasing E_s and R moves TSP-1000 from about 0.96% to 0.20% Drop and MVC RB-LARGE from 3.6% to 0.062%. The smooth decrease across budgets shows that the constraint-preserving transition and closure give PACE a controllable inference-time scaling axis rather than a brittle post-processing knob.

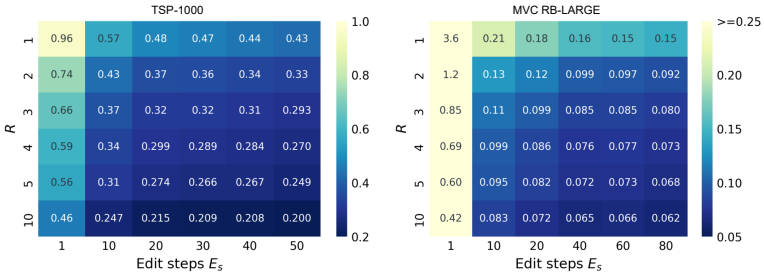


Figure 2: Effect of edit and refinement budgets. Cells show Drop (%) over E_s and R.

MCL, MVC, and MCut. Table 5 completes the graph evaluation beyond independent set: PACE reaches 2.96% Drop on MCL RB-LARGE, 0.07% on both MVC scales, and negative Drop on MCut at larger budgets. These results show that the same node-oriented state semantics covers clique, cover, and cut objectives, with MVC giving the strongest graph gains and MCut sometimes exceeding the time-limited Gurobi reference.

Generalization and coverage. Across the edge-oriented and node-oriented results, the seven tasks demonstrate that shared feasible partial-state semantics work across edge-oriented routing and node-oriented combinatorial optimization. Additional cross-scale and cross-distribution behavior is reported in Tables 12 to 16.

Table 5: Results on MCL, MVC, and MCut. * denotes reported results from [17].

Maximum Clique (MCL)	RB-SMALL			RB-LARGE		
	Obj.↑	Drop↓	Time↓	Obj.↑	Drop↓	Time↓
Gurobi* [38]	19.08	0.00%	0.90s	40.18	0.00%	276.66s
Meta-EGN* [40]	17.51	8.30%	0.27s	33.79	15.49%	0.54s
DiffUCO: CE (S=4,F=1)* [41]	16.21	12.53%	1.41s	–	–	–
COExpander (S=1,D_s=20,T_s=1) [17]	18.77	1.89%	0.11s	36.76	8.80%	0.41s
COExpander (S=4,D_s=5,T_s=20) [17]	19.01	0.45%	1.74s	39.03	3.08%	18.67s
PACE (E_s=15,R=1)	18.81	1.73%	0.05s	39.05	3.03%	0.29s
PACE (E_s=15,R=3)	18.82	1.66%	0.16s	39.07	2.96%	2.04s
Minimum Vertex Cover (MVC)	**RB-SMALL**			**RB-LARGE**		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
Gurobi* [38]	205.76	0.00%	3.34s	968.23	0.00%	290.23s
Meta-EGN* [40]	208.97	1.56%	0.30s	1010.69	4.40%	1.03s
COExpander (S=1,D_s=20,T_s=1) [17]	206.75	0.48%	0.14s	969.92	0.18%	1.04s
COExpander (S=4,D_s=20,T_s=1) [17]	206.51	0.37%	0.15s	969.81	0.16%	1.35s
PACE (E_s=80,R=1)	205.97	0.10%	0.05s	969.62	0.15%	0.18s
PACE (E_s=80,R=5)	205.91	0.07%	0.34s	968.88	0.07%	2.77s
Maximum Cut (MCut)	**BA-SMALL**			**BA-LARGE**		
	Obj.↑	Drop↓	Time↓	Obj.↑	Drop↓	Time↓
Gurobi* [38]	727.84	0.00%	60.61s	2936.89	0.00%	300.21s
DiffUCO: CE (S=4,F=1)* [41]	727.53	0.06%	0.61s	2989.46	-1.77%	2.70s
COExpander (S=4,D_s=1,T_s=20) [17]	728.32	-0.05%	0.25s	2960.66	-0.80%	0.67s
PACE (E_s=250,R=1)	727.35	0.08%	0.15s	2975.36	-1.30%	0.95s
PACE (E_s=250,R=5)	728.34	-0.05%	0.81s	2978.34	-1.40%	6.46s
PACE (E_s=500,R=5)	728.43	-0.07%	0.93s	–	–	–

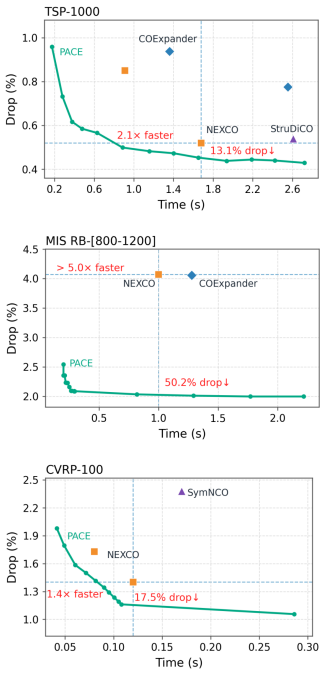


Figure 3: Quality–time trade-offs on TSP-1000, MIS RB-[800-1200], CVRP-100. Green curves trace PACE as budget increases; markers show baselines.

5.3 Test-Time Scaling

Fig. 3 visualizes the quality-time frontier induced by different PACE budgets. On TSP-1000 and MIS RB-[800-1200], larger budgets move PACE beyond the strongest neural baselines shown in the plot, while CVRP-100 exhibits the same budget-responsive trend. Together with Fig. 2, this shows that extra compute extends the same task-native partial state through edits and elite reopening rather than invoking a separate local-search routine. Feasibility audits in Table 11 show 100% feasible closures and active rejection of illegal raw proposals by Γ_I, while runtime breakdowns, architecture ablations, and visual trajectories are given in Tables 17 to 19 and Figs. 4 to 7.

5.4 Ablation Study

Table 6 tests whether the gains of PACE come from partial-state semantics itself. The state-blind scorer keeps the same Γ_I, closure, and growth schedule, but removes the current feasible partial state from the neural input. This roughly doubles TSP-500 Drop from 0.406% to 0.849% and increases TSP-1000 Drop from 0.429% to 0.763%. Collapsing to one-shot global prediction is worse, reaching 1.020% and 0.959% on the same TSP sizes. The effect is much larger on MIS RB-LARGE: full PACE obtains 2.000% Drop, whereas hiding state information gives 14.923% and one-shot prediction gives 15.293%. Thus the scorer must know the current task-native partial state; a plain heatmap followed by closure cannot explain the results. This supports the central modeling choice of PACE:

Table 6: Partial-state semantics ablation. Full and state-blind use the base budgets (E_s=50,R=1 for TSP and E_s=500,R=5 for MIS), while one-shot uses E_s=1,R=1.

Variant	**TSP-500**			**TSP-1000**			**MIS RB-LARGE**		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↑	Drop↓	Time↓
Full PACE	16.613	0.406%	0.604s	23.217	0.429%	2.664s	42.126	2.000%	2.190s
State-blind scorer	16.686	0.849%	0.562s	23.294	0.763%	2.588s	36.542	14.923%	1.879s
one-shot global prediction	16.715	1.020%	0.060s	23.340	0.959%	0.123s	36.380	15.293%	0.090s

learning amortized constraint editing on feasible partial states, then relying on a constraint-preserving transition and closure to maintain task-terminal feasibility throughout budgeted refinement.

6 Conclusion

PACE identifies feasible partial states as the shared semantics for neural combinatorial optimization: a scorer ranks edits, Γ_I commits only legal updates, and Comp_I closes any extendable state into a task-terminal feasible output. Across TSP, ATSP, CVRP, MIS, MVC, MCL, and MCut, these semantics yield competitive quality-time behavior, including a 14.19% average Drop reduction and 2.11$\times$ speedup in the summary comparison. A natural limitation is that PACE still relies on task-specific state, edit, and closure rules; extending these semantics to broader constraint families and reducing such task-specific design remain promising directions for future work.

Acknowledgments and Disclosure of Funding

Funding in direct support of this work: This work was partly supported by the Young Talent Fund of Xi'an Association for Science and Technology (Grant 0959202513148), the Zhejiang Provincial Excellent Funding for Postdoctoral Researchers (Grant ZJ2025156), and the China Postdoctoral Science Foundation (Grant 2025M784428). Bohao Li was supported by the Doctoral Student Academic Scholarship of Northwestern Polytechnical University.

Competing interests: The authors declare no competing interests.

References

[1] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[2] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

[3] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in neural information processing systems*, 34:26198–26211, 2021.

[4] Jakob Weissteiner and Sven Seuken. Deep learning—powered iterative combinatorial auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2284–2293, 2020.

[5] Jakob Weissteiner, Jakob Heiss, Julien Siems, and Sven Seuken. Bayesian optimization-based combinatorial assignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5858–5866, 2023.

[6] Joshua Holder, Natasha Jaques, and Mehran Mesbahi. Multi agent reinforcement learning for sequential satellite assignment problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26516–26524, 2025.

[7] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

[8] Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2022.

[9] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019.

[10] Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4731–4738, 2019.

[11] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

[12] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

[13] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[14] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2024.

[15] Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2025.

[16] Lvda Chen, Yang Li, and Junchi Yan. Maskco: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[17] Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Coexpander: Adaptive solution expansion for combinatorial optimization. In *International Conference on Machine Learning*, 2025.

