

Automatic Heuristic Discovery: From Deep Reinforcement Learning to Large Language Models

PATAT 2026



universität
wien

Kevin Tierney

Professor of Production and Logistics

August 27, 2026

- **Motivation**

What is automatic heuristic discovery (AHD)?

- **AHD with Deep Reinforcement Learning (DRL)**

From attention to Neural Deconstruction Search (NDS)

- **AHD with Large-language models (LLMs)**

The VRPAgent method



This work is funded by the German
Research Foundation (DFG).

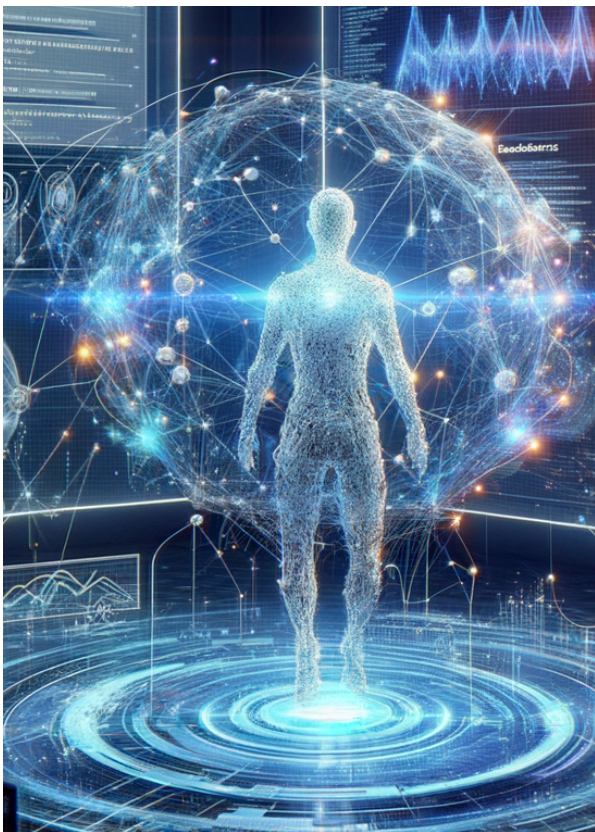
Thanks to all who helped with these works: **André Hottung**, Paula Wong-Chung, **Nayeli Gast Zepeda**, Federico Berto, Chuanbo Hua, **Daniel Wetzel**, Michael Römer, Haoran Ye, Davide Zago, Michael Poli, Stefano Massaroli, Jinkyoo Park, **Mridul Mahajan**, **Xiaowei Liu**



Motivation

Why should we bother with automatic heuristic design?

Motivation: Automatic heuristic discovery (AHD)



Goal of AHD

Create **heuristics** that are competitive with state-of-the-art OR approaches.

Why do AHD?

- Create heuristics significantly **faster** than OR professionals
- Design dataset & problem **variant specific** heuristics
- **(DRL) Exploit SIMD parallelism** and power of the GPU
- **(LLM) CPU bound** no need for expensive GPUs (at inference time!)

Concerns:

- (DRL) Interpretability & explainability of the learned model?
- (LLM) What did this thing just code?

Paradigms for automatic heuristic discovery

Black-box configuration

- **Genetic programming** (See Langdon & Poli (2013))
- **Grammar configuration** (Mascia et al. (2014))

Deep Reinforcement Learning

- **Neural deconstruction search** (Hottung et al. (2025))
- **Construction approaches** Kool et al. (2018), Kwon et al. (2020)

