

FedCSIS 2026 · Paper Presentation

# Preserving Optimization Algorithm Expertise by means of Executable Algorithm Knowledge Graphs

A Worked Example on the TSP

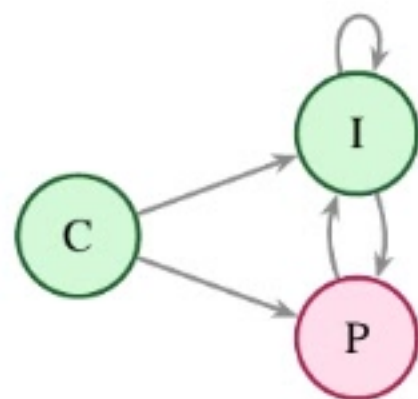
Camilo Chacón Sartori · José H. García · Andrei V. Tomut · Christian Blum

Catalan Institute of Nanoscience and Nanotechnology (ICN2), CSIC and BIST · Artificial Intelligence Research Institute (IIIA-CSIC) · Bellaterra, Spain

## Where does algorithm expertise live?

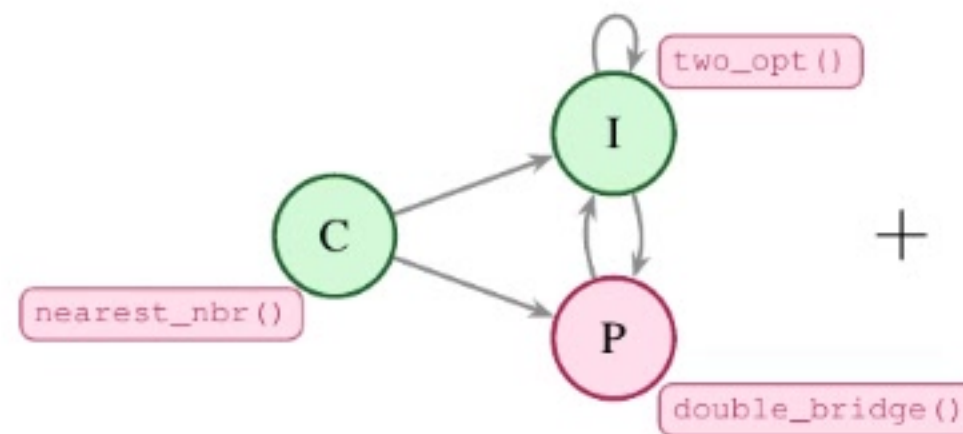
- Inside source code: which operator, in which order, under which conditions
- Inside the transient search traces of a solver run
- New problem: the logic is rewritten and the record of what worked is discarded

# A GEAKG has three separable layers



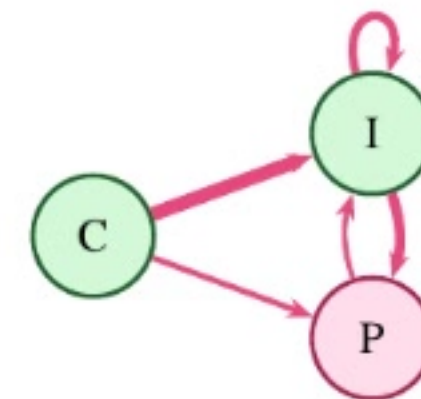
**L0** typed topology  
admissible category  
transitions only