[18] Yu Wang, Yang Li, Jiale Ma, Junchi Yan, and Yi Chang. Native adaptive solution expansion for diffusion-based combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[19] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

[20] André Hottung, Mridul Mahajan, and Kevin Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[21] Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 37:6081–6125, 2024.

[22] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.

[23] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*, 2019.

[24] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

[25] Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. Goal: A generalist combinatorial optimization agent learner. In *The Thirteenth International Conference on Learning Representations*, 2025.

[26] Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Ml4co-bench-101: Benchmark machine learning for classic combinatorial problems on graphs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[27] Shengyu Feng, Weiwei Sun, Shanda Li, Ameet Talwalkar, and Yiming Yang. Frontierco: Real-world and large-scale evaluation of machine learning solvers for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[28] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006.

[29] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

[30] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[31] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, pages 25531–25546, 2022.

[32] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.

[33] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, et al. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5278–5289, 2025.

[34] Yu Wang, Yang Li, Junchi Yan, and Yi Chang. StruDiCO: Structured denoising diffusion with gradient-free inference-stage boosting for memory and time efficient combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2025.

[35] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34:5138–5149, 2021.

[36] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[37] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. Finding near-optimal independent sets at scale. In *2016 Proceedings of the eighteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 138–150. SIAM, 2016.

[38] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2023. URL `https://www.gurobi.com`.

[39] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems*, 36:11952–11969, 2023.

[40] Haoyu Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta-learning. *arXiv preprint arXiv:2301.03116*, 2023.

[41] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *International Conference on Machine Learning*, pages 43346–43367. PMLR, 2024.

A Task-wise Instantiations of PACE

A.1 Overview of Feasible Partial-state Semantics

PACE uses the same feasible partial-state semantics across all seven tasks. Operationally, for an instance I, each task specifies an ambient partial-state space $\mathcal{X}(I)$ and operates on feasible partial states in $\mathcal{X}_{\text{ext}}(I)$. A state records committed decisions that must be preserved by any task-terminal output $y \in \mathcal{Y}(I)$ under the consistency relation $x \preceq_I y$. The edit scorer $C_\theta(x, I)$ ranks task-native candidate edits from the current state, typically through the implemented candidate set $\mathcal{M}_I(x)$. The projected editing operator Γ_I then scans candidates in score order and commits only edits that pass the task-native feasibility predicate at the current scan state. Finally, the closure operator Comp_I maps an extendable partial state to a task-terminal output while preserving all committed edits.

Thus the neural component and the solver loop are shared: score edits, project the score order through a constraint-preserving transition, and close the resulting state whenever a terminal output is required. Task differences enter only through the state representation, edit type, feasibility predicate, and closure rule. Table 7 reports these choices together with the task-terminal output and normalized objective. The next subsections describe the operational choices behind the table.

Table 7: PACE task instantiations across the seven combinatorial optimization tasks.

Problem	**State**	**Edit**	**Γ_I rule**	Comp_I	**Output**	**Objective**
TSP	path skeleton	edge	degree and subtour check	close path	Hamiltonian cycle	tour length
ATSP	directed path skeleton	arc	in and out degree check	close directed path	directed Hamiltonian cycle	tour cost
CVRP	customer chains	customer edge	degree, cycle, capacity check	attach depot	feasible routes	route length
MIS	independent set	vertex	independence check	greedy saturation	maximal independent set	$-\lvert S\rvert$
MVC	selected set	vertex	monotone insertion	cover uncovered edges	vertex cover	$\lvert S\rvert$
MCL	clique	vertex	clique check	greedy expansion	maximal clique	$-\lvert S\rvert$
MCut	partial assignment	(v, b)	unassigned check	fill remaining nodes	complete cut	$-\mathrm{cut}(y)$

A.2 Edge-oriented Routing Tasks: TSP, ATSP, and CVRP

For edge-oriented routing tasks, PACE edits a partial route skeleton rather than a complete tour or route set. The scorer produces pairwise edge or arc scores, but these scores are only priorities. Whether a candidate can be committed is decided by the evolving feasible partial state inside Γ_I.

For TSP, a state is an undirected path skeleton over all nodes. The committed ON edges determine the current degree of each node and the connected components of the skeleton. A task-native edit is an undirected edge $\{i, j\}$. During editing, the intended invariant is a path cover: each component is a simple path or an isolated node, and no cycle is formed before the final Hamiltonian cycle is closed. The state therefore carries exactly the information needed to decide whether a high-scoring edge can be accepted without violating degree or subtour constraints.

For ATSP, the state stores predecessor and successor commitments. A task-native edit is a directed arc (i, j), with i acting as the tail and j as the head. The intended invariant is a collection of directed paths until the final directed cycle is closed. The state tracks whether each node already has a committed successor and predecessor, and whether an arc would concatenate two directed paths or create a premature directed cycle.

For CVRP, a state is a set of customer chains. A task-native edit is a customer-customer edge; depot edges are not scored by the edit scorer. The state tracks customer endpoint degrees, chain components, and the load of each chain. An accepted edge joins two chain endpoints from distinct components, keeps the customer-only graph acyclic, and merges their loads only when the merged chain remains capacity feasible. Depot edges are left mostly to closure, so projected editing focuses on building capacity-feasible customer chains while postponing route attachment decisions.

A.3 Node-oriented Graph Tasks: MIS, MVC, MCL, and MCut

For node-oriented graph tasks, PACE edits a selected set or a partial assignment. MIS, MVC, and MCL use node edits, while MCut uses label edits.

For MIS, a state is a growing independent selected set S_x. A task-native edit adds a compatible unselected vertex $v \notin S_x$ to S_x. Compatibility is local to the current selected set: v must have no

Table 8: Task-native acceptance checks used by projected editing.

Task	Acceptance check in Φ_I
TSP	Accept an edge if the updated endpoint degrees remain at most two and no subtour is created, except when the edge closes the final spanning path.
ATSP	Accept an arc if the tail has no committed successor, the head has no committed predecessor, and no premature directed cycle is formed.
CVRP	Accept a customer edge if it joins endpoints of two distinct customer chains, keeps customer degrees at most two, preserves customer-only acyclicity, and the merged load respects vehicle capacity.
MIS	Accept an unselected vertex if it has no edge to the current selected set.
MVC	Accept an unselected vertex by monotone insertion into the selected set.
MCL	Accept an unselected vertex if it is adjacent to every vertex in the current selected set.
MCut	Accept a label edit if the vertex is still unassigned and the proposed label is binary.

edge to any committed selected vertex. Closure later saturates the independent set to the task-terminal maximal form.

For MVC, a state is a monotone selected set S_x. A task-native edit adds an unselected vertex to S_x. The partial state need not cover all edges during editing. Its extendability follows from monotonicity of vertex cover completion: adding vertices cannot invalidate the possibility of covering remaining uncovered edges. Closure is responsible for completing the cover.

For MCL, a state is a clique selected set S_x. A task-native edit adds an unselected vertex $v \notin S_x$ adjacent to every vertex already in S_x. The partial state remains clique-consistent after each accepted edit, and closure greedily expands it to a maximal clique.

For MCut, a state is a partial binary assignment. Some vertices have committed side labels and the remaining vertices are open. A task-native edit assigns an open vertex v to one of the two sides, $b \in \{0, 1\}$. Feasibility is consistency of the partial assignment: committed labels are fixed, and no vertex is assigned twice.

A.4 Feasibility Predicates for Projected Editing

Projected editing uses the task-native feasibility predicate Φ_I inside Γ_I. Candidates are scanned in descending score order. After an edit is accepted, it is committed immediately, and all later feasibility checks are evaluated on the updated scan state. This makes acceptance state dependent: a candidate that is valid early in the scan may become invalid after another edit changes degrees, components, selected sets, or assigned labels.

These checks are the implementation rules used by the transition. They define how scored candidates become committed edits during projected editing. The corresponding soundness statements are proved in Sec. B.2.

A.5 Closure Operators for Terminal Completion

The closure operator Comp_I is the task-specific completion routine used when PACE needs a task-terminal output. Closure is a feasibility completion mechanism. It preserves committed edits, returns an element of $\mathcal{Y}(I)$, and does not claim optimality for $\tilde{f}_I$.

For TSP, closure connects remaining path endpoints across the current path cover and then closes the final spanning path into a Hamiltonian cycle. If the state is already a Hamiltonian cycle, closure returns it unchanged.

For ATSP, closure connects the tail of one directed path to the head of another directed path until a single spanning directed path remains, and then closes the final directed cycle. If the state is already a Hamiltonian directed cycle, closure returns it unchanged.

For CVRP, closure orients each customer chain and attaches both ends to the depot. Each chain becomes a depot-rooted route, and the chain load tracked during editing gives the route capacity check.

For MIS, closure greedily adds compatible vertices until no remaining vertex can be inserted without violating independence. The result is a maximal independent set.

For MVC, closure repeatedly selects endpoints of uncovered edges according to the fixed completion rule until every edge is covered. The result is a vertex cover containing the committed selected set.

For MCL, closure greedily adds vertices adjacent to every currently selected vertex until no compatible vertex remains. The result is a maximal clique.

For MCut, closure assigns every remaining open vertex using the fixed completion rule. The result is a complete binary cut assignment that agrees with all committed labels.

B Formal Guarantees for the PACE Framework

This appendix proves the structural claims used in Sec. 4. The objects $\mathcal{Y}(I)$, Ω_I, $\tilde{f}_I$, $\mathcal{X}(I)$, $\mathcal{X}_{\mathrm{ext}}(I)$, $x \preceq_I y$, C_θ, Γ_I, and Comp_I are the ones introduced in Secs. 3 and 4. We use only local notation for task-specific committed structure. If c is an edit admissible at a state x, then $x \oplus c$ denotes the state obtained by committing that edit. The task-native feasibility predicate used by Γ_I is denoted by $\Phi_I(x, c) \in \{0, 1\}$.