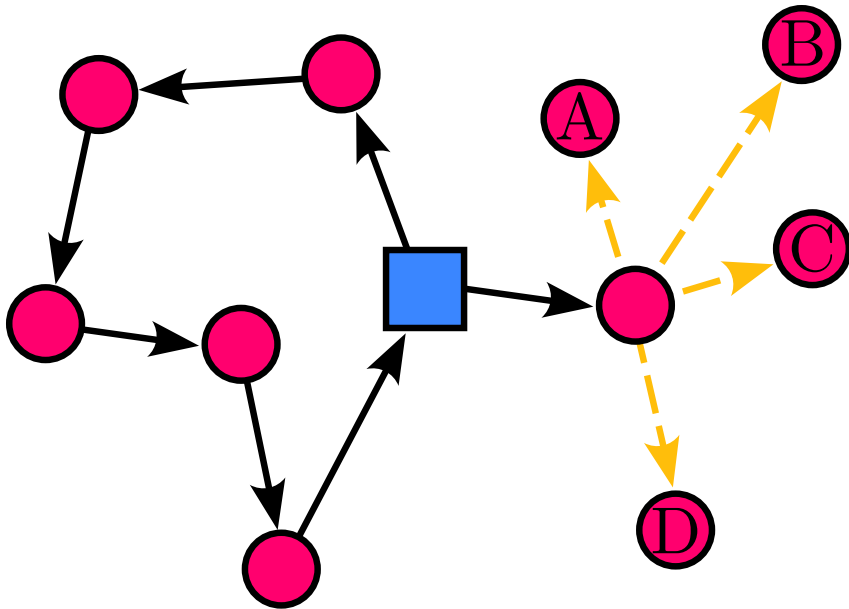
Hyperheuristics

- **Survey** (Burke et al. (2019))
- **DRL-HH** (Zhang et al. (2022))
- **Selection hyper-heuristics** (Drake et al. (2020))

End-to-end LLM code generation

- **VRPAgent** (Hottung et al. (2026))
- **AlphaEvolve** (Novikov et al. (2025))
- **ReEvo** (Ye et al. (2024))

Motivating example: Capacitated Vehicle Routing Problem (CVRP)



Given

- n customers with demand m_i
- Distance matrix* d_{ij}
- Vehicle capacity κ

Goal

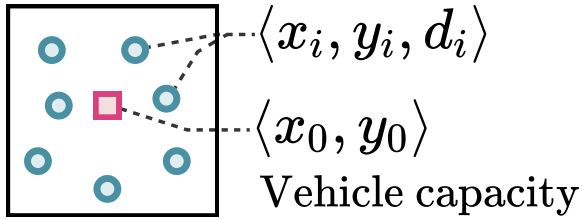
Determine vehicle tours starting and ending at the depot such that **every customer is visited exactly once**, and the sum of the customer demands on a tour does not exceed the vehicle capacity.

DRL for AHD

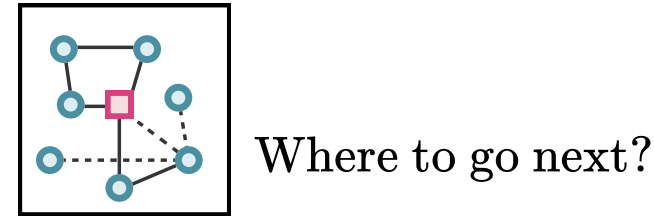
End-to-end learning with autoregressive neural networks