+



**L1** executable operators  
validated code bound  
to each role

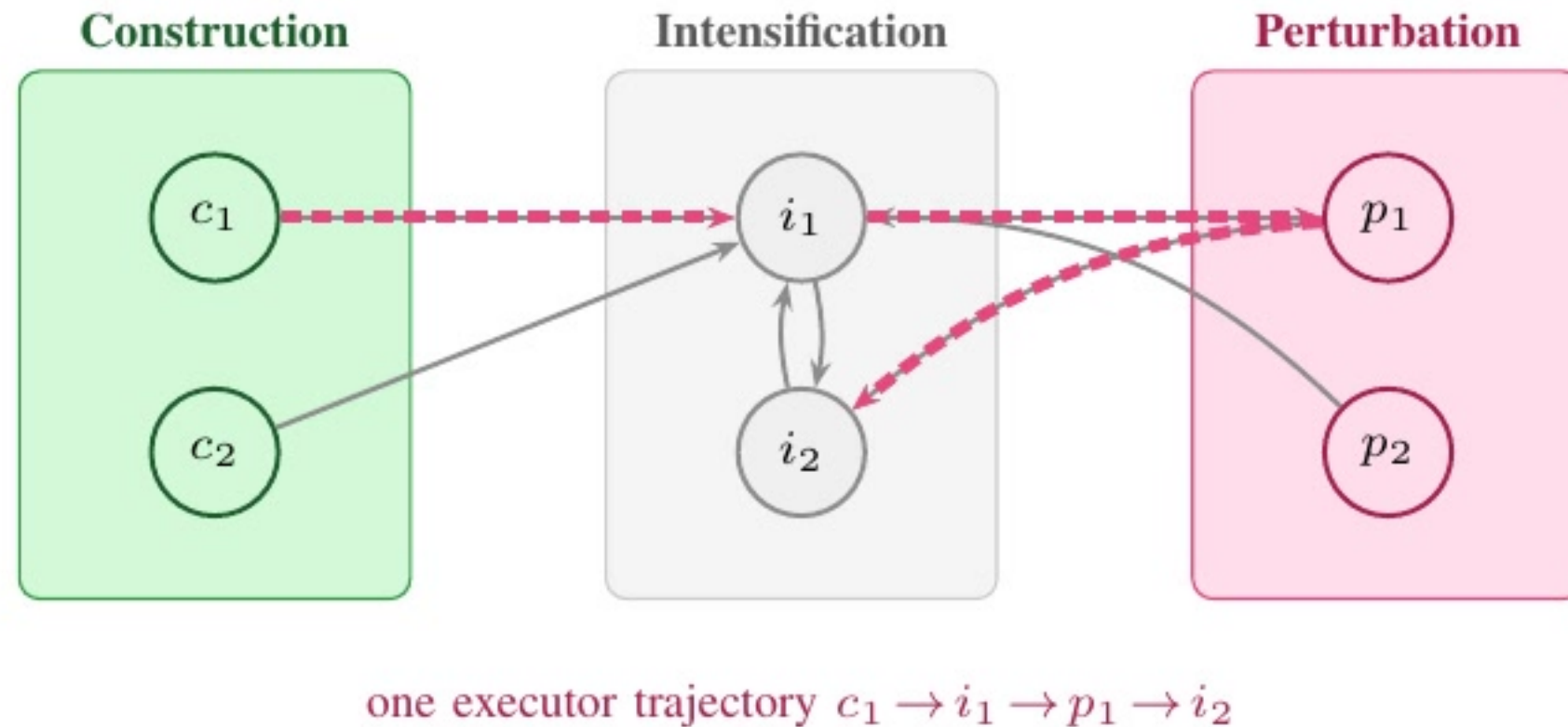
+



**L2** learned weights  
edge thickness  $\propto$  pheromone  
(fitted offline, then frozen)

L0 typed topology · L1 validated executable operators · L2 learned pheromone weights

## Step 1: the role schema (a human derives it from Iterated Local Search)



Three categories, eleven roles, seven admissible category transitions, fourteen edges

# The eleven roles, with a representative operator each

TABLE II  
THE ELEVEN ROLES OF THE PERMUTATION-OPTIMIZATION SCHEMA,  
GROUPED BY CATEGORY, WITH A REPRESENTATIVE OPERATOR FOR EACH.  
THE MIDDLE CATEGORY (LOCAL SEARCH) REALIZES INTENSIFICATION.