B.1 Formal Objects and Benchmark Assumptions

Assumption 1 (Routing benchmark structure)**.** *The routing statements below are made under the benchmark structure used in this paper.*

1. *TSP instances are complete undirected routing instances. Hence every unordered pair of distinct vertices is an available routing edge.*
2. *ATSP instances are given by full directed cost matrices. Hence every ordered pair of distinct vertices is an available routing arc.*
3. *CVRP instances are depot-complete capacitated routing instances with no fixed hard cap on the number of routes. Hence every customer can be connected to the depot, and route feasibility is governed by the vehicle capacity.*

Proposition 1 (Extendability implies task-native structural admissibility)**.** *Fix an instance I of one of the seven tasks and let $x \in \mathcal{X}_{\mathrm{ext}}(I)$. Then the committed structure in x satisfies the following task-native invariants.*

1. *For TSP under Assumption 1, the committed ON-edge graph is either already a Hamiltonian cycle or a spanning disjoint union of simple paths, with isolated vertices treated as paths of length zero. Equivalently, every vertex has committed degree at most two, and no committed cycle exists unless it already spans all vertices.*
2. *For ATSP under Assumption 1, the committed arc graph is either already a Hamiltonian directed cycle or a spanning disjoint union of directed paths and isolated vertices. Equivalently, every vertex has committed out-degree at most one and in-degree at most one, and no committed directed cycle exists unless it already spans all vertices.*
3. *For CVRP under Assumption 1, the customer-only committed graph is a disjoint union of capacity-feasible customer chains. Equivalently, every customer has committed customer-only degree at most two, the customer-only committed graph is acyclic, and every customer component has total demand at most the vehicle capacity.*
4. *For MIS, the committed selected set is an independent set.*
5. *For MVC, the committed selected set is consistent with the monotone selected-set state representation. No additional nontrivial invariant is needed for one-step soundness.*
6. *For MCut, the committed side labels form a consistent partial binary assignment. No additional nontrivial invariant is needed for one-step soundness.*
7. *For MCL, the committed selected set is a clique.*

Proof. By Eq. (3), there exists a task-terminal witness $y \in \mathcal{Y}(I)$ such that $x \preceq_I y$.

For TSP, the witness y is a Hamiltonian cycle. Every committed ON edge in x appears in y. Hence every vertex has committed degree at most two. A proper committed cycle would be a proper cycle contained in the Hamiltonian cycle y, which is impossible because a simple Hamiltonian cycle has no proper cycle subgraph using only its own edges. Therefore any committed cycle must be the full Hamiltonian cycle, and otherwise the committed graph is a spanning disjoint union of paths and isolated vertices.

For ATSP, the witness y is a Hamiltonian directed cycle. Every committed arc in x appears in y. Hence each vertex has committed out-degree at most one and in-degree at most one. A proper committed directed cycle would be a proper directed cycle contained in y, which is impossible. Thus the committed arc graph is either the full Hamiltonian directed cycle or a spanning disjoint union of directed paths and isolated vertices.

For CVRP, the witness y is a depot-rooted capacity-feasible route set. Every committed customer-customer edge in x appears on one route of y. Removing depot edges from each route of y leaves disjoint customer chains. Therefore the committed customer-only graph has maximum degree two and no customer-only cycle. Each connected customer component of x is contained in one feasible route of y, so its demand is at most the vehicle capacity.

For MIS, y is a maximal independent set containing all committed selected vertices, so the committed selected set is independent. For MVC, y is a vertex cover containing the committed selected set; since vertex cover feasibility is monotone under adding vertices, selected-set consistency is the only invariant used below. For MCut, y is a complete binary assignment agreeing with the committed labels, so the current labels are a consistent partial assignment. For MCL, y is a maximal clique containing all committed selected vertices, so the committed selected set is a clique. □

B.2 One-step Soundness of Projected Editing

Lemma 1 (TSP one-step soundness)**.** *Assume the TSP part of Assumption 1. Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be a TSP state and let $c = \{u, v\}$ be an edge with $\Phi_I(x, c) = 1$. Then $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. By Proposition 1, the committed graph of x is either already a Hamiltonian cycle or a spanning path cover. If it is already terminal, the TSP feasibility predicate has no legal edge to accept because every vertex already has degree two and the terminal cycle is closed. Hence an accepted edge is considered only in the path-cover case.

The predicate Φ_I accepts $\{u, v\}$ only if both endpoint degrees remain at most two after insertion and no subtour is created unless the insertion closes the final spanning path. If u and v are in different path components, the degree test implies that they are endpoints of those components. Adding $\{u, v\}$ merges two simple paths into one simple path and creates no cycle. The updated committed graph remains a spanning path cover. If u and v are in the same component, the subtour test permits the insertion only when that component already spans all vertices. Then the component is a spanning path and $\{u, v\}$ closes it into a Hamiltonian cycle.

It remains to show extendability in the nonterminal merge case. Let the updated path components be $P_1, \ldots, P_k$. Choose an orientation of each path and write its endpoints as (a_i, b_i). Completeness gives the edges $\{b_i, a_{i+1}\}$ for $i = 1, \ldots, k-1$ and $\{b_k, a_1\}$. Adding these edges closes the path cover into a Hamiltonian cycle preserving every committed edge. Thus the updated state has a witness in $\mathcal{Y}(I)$. In the final-cycle case, the updated state itself is the witness. Therefore $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$. □

Lemma 2 (ATSP one-step soundness)**.** *Assume the ATSP part of Assumption 1. Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be an ATSP state and let $c = (u, v)$ be an arc with $\Phi_I(x, c) = 1$. Then $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. By Proposition 1, the committed arc graph of x is either already a Hamiltonian directed cycle or a spanning collection of directed paths and isolated vertices. A terminal directed cycle admits no accepted new arc because every vertex already has one successor and one predecessor. Thus we consider the directed-path case.

The predicate accepts (u, v) only if u has no committed successor, v has no committed predecessor, and no directed subtour is created unless the insertion closes the final spanning directed path. If u and v lie in different directed path components, then u is the tail of its component and v is the head of its

component. Adding (u, v) concatenates the two components, preserves the in-degree and out-degree bounds, and creates no directed cycle. If u and v lie in the same component, acceptance means that the component already spans all vertices and that the insertion closes its tail to its head, yielding a Hamiltonian directed cycle.

For a nonterminal updated collection of directed paths $Q_1, \ldots, Q_k$, write h_i and t_i for the head and tail of Q_i. A full directed cost matrix provides arcs (t_i, h_{i+1}) for $i = 1, \ldots, k-1$ and (t_k, h_1). These arcs link the components into a Hamiltonian directed cycle that preserves all committed arcs. In the cycle-closing case the updated state is already terminal. Hence $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$. □

Lemma 3 (CVRP one-step soundness)**.** *Assume the CVRP part of Assumption 1. Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be a CVRP state and let $c = \{u, v\}$ be a customer-customer edge with $\Phi_I(x, c) = 1$. Then $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. By Proposition 1, the committed customer-only graph of x is a disjoint union of capacity-feasible customer chains. The predicate accepts $\{u, v\}$ only when u and v are endpoints of two distinct customer chains, the insertion does not create a customer-only cycle, and the merged component load is at most the vehicle capacity Q_I. Adding the edge therefore merges two chains into one longer customer chain. All other chains are unchanged, and the merged chain is capacity-feasible by the load test.

To construct a terminal witness, orient every updated chain $P_i = (v_1^i, \ldots, v_{m_i}^i)$. Depot completeness gives the route

$$0 \to v_1^i \to \cdots \to v_{m_i}^i \to 0 \tag{11}$$

for each chain. Each route has load at most Q_I. Because the benchmark imposes no hard cap on the number of routes, these routes may remain separate and together cover all customers. The resulting depot-rooted route set belongs to $\mathcal{Y}(I)$ and preserves all committed customer-customer edges, so $x \oplus c \in \mathcal{X}_{\mathrm{ext}}(I)$. □

Lemma 4 (MIS one-step soundness)**.** *Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be an MIS state and let v be a vertex with $\Phi_I(x, v) = 1$. Then $x \oplus v \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. Let S_x be the committed selected set. By Proposition 1, S_x is independent. The MIS predicate accepts v only if $v \notin S_x$ and v is adjacent to no vertex in S_x. Therefore $S_x \cup \{v\}$ is independent. A finite independent set can be greedily saturated by repeatedly adding any vertex that is adjacent to no selected vertex. The result is a maximal independent set in $\mathcal{Y}(I)$ containing $S_x \cup \{v\}$. Hence the updated state is extendable. □

Lemma 5 (MVC one-step soundness)**.** *Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be an MVC state and let v be a vertex with $\Phi_I(x, v) = 1$. Then $x \oplus v \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. Let S_x be the committed selected set. The MVC predicate commits vertices by monotone insertion, so the updated set is $S_x \cup \{v\}$. Vertex-cover extendability is monotone: any selected set can be extended to a vertex cover by adding endpoints of uncovered edges, and in particular by adding all remaining vertices if necessary. Thus there exists a vertex cover in $\mathcal{Y}(I)$ containing the updated selected set, which proves extendability. □