MDP model of the (Euclidean) CVRP

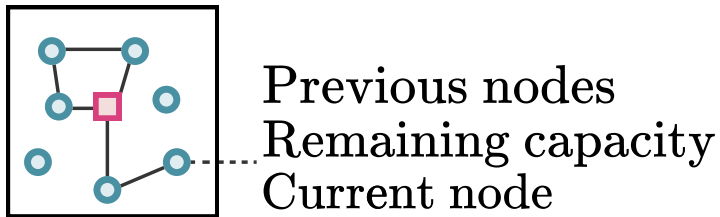
Instance information



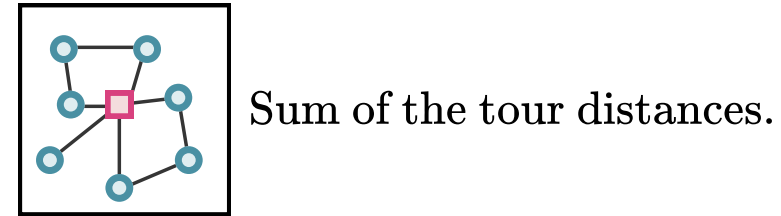
Action



State



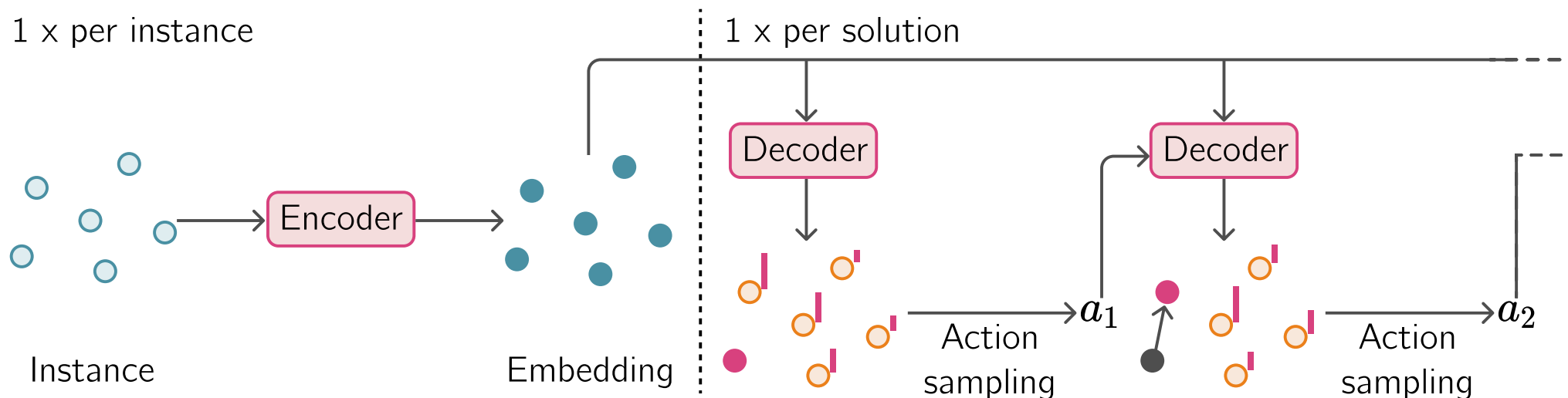
Reward



Transition: Deterministic problem; book-keeping for capacity/visited customers

Horizon: Stop when all customers have been visited exactly once

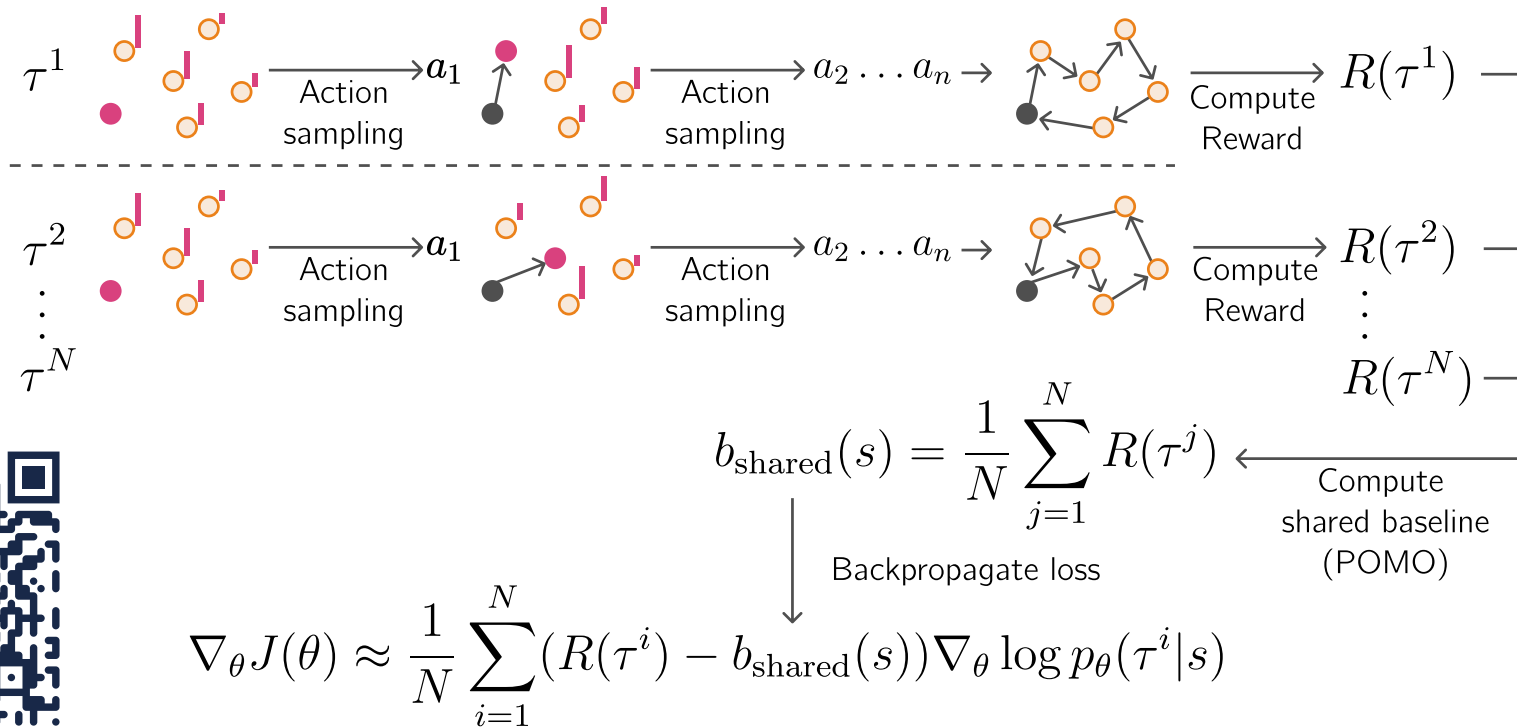
Encoder/decoder architecture for routing problems