| Category                    | Role                | Representative operator |
|-----------------------------|---------------------|-------------------------|
| Construction                | const_greedy        | nearest neighbor        |
|                             | const_insertion     | cheapest insertion      |
|                             | const_savings       | Clarke–Wright savings   |
|                             | const_random        | randomized build        |
| Local search<br>(intensify) | ls_intensify_small  | 2-opt, swap             |
|                             | ls_intensify_medium | 3-opt, or-opt           |
|                             | ls_intensify_large  | Lin–Kernighan           |
|                             | ls_chain            | chained / VND           |
| Perturbation                | pert_escape_small   | double bridge           |
|                             | pert_escape_large   | ruin-and-recreate       |
|                             | pert_adaptive       | history-based escape    |

Table II of the paper

## Step 2: generate operators, admit only what runs and returns a feasible tour

```
def two_opt(tour, D, max_iter=1000,
            **kw):
    tour = tour.copy(); n = len(tour)
    improved = True
    while improved:
        improved = False
        for i in range(n - 1):
            for j in range(i + 2, n):
                a = tour[i]
                b = tour[i+1]
                c = tour[j]
                d = tour[(j+1) % n]
                g = D[a][c] + D[b][d]
                g -= D[a][b] + D[c][d]
                if g < 0:
                    s = tour[i+1:j+1]
                    s.reverse()
                    tour[i+1:j+1] = s
                    improved = True
    return tour
```

A 2-opt operator bound to ls\_intensify\_small (Listing 2)

## Step 3: assemble the topology and seed the weights

```
You are an expert in combinatorial optimization
metaheuristics.
Return a meta-algorithm for TSP/JSSP/VRP.

AVAILABLE ROLES
CONSTRUCTION: const_greedy, const_insertion,
               const_savings, const_random
LOCAL SEARCH: ls_intensify_{small,medium,large},
              ls_chain
PERTURBATION: pert_escape_{small,large},
              pert_adaptive

TASK
1. Select 8-12 roles.
2. Define 12-18 weighted edges (0.0-1.0).
3. Escape edges (ls -> perturbation) MUST
   carry a condition.

REQUIRED CONDITIONAL EDGE
ls_intensify_small -> pert_escape_small
{when: stagnation, threshold: 3, boost: 2.0}

WEIGHT GUIDELINES
construction -> ls : 0.80-0.90
ls -> ls (escalate): 0.65-0.80
ls -> perturbation : 0.30-0.50 (conditional)
perturbation -> ls : 0.85-0.95
Design seed: <<SEED>>
```

The generation template: role catalog, weight guidelines, one required conditional edge (Listing 3)

## Step 4: learn edge weights offline with the MAX-MIN Ant System

---

**Algorithm 1** Offline L2 weight learning (MMAS)

---

**Require:** topology  $(V, E)$ , operators  $\Lambda$ , problem instances  $\mathcal{I}$ ,  
maximum number of iterations  $N$

- 1: initialize  $\Phi(e) \leftarrow \tau_{\max}$  for all  $e \in E$
  - 2: **for**  $t = 1$  **to**  $N$  **do**
  - 3:   sample trajectories  $\{\tau^{(k)}\}$  by weighted role selection  
    over  $E$
  - 4:   evaluate each  $\tau^{(k)}$  on  $\mathcal{I}$ , record tour cost
  - 5:    $\Phi \leftarrow (1 - \rho) \Phi$  // evaporation
  - 6:   reinforce edges of the best trajectory:  $\Phi(e) += \Delta$
  - 7:   clamp  $\Phi(e) \leftarrow \min(\max(\Phi(e), \tau_{\min}), \tau_{\max})$
  - 8: **end for**
  - 9: **return** frozen  $\Phi$
- 

Pheromones evaporate, reinforce along the best trajectory, and stay clamped to a bounded interval

## Running it: the Symbolic Executor, zero language-model tokens

---

**Algorithm 2** Symbolic Executor (zero runtime model tokens)

---

**Require:** snapshot  $(\mathcal{S}, V, E, \Lambda, \Phi, \Sigma)$ , instance  $x$ , budget  $B$

1:  $s \leftarrow$  entry role in  $K_0$ ;  $\text{best} \leftarrow$  initial tour