Lemma 6 (MCut one-step soundness)**.** *Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be an MCut state and let (v, b) be an assignment edit with $\Phi_I(x, (v, b)) = 1$. Then $x \oplus (v, b) \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. The MCut predicate accepts (v, b) only when v is currently unassigned and $b \in \{0, 1\}$. Committing the edit therefore preserves consistency of the partial binary assignment. Every remaining unassigned vertex can be assigned arbitrarily to either side. This gives a complete binary cut assignment in $\mathcal{Y}(I)$ agreeing with the updated state, so the updated state is extendable. □

Lemma 7 (MCL one-step soundness)**.** *Let $x \in \mathcal{X}_{\mathrm{ext}}(I)$ be an MCL state and let v be a vertex with $\Phi_I(x, v) = 1$. Then $x \oplus v \in \mathcal{X}_{\mathrm{ext}}(I)$.*

Proof. Let S_x be the committed selected set. By Proposition 1, S_x is a clique. The MCL predicate accepts v only if $v \notin S_x$ and v is adjacent to every vertex in S_x. Hence $S_x \cup \{v\}$ is a clique. A finite clique can be greedily expanded by repeatedly adding any vertex adjacent to all currently selected vertices. The result is a maximal clique in $\mathcal{Y}(I)$ containing $S_x \cup \{v\}$, so the updated state is extendable. □

B.3 Constructive Completion for Task-terminal Outputs

Proposition 2 (TSP completion)**.** *Assume the TSP part of Assumption 1. For every TSP state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$. If x is already task-terminal, then $\text{Comp}_I(x) = x$.*

Proof. If x is already a Hamiltonian cycle, closure returns x. Otherwise, Proposition 1 gives a spanning path cover. While more than one path component remains, Comp_I connects endpoints of two distinct path components according to its fixed tie-breaking rule. Completeness guarantees that the chosen edge exists, and the operation merges two paths into one path without creating a cycle. The number of components decreases by one, so after finitely many steps a single spanning path remains. Completeness then gives the edge between its two endpoints, which closes a Hamiltonian cycle. All added edges are additional to the committed edges, so the output preserves x and belongs to $\mathcal{Y}(I)$. □

Proposition 3 (ATSP completion)**.** *Assume the ATSP part of Assumption 1. For every ATSP state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$. If x is already task-terminal, then $\text{Comp}_I(x) = x$.*

Proof. If x is already a Hamiltonian directed cycle, closure returns x. Otherwise, Proposition 1 gives a spanning collection of directed paths and isolated vertices. While more than one component remains, Comp_I connects the tail of one component to the head of another according to its fixed tie-breaking rule. The full cost matrix guarantees that the required arc exists. This operation concatenates two directed paths and reduces the number of components by one. The process terminates with a single spanning directed path, and the full cost matrix gives the arc from its tail to its head. Adding that arc yields a Hamiltonian directed cycle preserving all committed arcs. Hence the closure output lies in $\mathcal{Y}(I)$ and is consistent with x. □

Proposition 4 (CVRP completion)**.** *Assume the CVRP part of Assumption 1. For every CVRP state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$.*

Proof. By Proposition 1, the committed customer-only graph is a disjoint union of capacity-feasible customer chains. Closure orients each chain using its fixed tie-breaking rule and attaches both ends to the depot. Depot completeness gives both depot edges. The load of each route is the demand of one chain and is at most the vehicle capacity. Since there is no fixed hard cap on the number of routes, each chain may form its own route. The resulting route set covers all customers, satisfies capacity, preserves all committed customer-customer edges, and therefore lies in $\mathcal{Y}(I)$. □

Proposition 5 (MIS completion)**.** *For every MIS state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$.*

Proof. Let S_x be the committed selected set. By Proposition 1, S_x is independent. Closure scans vertices in its fixed order and adds a vertex exactly when it is adjacent to no currently selected vertex. Independence is preserved after every addition. Because the graph is finite, the scan-and-add procedure terminates. At termination, no unselected vertex can be added without violating independence, so the result is a maximal independent set. It contains S_x, belongs to $\mathcal{Y}(I)$, and preserves the state. □

Proposition 6 (MVC completion)**.** *For every MVC state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$.*

Proof. Let S_x be the committed selected set. Closure repeatedly finds an uncovered edge and adds one of its endpoints according to its fixed rule. Each step strictly increases the selected set unless all edges are already covered. Since the vertex set is finite, the procedure terminates. When it terminates,

every edge has at least one selected endpoint, so the resulting set is a vertex cover. It contains S_x, hence it is in $\mathcal{Y}(I)$ and is consistent with x. □

Proposition 7 (MCut completion)**.** *For every MCut state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$.*

Proof. By Proposition 1, the committed side labels form a consistent partial binary assignment. Closure assigns every unassigned vertex a label in $\{0, 1\}$ using the task's fixed completion rule. The resulting assignment is complete and agrees with all committed labels. Thus it is an element of $\mathcal{Y}(I)$ and preserves x. □

Proposition 8 (MCL completion)**.** *For every MCL state $x \in \mathcal{X}_{\text{ext}}(I)$, $\text{Comp}_I(x)$ is well-defined, belongs to $\mathcal{Y}(I)$, and satisfies $x \preceq_I \text{Comp}_I(x)$.*

Proof. Let S_x be the committed selected set. By Proposition 1, S_x is a clique. Closure scans vertices in its fixed order and adds a vertex exactly when it is adjacent to every currently selected vertex. The clique property is preserved after every addition. Finiteness gives termination. At termination, no unselected vertex is adjacent to all selected vertices, so the result is a maximal clique. It contains S_x, belongs to $\mathcal{Y}(I)$, and preserves x. □

B.4 Anytime Feasibility under Budgeted Editing

The main text uses Γ_I in Eq. (4). We now state the scan recursion used by that operator. Let the candidate list in round t be $C_t = (c_{t,1}, \ldots, c_{t,L_t})$, sorted by the scores in $C_\theta(x_t, I)$. Define

$$z_{t,0} = x_t, \qquad a_{t,0} = 0, \tag{12}$$

where $a_{t,j}$ counts accepted edits in round t. For $j = 1, \ldots, L_t$, set

$$(a_{t,j}, z_{t,j}) = \begin{cases} \big(a_{t,j-1} + 1,\ z_{t,j-1} \oplus c_{t,j}\big), & \text{if } a_{t,j-1} < m_t \text{ and } \Phi_I(z_{t,j-1}, c_{t,j}) = 1, \\ \big(a_{t,j-1},\ z_{t,j-1}\big), & \text{otherwise.} \end{cases} \tag{13}$$

Finally $x_{t+1} = z_{t,L_t}$. This is the local specialization of Γ_I used below.

Theorem 1 (State-space closure under projected editing)**.** *Fix one of the seven tasks. Assume Assumption 1 for the routing task when applicable. Let $x_0 \in \mathcal{X}_{\text{ext}}(I)$ and let $z_{t,j}$ be defined by Eqs.* (12) *and* (13) *for each editing round $t = 0, \ldots, T-1$. Then every scan state remains in the extendable subset:*

$$z_{t,j} \in \mathcal{X}_{\text{ext}}(I), \qquad \forall t \in \{0, \ldots, T-1\},\ \forall j \in \{0, \ldots, L_t\}. \tag{14}$$

Consequently $x_{t+1} = z_{t,L_t} \in \mathcal{X}_{\text{ext}}(I)$ for every round.

Proof. We use induction over the outer edit round and, inside each round, induction over the scan index. The outer base case is $x_0 \in \mathcal{X}_{\text{ext}}(I)$ by assumption.

Assume $x_t \in \mathcal{X}_{\text{ext}}(I)$. The inner base case is $z_{t,0} = x_t$, so $z_{t,0}$ is extendable. Suppose $z_{t,j-1} \in \mathcal{X}_{\text{ext}}(I)$. If the jth candidate is rejected because the budget is exhausted or $\Phi_I(z_{t,j-1}, c_{t,j}) = 0$, then Eq. (13) gives $z_{t,j} = z_{t,j-1}$, so extendability is unchanged. If it is accepted, then $\Phi_I(z_{t,j-1}, c_{t,j}) = 1$. By Proposition 1, the current scan state satisfies the task-native invariants required by the corresponding one-step lemma among Lemmas 1 to 7. That lemma yields

$$z_{t,j} = z_{t,j-1} \oplus c_{t,j} \in \mathcal{X}_{\text{ext}}(I). \tag{15}$$

Thus all scan states in round t are extendable, and in particular $x_{t+1} = z_{t,L_t}$ is extendable. This closes the outer induction and proves the claim for all rounds and scan indices. □

Corollary 1 (Anytime terminal completion)**.** *Under the assumptions of Theorem 1, after any number of editing rounds T, the closed output*

$$y_T = \text{Comp}_I(x_T) \tag{16}$$

satisfies $y_T \in \mathcal{Y}(I)$ and $x_T \preceq_I y_T$.