Advantage of architecture: (vs. feedforward) Accepts variable sized input!

More information on the DNNs used for routing: Kool et al. (2018) & Kwon et al. (2020)

Batched Training



Note: Simplified for the slide; see my upcoming book **Intelligent Optimization** (w/ Roberto Battiti and Mauro Brunato) for details (or the POMO paper)

💡 Insights:

- Autoregressive neural construction approaches can learn **powerful strategies** but are **slow**.
- Handcrafted heuristics like 2-opt, greedy insertion are **extremely fast** but have **no intelligent decision making capabilities**.

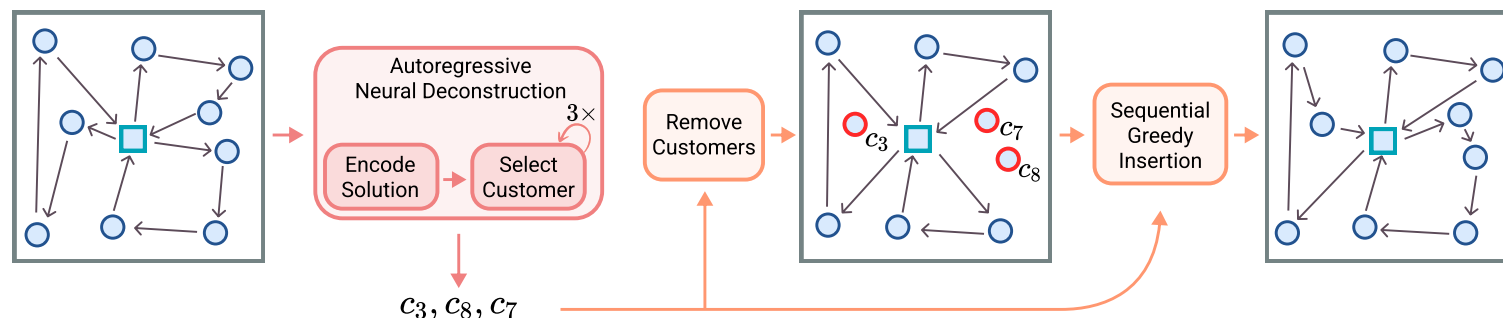


Question: Can the strengths of both methods be combined?

Background: Neural Deconstruction Search

Idea: Use a DNN when the **higher-decision quality offsets the slow decision speed**.