2: **for**  $b = 1$  **to**  $B$  **do**

3:    $A \leftarrow \{v \mid (s, v) \in E\}$     // L0-admissible successors

4:   sample  $v \in A$  with prob.  $\propto \Phi(s, v) \cdot \beta_{\Sigma}(s, v)$     //  $\beta_{\Sigma}$ :  
rule boost

5:    $o \leftarrow$  operator from  $\Lambda(v)$ ;  $\text{cand} \leftarrow o(\text{best}, x)$

6:   **if**  $\text{cost}(\text{cand}) < \text{cost}(\text{best})$  **then**

7:      $\text{best} \leftarrow \text{cand}$

8:   **end if**

9:    $s \leftarrow v$

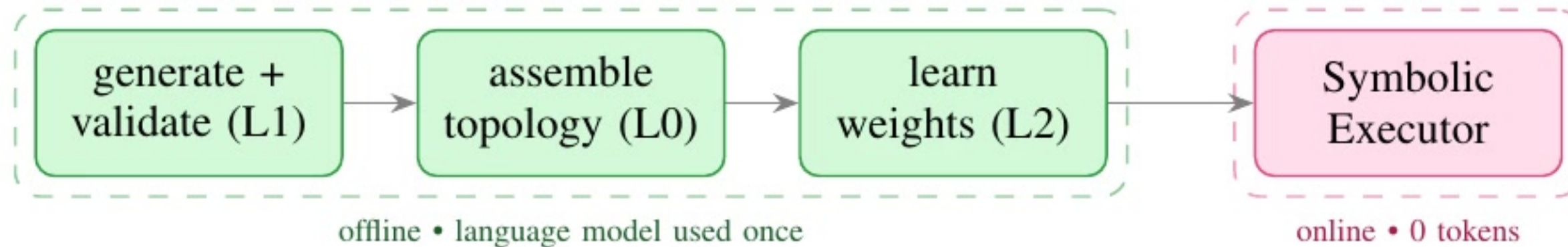
10: **end for**

11: **return**  $\text{best}$

---

Sample the next role among admissible successors, apply one operator, accept only on improvement

## Lifecycle: the language model is used once, offline



Acquisition and refinement offline (green); execution online at 0 tokens (pink)