Proof. By Theorem 1, $x_T \in \mathcal{X}_{\text{ext}}(I)$. The corresponding constructive completion proposition among Propositions 2 to 8 applies to x_T and gives $\text{Comp}_I(x_T) \in \mathcal{Y}(I)$ with $x_T \preceq_I \text{Comp}_I(x_T)$. □

B.5 Alignment of the Edit-ranking Loss

Proposition 9 (Implemented ranking-term alignment)**.** *Fix a training partial state x, a reference $y^{\mathrm{ref}} \in \mathcal{Y}(I)$, and a nonempty positive set $\mathcal{F}_I^+(x; y^{\mathrm{ref}})$ as in Eqs.* (7) *and* (8)*. Let*

$$\begin{aligned} F^+ &= \mathcal{F}_I^+(x; y^{\mathrm{ref}}), \\ M &= \mathcal{M}_I(x), \\ p(c) &= p_\theta(c \mid x, I). \end{aligned} \tag{17}$$

For every $c \in M$,

$$\frac{\partial \mathcal{L}_{\mathrm{edit}}(x, y^{\mathrm{ref}})}{\partial s_\theta(c; x, I)} = p(c) - \frac{\mathbf{1}\{c \in F^+\}}{|F^+|}. \tag{18}$$

Consequently, every implemented negative $c^- \in M \setminus F^+$ has positive logit gradient

$$\frac{\partial \mathcal{L}_{\mathrm{edit}}}{\partial s_\theta(c^-; x, I)} = p(c^-) > 0, \tag{19}$$

and the aggregate positive-set gradient is

$$\sum_{c^+ \in F^+} \frac{\partial \mathcal{L}_{\mathrm{edit}}}{\partial s_\theta(c^+; x, I)} = \sum_{c^+ \in F^+} p(c^+) - 1 = -p(M \setminus F^+) \leq 0. \tag{20}$$

The inequality is strict whenever candidate negatives exist. If $M = F^+$, then

$$\frac{\partial \mathcal{L}_{\mathrm{edit}}}{\partial s_\theta(c; x, I)} = p(c) - \frac{1}{|F^+|}, \qquad c \in F^+, \tag{21}$$

so the ranking term only shapes the distribution inside the positive set and is stationary on this normalization set exactly when p is uniform on F^+.

Proof. For the softmax in Eq. (7),

$$\frac{\partial \log p(c')}{\partial s_\theta(c; x, I)} = \mathbf{1}\{c = c'\} - p(c). \tag{22}$$

Substituting this identity into Eq. (8) gives Eq. (18). If $c^- \notin F^+$, the indicator term is zero, giving Eq. (19). Under gradient descent, a positive gradient on a negative logit decreases that logit.

Summing Eq. (18) over F^+ gives Eq. (20). Equivalently, if a scalar offset α is added to all positive logits and no negative logit is changed, then

$$\begin{aligned} \frac{\partial \mathcal{L}_{\mathrm{edit}}}{\partial \alpha} &= \sum_{c^+ \in F^+} \frac{\partial \mathcal{L}_{\mathrm{edit}}}{\partial s_\theta(c^+; x, I)} \\ &= -p(M \setminus F^+). \end{aligned} \tag{23}$$

Thus, when negatives exist, increasing the positive logits as a group decreases the loss and shifts softmax mass toward reference-consistent edits. When $M = F^+$, there is no positive-versus-negative separation. In that case Eq. (18) reduces to Eq. (21), and all gradients vanish exactly at the uniform distribution on F^+. □

Remark 1 (Calibration terms). The proposition concerns the ranking term in Eqs. (7) and (8). BCE, two-class CE, column-wise consistency terms for ATSP, and related calibration losses used in task-specific implementations are auxiliary implementation terms. They can be useful for calibration or dense supervision, but they are not the core formal claim above. If $\mathcal{F}_I^+(x; y^{\mathrm{ref}}) = \varnothing$, Proposition 9 is not invoked for that state.

B.6 Elite-pool Monotonicity and Scope of Claims

Theorem 2 (Outer-pool best-so-far monotonicity)**.** *Let $K \geq 1$. Suppose the outer pool at round r is a nonempty set $\mathcal{P}_r \subseteq \mathcal{Y}(I)$ and the next update is elitist:*

$$\mathcal{P}_{r+1} = \mathrm{TopK}\big(\mathcal{P}_r \cup \widetilde{\mathcal{P}}_{r+1}, K\big), \tag{24}$$

where $\widetilde{\mathcal{P}}_{r+1} \subseteq \mathcal{Y}(I)$ is the set of newly closed outputs and TopK *keeps the K elements with smallest normalized minimized objective $\tilde{f}_I$, preserving at least one minimizer of its input. Define*

$$b_r = \min_{y \in \mathcal{P}_r} \tilde{f}_I(y). \tag{25}$$

Then $b_{r+1} \leq b_r$.

Proof. Since $\mathcal{P}_r \subseteq \mathcal{P}_r \cup \widetilde{\mathcal{P}}_{r+1}$,

$$\begin{aligned} \min_{y \in \mathcal{P}_r \cup \widetilde{\mathcal{P}}_{r+1}} \tilde{f}_I(y) &\leq \min_{y \in \mathcal{P}_r} \tilde{f}_I(y) \\ &= b_r. \end{aligned} \tag{26}$$

The elitist TopK update preserves a minimizer of the union, so the minimum value inside $\mathcal{P}_{r+1}$ equals the minimum value of the union. Therefore $b_{r+1} \leq b_r$. □

Remark 2 (Scope of the monotonicity claim). Theorem 2 is only a supporting property of the outer elite pool under the normalized minimized objective $\tilde{f}_I$. It does not imply global optimality, convergence to an optimum, an approximation ratio, or monotonic objective improvement over the inner edit steps of Γ_I.

C Experimental Setup and Implementation Details

C.1 Benchmarks, Metrics, and Reference Solvers

The evaluation covers seven tasks: TSP, ATSP, CVRP, MIS, MCL, MVC, and MCut. These tasks instantiate the same PACE semantic contract on two problem families. The routing benchmarks include uniform Euclidean TSP and CVRP instances, together with full directed cost-matrix ATSP instances. The graph benchmarks include RB, ER, SATLIB, and BA instances following ML4CO-Bench-101 [26]. Dataset families, reference objectives, and held-out instances are summarized in Table 9.

Table 9: Dataset statistics for the PACE evaluation benchmarks.

Problem	Dataset	Reference Solver	Testing Data Size	Obj.
TSP	TSP-50	Concorde	1,280	5.688
	TSP-100	Concorde	1,280	7.756
	TSP-500	Concorde	128	16.546
	TSP-1000	Concorde	128	23.118
ATSP	ATSP-50	LKH(1000)	2,500	1.5545
	ATSP-100	LKH(1000)	2,500	1.5660
	ATSP-200	LKH(1000)	100	1.5647
	ATSP-500	LKH(1000)	100	1.5734
CVRP	CVRP-50	HGS	10,000	10.366
	CVRP-100	HGS	10,000	15.563
	CVRP-200	HGS	100	19.630
	CVRP-500	HGS	100	37.154
MIS	RB-[200-300]	KaMIS(60s)	500	20.090
	RB-[800-1200]	KaMIS(60s)	500	43.004
	ER-[700-800]	KaMIS(60s)	128	44.969
	SATLIB	KaMIS(60s)	500	425.954
MCL	RB-SMALL	Gurobi(60s)	500	19.082
	RB-LARGE	Gurobi(300s)	500	40.182
MVC	RB-SMALL	Gurobi(60s)	500	205.764
	RB-LARGE	Gurobi(300s)	500	968.228
MCut	BA-SMALL	Gurobi(60s)	500	727.844
	BA-LARGE	Gurobi(300s)	500	2936.886

The reference solver column uses Concorde for TSP, LKH for ATSP, HGS for CVRP, KaMIS for MIS, and Gurobi for MCL, MVC, and MCut. Solver limits are shown in the table where applicable.

Obj. denotes the averaged task objective in the direction indicated by the table. Drop reports relative deviation from the reference solver objective, and Time denotes average inference time per instance. In terms of the normalized minimized objective $\tilde{f}_I$, the relative deviation is

$$100 \cdot \frac{\tilde{f}_I(y) - \tilde{f}_I(y^{\mathrm{ref}})}{|\tilde{f}_I(y^{\mathrm{ref}})|}. \tag{27}$$

For minimization tasks, Drop measures excess objective relative to the reference solver. For maximization tasks, Drop measures loss relative to the reference objective through the same normalized minimized objective; negative Drop indicates that the method exceeds the time-limited reference solver on the reported instances.

C.2 Training Data and Model Configurations

Table 10 reports the training data size, number of epochs, batch size, learning rate, warmup, layer counts, attention heads, and hidden dimension for each trained PACE model.

Table 10: Hyperparameters and datasets for training PACE across different combinatorial optimization tasks. Data sizes follow ML4CO-Bench-101 [26].

Problem	**Model**	**Pretrain**	**Training Parameters**					**Model Parameters**		
			Data Size	Epoch	Batch Size	LR	Warmup	Layers	Heads	Hidden Dim.
TSP	Uniform-50	–	1,280,000	800	1024	1e-3	1000	16, 6	8	256
	Uniform-100	–	1,280,000	800	1024	1e-3	1000	16, 6	8	256
	Uniform-500	Uniform-100	128,000	800	128	1e-3	1000	16, 6	8	256
	Uniform-1000	Uniform-500	64,000	600	32	1e-3	1000	16, 6	8	256
ATSP	Uniform-50	–	640,000	600	1024	1e-3	1000	16, 6	8	256
	Uniform-100	Uniform-50	128,000	600	1024	1e-3	1000	16, 6	8	256
	Uniform-200	Uniform-100	32,000	600	256	1e-3	1000	16, 6	8	256
	Uniform-500	Uniform-200	6,400	10	128	2e-5	100	16, 6	8	256
CVRP	Uniform-50	–	1,280,000	600	1024	5e-4	1000	16, 6	8	256
	Uniform-100	Uniform-50	640,000	600	1024	5e-4	1000	16, 6	8	256
	Uniform-200	Uniform-100	128,000	600	512	5e-4	500	16, 6	8	256
	Uniform-500	Uniform-200	12,800	600	128	5e-4	100	16, 6	8	256
MIS	RB-[200-300]	–	64,000	600	256	5e-4	1000	0, 16	4	256
	RB-[800-1200]	RB-[200-300]	6,400	300	32	2e-4	1000	0, 16	4	256
	ER-[700-800]	–	12,800	300	32	2e-4	1000	0, 24	4	256
	SATLIB	–	39,500	600	32	5e-4	1000	0, 16	4	256
MCL	RB-SMALL	–	64,000	600	256	5e-4	1000	0, 16	4	256
	RB-LARGE	RB-SMALL	6,400	300	32	2e-4	1000	0, 16	4	256
MVC	RB-SMALL	–	128,000	600	256	1e-3	1000	0, 16	4	256
	RB-LARGE	RB-SMALL	6,400	300	32	5e-4	1000	0, 16	4	256
MCut	BA-SMALL	–	128,000	600	256	5e-4	1000	0, 16	4	256
	BA-LARGE	–	128,000	600	32	2e-4	1000	0, 16	4	256

PACE uses the same semantic contract across tasks: the edit scorer ranks candidate edit actions on the current feasible partial state, Γ_I filters and commits feasible edits, and Comp_I produces a task-terminal feasible output when invoked. The model instances are nevertheless trained per task and scale. When the table specifies a pretrain source, the corresponding larger-scale routing or graph model is initialized from the smaller-scale checkpoint shown in the same column. This curriculum pattern is used for several large TSP, ATSP, CVRP, MIS, MCL, and MVC settings, while the table also lists settings trained without such initialization.