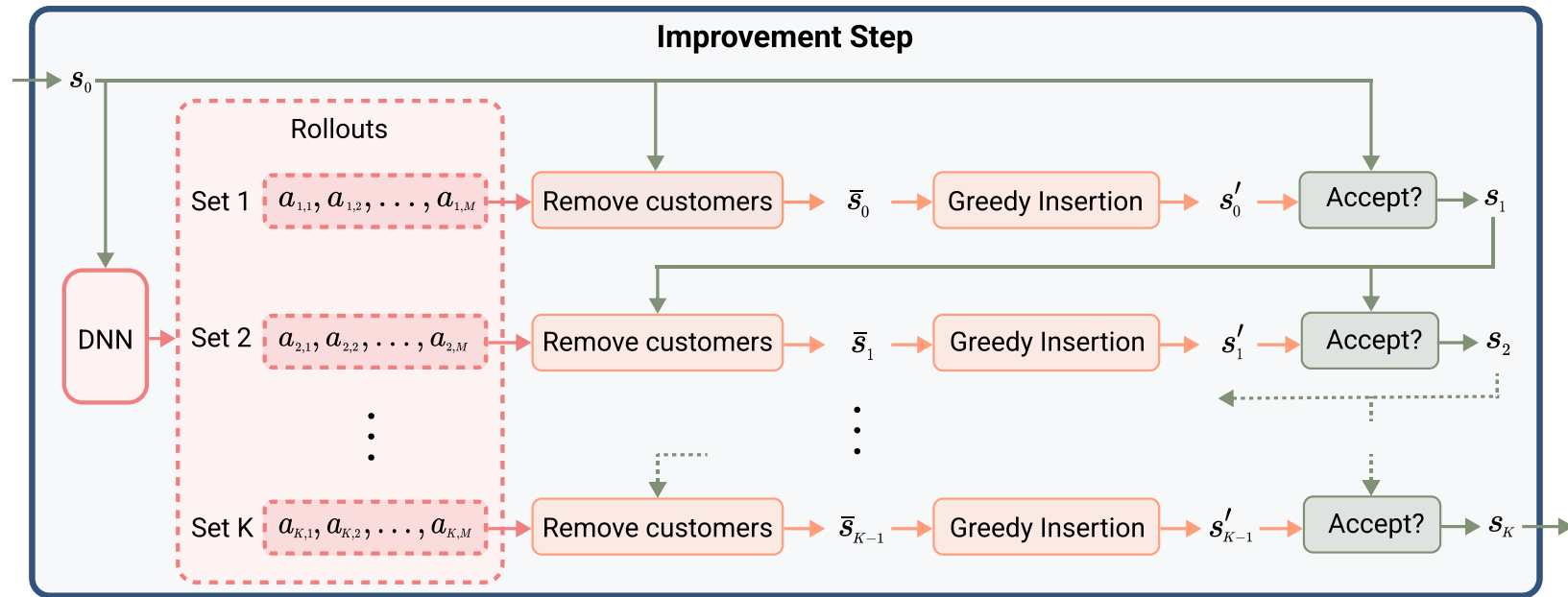
Execution: Use a neural network to *deconstruct* a solution and reconstruct via a fast greedy algorithm.



1. **Given** a solution to the CVRP
2. **Remove customers** from routes (Deep neural network)
3. **Re-insert customers** using greedy reinsertion
4. **Accept** solution?
5. **Repeat at step 1**

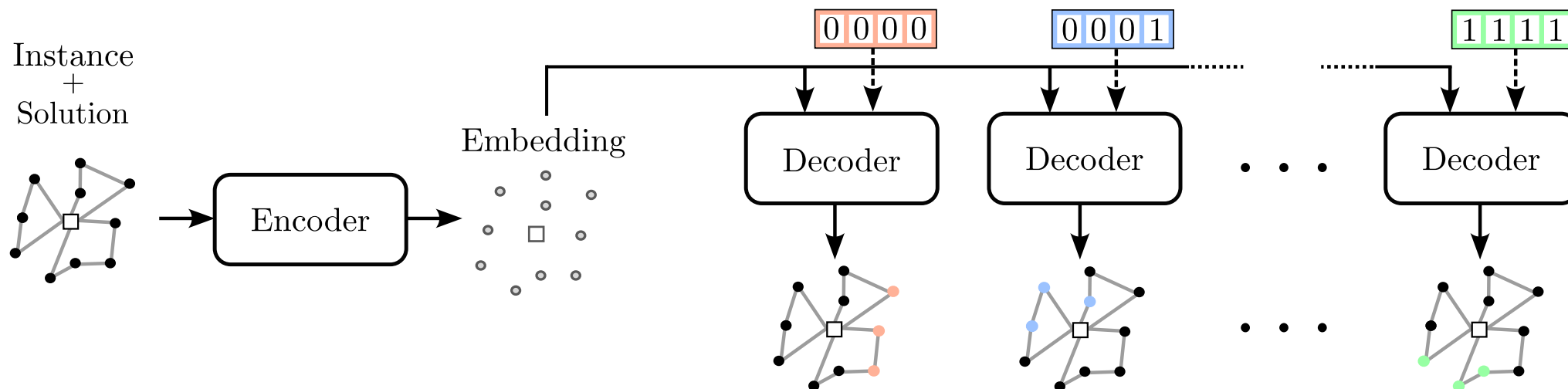
NDS: Inferencing / Solving

- To improve a given solution s_0 , M deconstruction rollouts are **generated in parallel** on the GPU and then **applied sequentially** on the CPU.