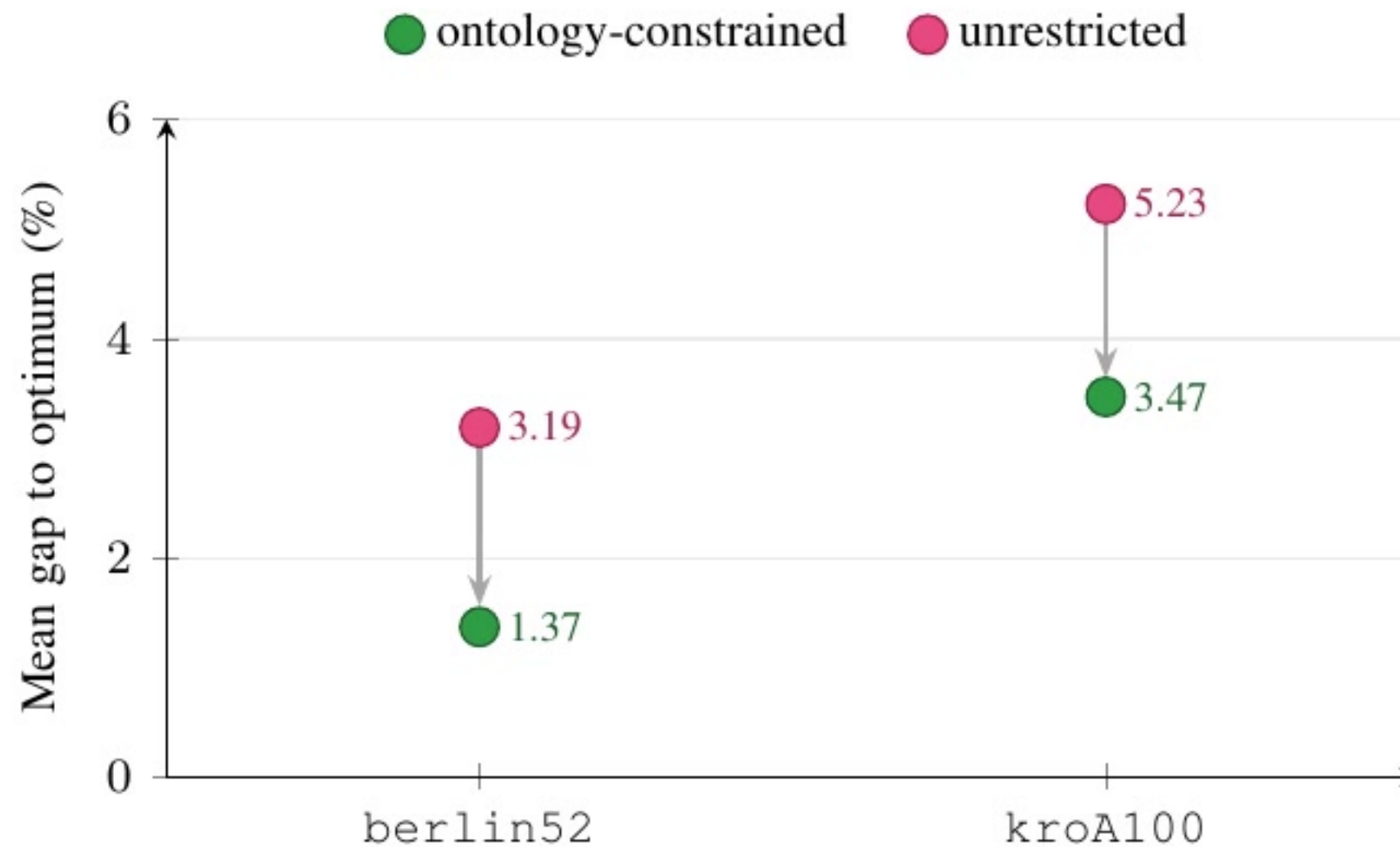
# Result 1: feasible tours at every model scale

TABLE III  
SINGLE-DOMAIN TSP SUMMARY. FEASIBLE TOURS ARE OUT OF SEVEN INSTANCES; DEPLOYMENT USES NO LANGUAGE-MODEL TOKEN REGARDLESS OF WHICH MODEL SCALE SYNTHESIZED THE OPERATORS OFFLINE.

| Configuration                  | Feasible | Deploy. tokens |
|--------------------------------|----------|----------------|
| GEAKG (gpt-5.2 operators)      | 7 / 7    | 0              |
| GEAKG (gpt-4o-mini operators)  | 7 / 7    | 0              |
| GEAKG (Qwen2.5-14B, local)     | 7 / 7    | 0              |
| LLaMEA standalone (local, 10k) | 1 / 7    | —              |

Seven TSPLIB instances; operators synthesized by gpt-5.2, gpt-4o-mini, and a local Qwen2.5-14B