C.3 Scorer Architecture and State Encodings

PACE uses two scorer families, one for edge-oriented routing and one for node-oriented graph combinatorial optimization. Both families return edit logits only. They do not enforce feasibility and do not perform terminal completion; these roles are handled by the task-native constraint-preserving transition Γ_I and closure Comp_I. The scorer also does not condition explicitly on the refinement step. Repeated improvement is driven by the evolving feasible partial state x, not by a time input.

Edge-oriented routing scorer. For TSP, ATSP, and CVRP, the scorer is a Transformer with an encoder and decoder over raw task descriptors $\phi_I(i)$. For Euclidean routing, the descriptors include node coordinates; for CVRP, they also include customer demand. Inputs are first projected as

$$h_i^0 = W_{\text{in}}\phi_I(i). \tag{28}$$

The encoder builds instance embeddings, and the decoder returns state-aware node states u_i. The current feasible partial state enters attention through an additive bias:

$$\text{Attn}(Q, K, V; x, I) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + B_{\text{route}}(x, I)\right) V. \tag{29}$$

The edge-oriented routing heads then map node states to edit logits. TSP uses a symmetric edge head,

$$s_\theta(\{i, j\}; x, I) = u_i^\top u_j, \tag{30}$$

whereas ATSP uses separate source and target projections to preserve direction:

$$s_\theta((i, j); x, I) = (W_s u_i)^\top (W_t u_j). \tag{31}$$

CVRP reuses the pairwise routing head on customer nodes. Depot connections are not scored as the main edit type and are inserted mainly by Comp_I.

Node-oriented graph scorer. For MIS, MVC, MCL, and MCut, the scorer is a graph Transformer with adjacency bias. The feasible partial state is represented by node state embeddings $e_i(x)$ and combined with structural node features $g_i(I)$:

$$h_i^0 = W_{\text{state}} e_i(x) + W_{\text{feat}} g_i(I). \tag{32}$$

Attention uses the graph adjacency as an additive bias:

$$\text{Attn}(Q, K, V; I) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + A_I\right) V, \tag{33}$$

where A_I denotes the adjacency bias. MIS, MVC, and MCL use one node logit per candidate vertex. MCut uses label logits for assigning an open vertex to one of the two sides.

C.4 Hardware and Runtime Measurement

All experiments, including training and testing, were run with an Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz and an NVIDIA A100 40GB GPU. Time is reported as average inference time per instance. Runtime measurements correspond to the same budget settings reported in the result tables, including the stated values of E_s, R, K, and B when those budget parameters are used.

The reported time includes the neural scorer forward pass, candidate construction, projected editing through Γ_I, closure, objective evaluation, and elite pool updates when applicable. It also includes reopen or refinement operations when they are part of the listed budget. Component-level timing is deferred to Table 17, which separates these costs on representative benchmarks.

D Supplementary Experimental Results

The main experiments report the headline quality and runtime comparisons. This appendix adds checks on feasibility robustness, cross-scale and cross-distribution transfer, real-world TSP behavior, runtime composition, ablations, summary comparisons, and qualitative state-editing trajectories. Together, these results provide supplementary evidence for how PACE state semantics behave under different data regimes, budgets, and implementation choices.

D.1 Feasibility Robustness of Projection and Closure

Table 11 audits three quantities on representative constrained tasks: raw top candidate proposals before task-specific feasibility masking, rejection by Γ_I at the current feasible partial state, and feasibility of the closed task-terminal feasible outputs.

All checked closures are feasible across ATSP, CVRP, and MCut in this audit. The nonzero rejection rates show that Γ_I is active in the solve loop rather than merely decorative: scored proposals are filtered by the current feasible partial state before they can mutate the trajectory. The MCut rejection rate is especially high because repeated rounds leave many vertices already assigned while raw label proposals can still revisit them. These results support the PACE semantic contract that neural scores propose candidate edits, while task-native projection and closure maintain feasible terminal outputs.

Table 11: Feasibility robustness audit on representative constrained tasks. Raw top-K proposals are audited before task-specific feasibility masking; a proposal is counted as rejected when Γ_I would reject it in the current partial state.

Problem	Dataset	Setting	Instances	Checked closures	Closure feasible	Raw proposals	Rejected by Γ_I
ATSP	ATSP-100	E_s=50,R=1	2,500	125,000	125,000 (100.0%)	60,912,428	30,364,185 (49.8%)
CVRP	CVRP-100	E_s=50,R=1	10,000	10,000	10,000 (100.0%)	115,350,958	37,983,532 (32.9%)
MCut	BA-SMALL	E_s=250,R=5	500	2,500	2,500 (100.0%)	25,249,047	22,012,502 (87.2%)

D.2 Cross-scale and Cross-distribution Generalization

Table 12: Cross-scale generalization results on TSP. * denotes reported results from prior work.

Testing \ Training		TSP-50	TSP-100	TSP-500	TSP-1K
TSP-50	DIFUSCO (T_s=50)* [12]	0.09%	0.25%	2.55%	2.71%
	T2T (T_s=50,T_g=30)* [13]	0.02%	0.11%	1.60%	**1.10%**
	Fast T2T (T_s=5,T_g=5)* [14]	0.09%	0.36%	1.02%	1.26%
	PACE (E_s=50,R=5)	**0.001%**	**0.004%**	**0.775%**	1.653%
TSP-100	DIFUSCO (T_s=50)* [12]	1.44%	0.23%	3.44%	3.31%
	T2T (T_s=50,T_g=30)* [13]	0.56%	0.17%	2.47%	2.19%
	Fast T2T (T_s=5,T_g=5)* [14]	0.12%	0.02%	**0.40%**	**0.55%**
	PACE (E_s=50,R=5)	**0.083%**	**0.003%**	1.041%	1.621%
TSP-500	DIFUSCO (T_s=50)* [12]	4.16%	3.04%	1.40%	1.85%
	T2T (T_s=50,T_g=30)* [13]	3.79%	2.25%	0.91%	1.22%
	Fast T2T (T_s=5,T_g=5)* [14]	2.67%	1.77%	0.38%	0.95%
	PACE (E_s=50,R=5)	**2.501%**	**1.672%**	**0.254%**	**0.233%**
TSP-1K	DIFUSCO (T_s=50)* [12]	4.54%	3.98%	2.65%	2.21%
	T2T (T_s=50,T_g=30)* [13]	4.66%	3.81%	1.61%	1.30%
	Fast T2T (T_s=5,T_g=5)* [14]	3.46%	**3.08%**	1.06%	0.58%
	PACE (E_s=50,R=5)	**2.557%**	3.893%	**0.529%**	**0.249%**

TSP. Table 12 evaluates cross-scale transfer across TSP-50, TSP-100, TSP-500, and TSP-1000. The PACE rows are strongest on matched-scale and large-scale training. Small-scale PACE checkpoints transfer well to TSP-50 and TSP-100, while larger TSP-500 and TSP-1000 checkpoints improve transfer to larger test sizes. Training at TSP-500 gives much lower Drop on TSP-1000 than training at TSP-50 or TSP-100, which is consistent with a state-conditioned scorer benefiting from partial-state contexts seen at similar or larger scales. The remaining off-scale cells give a more detailed picture of transfer behavior across direction and scale.

MIS. Table 13 treats RB-to-ER and RB-to-SATLIB transfer as distribution-shift stress tests. The RB-[800-1200] model gives a 9.01% Drop on ER-[700–800] and a 5.94% Drop on SATLIB, while the RB-[200–300] model reports a 6.56% Drop on SATLIB. These values are higher than the matched-distribution MIS results in the main tables, reflecting the stronger shift in graph family. The selected-set state semantics still transfer across RB, ER, and SATLIB settings while using the same transition and closure semantics.

Table 13: Cross-distribution generalization results on MIS.

Training \ Testing	ER-[700-800]		SATLIB	
	Obj.↑	Drop↓	Obj.↑	Drop↓
RB-[200-300]	–	–	398.13	6.56%
RB-[800-1200]	40.91	9.01%	400.69	5.94%

MCL and MVC. Table 14 reports small-to-large and large-to-small transfer for MCL. For MCL, matched-scale training is clearly strongest. The RB-SMALL model reaches 1.73% Drop on RB-SMALL but degrades to 11.00% Drop on RB-LARGE, while the RB-LARGE

Table 14: Cross-scale generalization results on MCL.

Training \ Testing	RB-SMALL		RB-LARGE	
	Obj.↑	Drop↓	Obj.↑	Drop↓
RB-SMALL	18.81	1.73%	35.73	11.00%
RB-LARGE	18.54	3.15%	39.05	3.03%

model reaches 3.03% Drop on RB-LARGE and 3.15% Drop on RB-SMALL. The visible cost of scale mismatch suggests that clique scoring is sensitive to the scale and density regimes represented during training.

Table 15 gives the corresponding transfer results for MVC. MVC shows the same matched-scale pattern, with 0.10% Drop on RB-SMALL for the RB-SMALL model and 0.15% Drop on RB-LARGE for the RB-LARGE model. Off-scale transfer gives 1.81% Drop from RB-SMALL to RB-LARGE and 1.61% Drop from RB-LARGE to RB-SMALL. Together, the MCL and MVC tables show that the node-oriented state semantics can be applied across scale changes, with matched training providing the strongest operating point against strong reference solvers.

Table 15: Cross-scale generalization results on MVC.

Training \ Testing	RB-SMALL		RB-LARGE	
	Obj.↓	Drop↓	Obj.↓	Drop↓
RB-SMALL	205.97	0.10%	985.72	1.81%
RB-LARGE	209.06	1.61%	969.62	0.15%

D.3 Real-world TSP Evaluation on TSPLIB

Table 16 evaluates a model trained on random 100-node TSP instances on TSPLIB instances with 50 to 200 nodes. The table reports solution-quality Drop values against prior methods on each TSPLIB instance. PACE obtains a mean Drop of 0.07%, compared with 0.13% for StruDiCO in the table and larger mean Drop values for the other listed baselines. Most PACE entries are near zero, and the mean result indicates strong transfer from random Euclidean training data to structured TSPLIB geometry.

Table 16: Solution quality for methods trained on random 100-node TSP instances and evaluated on **TSPLIB instances with 50–200 nodes**. * denotes results quoted from previous works [34].

Instances	AM*	Learn2OPT*	GNNGLS*	DIFUSCO*	T2T*	Fast T2T*	StruDiCO*	PACE
eil51	16.767%	1.725%	1.529%	2.82%	0.14%	0.00%	0.00%	0.00%
berlin52	4.169%	0.449%	0.142%	0.00%	0.00%	0.00%	0.00%	-0.00%
st70	1.737%	0.040%	0.764%	0.00%	0.00%	0.00%	0.00%	-0.00%
eil76	1.992%	0.096%	0.163%	0.34%	0.00%	0.00%	0.00%	-0.00%
pr76	0.816%	1.228%	0.039%	1.12%	0.40%	0.00%	-0.00%	-0.00%
rat99	2.645%	0.123%	0.550%	0.09%	0.09%	0.00%	0.00%	-0.00%
kroA100	4.017%	18.313%	0.728%	0.10%	0.00%	0.00%	0.00%	-0.00%
kroB100	5.142%	1.119%	0.147%	2.29%	0.74%	0.65%	0.00%	0.00%
kroC100	0.972%	0.349%	1.571%	0.00%	0.00%	0.00%	0.00%	-0.00%
kroD100	2.717%	0.866%	0.572%	0.07%	0.00%	0.00%	0.00%	-0.00%
kroE100	1.470%	1.832%	1.216%	3.83%	0.27%	0.13%	2.15%	-0.00%
rd100	3.407%	1.725%	0.003%	0.08%	0.00%	0.00%	0.11%	-0.00%
eil101	2.994%	1.529%	0.03%	0.03%	0.00%	0.00%	0.00%	-0.00%
lin105	1.739%	1.867%	0.606%	0.00%	0.00%	0.00%	0.18%	-0.00%
pr107	3.933%	0.898%	0.439%	0.91%	0.61%	0.62%	0.18%	-0.00%
pr124	2.677%	10.232%	0.755%	1.02%	0.08%	0.08%	-0.00%	-0.00%
bier127	5.908%	3.044%	1.948%	0.94%	0.54%	1.50%	0.04%	0.96%
ch130	3.182%	0.709%	3.519%	0.29%	0.06%	0.00%	0.24%	0.00%
pr136	5.064%	0.000%	3.387%	0.19%	0.10%	0.01%	0.04%	-0.00%
pr144	7.641%	1.526%	3.581%	0.80%	0.50%	0.39%	0.00%	0.03%
ch150	1.584%	0.321%	2.113%	0.57%	0.49%	0.00%	0.04%	-0.00%
kroA150	3.784%	0.724%	2.984%	0.34%	0.14%	0.00%	0.24%	-0.00%
kroB150	2.437%	0.886%	3.258%	0.30%	0.00%	0.07%	0.00%	0.05%
pr152	7.494%	3.119%	3.119%	1.69%	0.83%	0.19%	0.69%	0.19%
u159	7.551%	0.054%	1.020%	0.82%	0.00%	0.00%	0.00%	-0.00%
rat195	6.893%	0.743%	1.666%	1.48%	1.27%	0.79%	-0.00%	0.27%
d198	373.020%	0.522%	4.727%	3.32%	1.97%	0.86%	-0.00%	0.39%
kroA200	7.106%	1.441%	2.029%	2.28%	0.57%	0.49%	0.00%	0.06%
kroB200	8.541%	0.646%	2.589%	2.35%	0.92%	2.50%	-0.00%	0.00%
Mean	**16.767%**	**1.725%**	**1.529%**	**0.97%**	**0.35%**	**0.28%**	**0.13%**	**0.07%**

D.4 Runtime Breakdown

Table 17 decomposes average inference time per instance into scorer forward, candidate construction, Γ_I projection, closure, reopen or refinement, objective or pool update, and other overhead. Across the representative settings, Γ_I projection is a small part of total runtime. The bottleneck varies by task. On TSP-1000, objective or pool update dominates the measured time, while candidate construction is also visible. On MIS RB-[800-1200], CVRP-100, and MCut BA-SMALL, the scorer forward pass is a major component. Closure is most visible for CVRP, where converting customer chains into depot-rooted route sets accounts for a larger share than in the other listed settings. These proportions support the design goal that feasibility control remains lightweight relative to scoring and bookkeeping in the measured cases.

Table 17: Runtime breakdown of PACE inference on representative benchmarks. Each component is shown as seconds with its share of total time in parentheses. Γ_I denotes the constraint-preserving projection.

Component	TSP-1000 E_s=50,R=1	MIS RB-[800-1200] E_s=500,R=1	CVRP-100 E_s=40,R=1	MCut BA-SMALL E_s=250,R=5
Total	2.672s	0.293s	0.108s	0.820s
Scorer forward	0.191s (7.1%)	0.148s (50.7%)	0.056s (52.0%)	0.527s (64.3%)
Candidate construction	0.544s (20.4%)	0.066s (22.4%)	0.009s (8.0%)	0.061s (7.4%)
Γ_I projection	0.006s (0.2%)	0.018s (6.2%)	0.001s (0.7%)	0.009s (1.1%)
Closure	0.008s (0.3%)	0.004s (1.3%)	0.027s (25.0%)	0.009s (1.1%)
Reopen/refinement	0.000s (0.0%)	0.000s (0.0%)	0.000s (0.0%)	0.031s (3.7%)
Objective/pool update	1.902s (71.2%)	0.000s (0.0%)	0.009s (7.9%)	0.100s (12.2%)
Other	0.021s (0.8%)	0.057s (19.3%)	0.007s (6.3%)	0.083s (10.2%)

D.5 Ablation Studies

Table 18 compares direct decoding, greedy decoding, and Γ_I projection under the same TSP budget. Projection improves solution quality over direct decoding on all listed TSP sizes. On TSP-500 and TSP-1000, it is also slightly better than greedy decoding in both Drop and time in the table. This supports keeping feasibility inside the transition through Γ_I, rather than treating the raw logits as a complete set of terminal decisions. The improvement is most visible at larger sizes, where local incompatibilities in edge choices are more costly.

Table 18: TSP decoding operator ablation under E_s=50,R=1.

Method	**TSP-100**			**TSP-500**			**TSP-1000**		
	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
Direct Decoding	7.758	0.028%	0.020s	16.715	1.023%	0.205s	23.335	0.937%	0.637s
Greedy Decoding	7.756	0.003%	0.048s	16.62	0.449%	0.732s	23.22	0.457%	2.860s
Γ_I Projection	7.756	0.003%	0.041s	16.61	0.406%	0.604s	23.22	0.429%	2.672s

Table 19 compares the Transformer scorer used by PACE with a GCN scorer on TSP-100 and TSP-1000. The Transformer scorer gives both lower Drop and faster runtime in the reported settings for each listed value of E_s. On TSP-1000, the runtime gap is especially large as the edit budget increases. This ablation supports the architecture choice for the edge-oriented routing scorer in PACE.

D.6 Supplementary Comparison Summary

Table 20 summarizes, for each dataset, the strongest neural baseline reported in the preceding result tables and compares it with the corresponding PACE setting. The summary shows broad gains with visible quality-time tradeoffs. PACE improves both Drop and Time on many TSP, MIS, MVC, and ATSP entries, and it is especially fast on several node-oriented graph settings. Some rows emphasize speed, such as TSP-500 and MCL RB-SMALL, while parts of the CVRP and MCut summaries

Table 19: Ablation studies on architecture for TSP-100 and TSP-1000. PACE uses the Transformer scorer, while PACE (GCN) replaces it with a GCN scorer.