## Result 2: the ontology constraint lowers the gap, it does not only prune

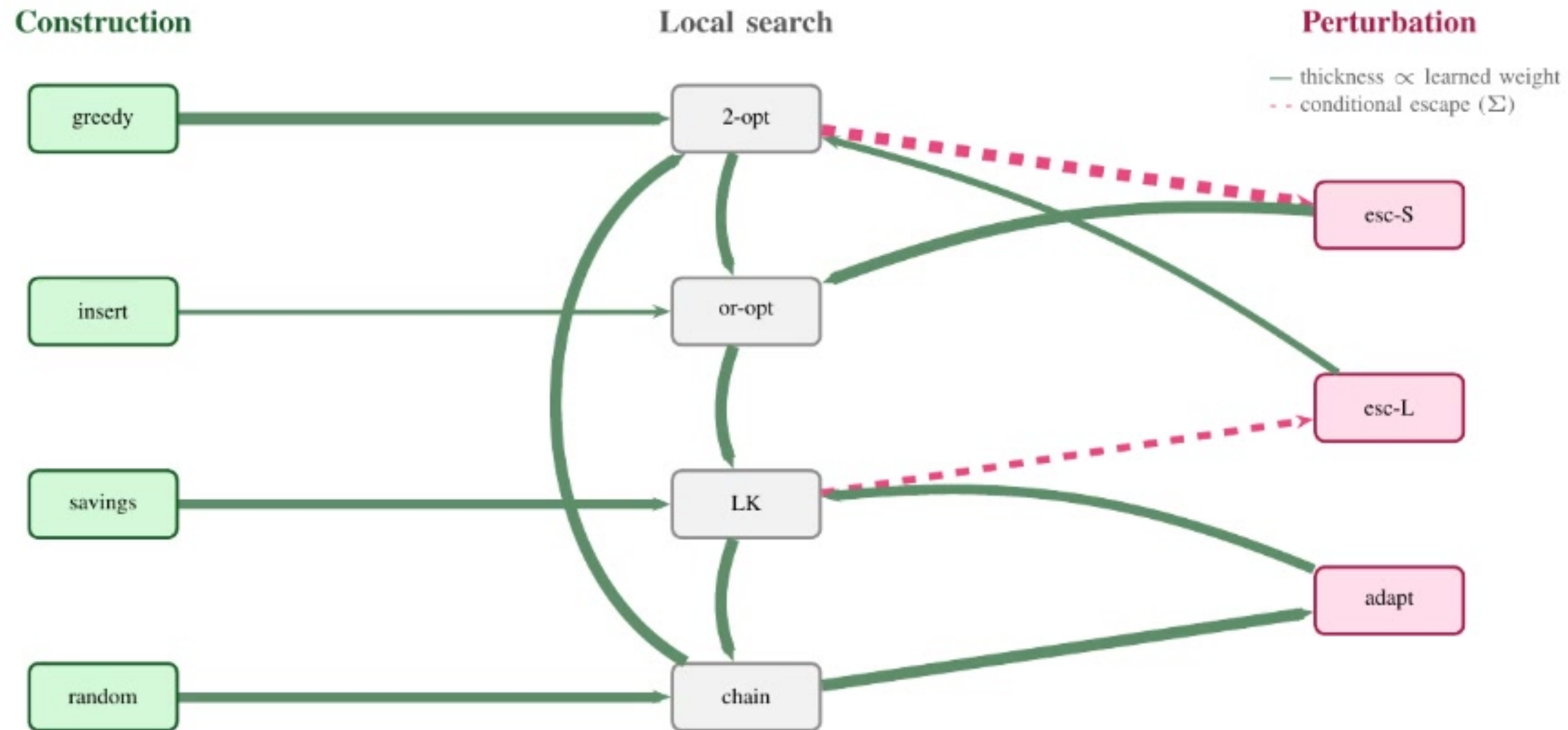


Mean gap to optimum, eight seeds, ant colony trained from scratch on each topology

## Result 3: is each layer load-bearing?

- Learned weights vs uniform weights: lower gap on 3 of 4 instances; largest gain on pr226 (1.58 points)
- Every operator replaced by a no-op: gap inflates to 298% to 1978%
- Optimality gaps of 3.4% to 8.0% across four instances, eight seeds, at zero deployment tokens

# The artifact: a kilobyte-scale snapshot you can read



Eleven roles, fourteen edges; thickness is the learned weight; dashed pink edges are conditional escapes

## Limitations the paper states

- The role schema requires human modeling (hours, not days, for the TSP)
- No online weight update: deployment behavior is fixed by the offline snapshot
- All evidence is single-domain by design; cross-domain transfer is deferred to the companion paper

## Takeaways

- Procedural know-how stored as a typed, executable, inspectable graph
- Language model used once, offline; deterministic executor at deployment
- Full version with the cross-domain study: [arXiv:2603.27922](https://arxiv.org/abs/2603.27922), "GEAKG: Generative Executable Algorithm Knowledge Graphs"