Architecture	E_s	**TSP-100**			**TSP-1000**		
		Obj.↓	Drop↓	Time↓	Obj.↓	Drop↓	Time↓
PACE (Transformer)	10	7.7562	0.005%	0.015s	23.2489	0.566%	0.629s
PACE (GCN)	10	7.7580	0.028%	0.140s	23.3347	0.937%	4.360s
PACE (Transformer)	20	7.7562	0.004%	0.022s	23.2297	0.482%	1.145s
PACE (GCN)	20	7.7570	0.015%	0.210s	23.2850	0.722%	8.080s
PACE (Transformer)	50	7.7561	0.003%	0.041s	23.2174	0.429%	2.672s
PACE (GCN)	50	7.7567	0.011%	0.390s	23.2702	0.658%	17.520s

emphasize different points on the quality-time frontier. Averaged across the summarized rows, PACE reduces Drop from 1.1918% to 1.0227%, corresponding to a 14.19% Drop reduction, and reduces average time from 2.4459s to 1.1599s, corresponding to a 2.11× speedup. This supports the overall quality-time competitiveness of PACE across the summarized rows.

Table 20: Summary comparison against the best neural baseline on each dataset.

Problem	**Dataset**	**Baseline**			**PACE**			**Drop Reduction**	**Speedup**
		Method	**Drop**	**Time**	**Setting**	**Drop**	**Time**		
TSP	TSP-50	StruDiCO	0.01%	0.06s	E_s=10,R=1	0.0014%	0.0118s	85.97%	5.09×
TSP	TSP-100	LEHD PRC 100	0.01%	0.64s	E_s=10,R=1	0.0045%	0.0151s	54.52%	42.33×
TSP	TSP-500	NEXCO (D_s=7)	0.25%	0.43s	E_s=20,R=1	0.4593%	0.3010s	-83.71%	1.43×
TSP	TSP-1000	NEXCO (D_s=7)	0.52%	1.68s	E_s=20,R=1	0.4825%	1.1446s	7.21%	1.47×
ATSP	ATSP-50	COExpander	0.17%	0.20s	E_s=50,R=1	0.1940%	0.1458s	-14.09%	1.37×
ATSP	ATSP-100	COExpander	0.95%	0.71s	E_s=50,R=1	0.3605%	0.5713s	62.05%	1.24×
ATSP	ATSP-200	COExpander	1.50%	2.64s	E_s=50,R=1	1.6262%	2.4685s	-8.41%	1.07×
ATSP	ATSP-500	COExpander	1.57%	16.48s	E_s=50,R=1	0.2466%	14.1346s	84.29%	1.17×
CVRP	CVRP-50	NEXCO (D_s=5)	0.85%	0.06s	E_s=40,R=1	1.1994%	0.0426s	-41.11%	1.41×
CVRP	CVRP-100	NEXCO (D_s=5)	1.40%	0.12s	E_s=40,R=1	1.1617%	0.1080s	17.02%	1.11×
CVRP	CVRP-200	NEXCO (D_s=5)	2.45%	0.39s	E_s=40,R=1	2.4758%	0.6059s	-1.05%	0.64×
CVRP	CVRP-500	RL4CO	4.17%	0.88s	E_s=40,R=1	3.1891%	2.6843s	23.52%	0.33×
MIS	RB-[200-300]	NEXCO (D_s=7)	1.66%	0.14s	E_s=100,R=1	1.1444%	0.0503s	31.06%	2.78×
MIS	RB-[800-1200]	NEXCO (D_s=7)	4.07%	1.00s	E_s=100,R=1	2.3595%	0.1997s	42.03%	5.01×
MIS	ER-[700-800]	NEXCO (D_s=7)	4.20%	0.56s	E_s=100,R=1	3.8328%	0.2417s	8.74%	2.32×
MIS	SATLIB	DIFUSCO	0.20%	2.96s	E_s=50,R=1	0.1831%	0.1642s	8.45%	18.03×
MCL	RB-SMALL	COExpander (S=4)	0.45%	1.74s	E_s=15,R=3	1.6583%	0.1821s	-268.52%	9.55×
MCL	RB-LARGE	COExpander (S=4)	3.08%	18.67s	E_s=15,R=1	3.0346%	0.3093s	1.47%	60.37×
MVC	RB-SMALL	COExpander (S=4)	0.37%	0.15s	E_s=80,R=1	0.1016%	0.0459s	72.55%	3.26×
MVC	RB-LARGE	COExpander (S=4)	0.16%	1.35s	E_s=80,R=1	0.1454%	0.1812s	9.14%	7.45×
MCut	BA-SMALL	COExpander (S=4)	-0.05%	0.25s	E_s=500,R=5	-0.0662%	0.9627s	32.48%	0.26×
MCut	BA-LARGE	DiffUCO: CE	-1.77%	2.70s	E_s=250,R=1	-1.2950%	0.9481s	-26.84%	2.85×
Mean	–	–	1.1918%	2.4459s	–	1.0227%	1.1599s	14.19%	2.11×

D.7 State-editing Trajectory Visualizations

Figs. 4 to 7 visualize held-out state-editing trajectories. Each trajectory starts from an empty feasible partial state, passes through intermediate committed structures, and is closed into a task-terminal feasible output.

For routing, the figures show path skeletons or customer chains as edits are committed before closure completes the tour or route set. For graph tasks, they show selected sets or partial assignments growing under the node-oriented transition. These visualizations provide qualitative support for the intended state semantics of PACE: the intermediate objects are not heatmaps or noisy full outputs, but feasible partial states that can be further edited and closed.

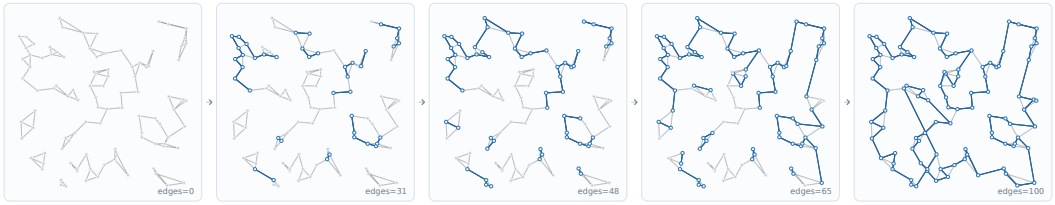


Figure 4: State editing trajectory for TSP path skeletons.



Figure 5: State editing trajectory for CVRP customer chains.

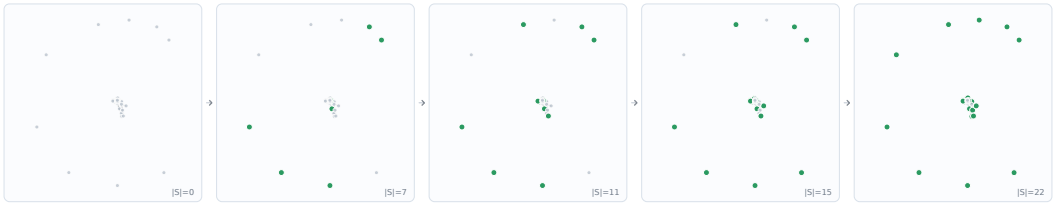


Figure 6: State editing trajectory for MIS selected sets.

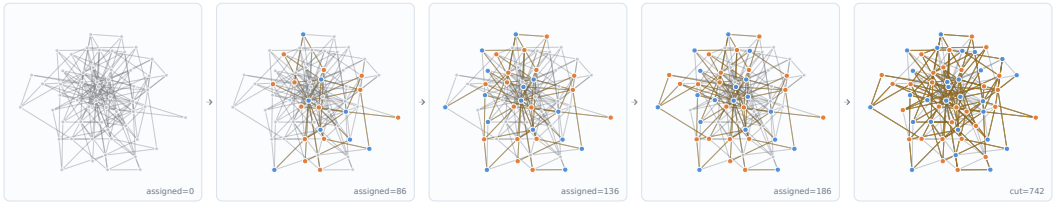


Figure 7: State editing trajectory for MCut partial assignments.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction explicitly state the claims made, including feasible partial-state semantics, the contributions, scope, and seven-task coverage. The claims match the experimental results in Sec. 5 and the structural guarantees in Sec. B.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in the Conclusion, where it states that PACE still relies on task-specific state, edit, and closure rules to instantiate the feasible partial-state semantics.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The theoretical objects, assumptions, and proofs are stated in Secs. B to 4. The results are numbered and cross-referenced.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experimental details and model settings are in Secs. C to 5. An anonymized supplemental package provides the source code for reproducing the reported experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: An anonymized supplemental package accompanying the submission provides the PACE code together with reproduction scripts and usage instructions for the reported experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The datasets, metrics, and main inference settings are summarized in Sec. 5. Training data, hyperparameters, model configurations, hardware, and runtime measurement details are reported in Sec. C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We follow the setting of previous ML4CO works to report the average solution quality and runtime over held-out instances in Sec. 5. We do not report separate error bars or significance tests.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The hardware information is in Sec. C, including the Intel(R) Xeon(R) Gold 6348 CPU and NVIDIA A100 40GB GPU used for training and testing. Runtime breakdowns for representative benchmarks are reported in Table 17.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We reviewed the NeurIPS Code of Ethics, and the research described in the paper conforms to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: This paper studies a foundational method for benchmark combinatorial optimization and does not target a specific deployment setting with direct societal impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk generative models, scraped datasets, or other assets that would require special misuse safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The original papers that introduce the models, datasets, benchmarks, and solvers used in the paper are cited in Secs. C and 5.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: An anonymized supplemental package accompanying the submission provides the PACE code together with documentation and scripts for reproducing the reported experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects research and therefore does not require IRB or equivalent review.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not an important, original, or non-standard component of the core methodology in this research.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.