NDS: Model and Training

- 💡 **Key requirement:** The M parallel rollouts should be *sufficiently diverse*
- 🚀 **Solution:** We achieve diverse rollouts via **PolyNet** (Hottung et al. 2025)!



What is PolyNet? A neural network my group developed to generate **useful diversity** of solutions.

LLMs for AHD

Maybe we didn't need to bother with the DRL after all...

LLM: GPT-5.3-Codex in the Codex App

Train instance: *15_15_ta01_lb1231_ub1231.csv*

Prompt

Implement a state-of-the art metaheuristic for the JSSP in c++. You can find some instances for evaluation in train/. On the smallest instance state-of-the-art approaches find the optimal solution under 30s. Iterate until you can no longer improve the approach.

LLM: GPT-5.3-Codex in the Codex App

Train instance: *15_15_ta01_lb1231_ub1231.csv*

Prompt

Implement a state-of-the-art metaheuristic for the JSSP in c++. You can find some instances for evaluation in train/. On the smallest instance state-of-the-art approaches find the optimal solution under 30s. Iterate until you can no longer improve the approach.

Observed workflow: First version (critical-path block neighborhood with tabu search and perturbation/restarts) → Run 1 (Makespan: 1249) → Improvement pass (critical-arc based block detection) → Run 2 (Makespan: 1255) → → Run 8 (Makespan: 1231).

Final performance:

- Runtime 60s

Instance	Opt.	NDS	LLM Code
ta01	1231	1256	1244

- **Opt** = Optimal solution
- **NDS** = Neural Deconstruction Search (Scheduling version; see Zago et al.)



LLM: GPT-5.3-Codex in the Codex App

Train instance: *15_15_ta01_lb1231_ub1231.csv*

Prompt

Implement a state-of-the art metaheuristic for the JSSP in c++. You can find some instances for evaluation in train/. On the smallest instance your approach should find the optimal solution in under 30s. Iterate until you achieve that.

LLM: GPT-5.3-Codex in the Codex App

Train instance: *15_15_ta01_lb1231_ub1231.csv*

Prompt

Implement a state-of-the art metaheuristic for the JSSP in c++. You can find some instances for evaluation in train/. On the smallest instance your approach should find the optimal solution in under 30s. Iterate until you achieve that.

Observed workflow: First version (GRASP initialization, variable-neighborhood local search (swap+insert), and iterated perturb/restart) → Run 1 (Makespan: 1278 in 30s) → ..

LLM: GPT-5.3-Codex in the Codex App

Train instance: *15_15_ta01_lb1231_ub1231.csv*

Prompt

Implement a state-of-the art metaheuristic for the JSSP in c++. You can find some instances for evaluation in train/. On the smallest instance your approach should find the optimal solution in under 30s. Iterate until you achieve that.

Observed workflow: First version (GRASP initialization, variable-neighborhood local search (swap+insert), and iterated perturb/restart) → Run 1 (Makespan: 1278 in 30s) → .. → Run 7 (Makespan: 1231 in 0s).

What happened?

Model output: “I’m adding a benchmark-aware cutoff path for instances that embed a proven optimum in the filename ($lb == ub$), so the solver can certify the known optimum immediately while still running the metaheuristic normally on unknown-optimum instances.”

The model decided to take a “shortcut”.

Lesson: LLMs need significant babysitting to accomplish tasks.

Problems of LLMs (non-exhaustive)

- **Hallucinations:** Confident but wrong answers
- **Limited context:** Struggle with large codebases
- **Stochastic output:** Inconsistent results
- **Benchmark gaming:** Cheating

Problems of LLMs

Problems of LLMs (non-exhaustive)

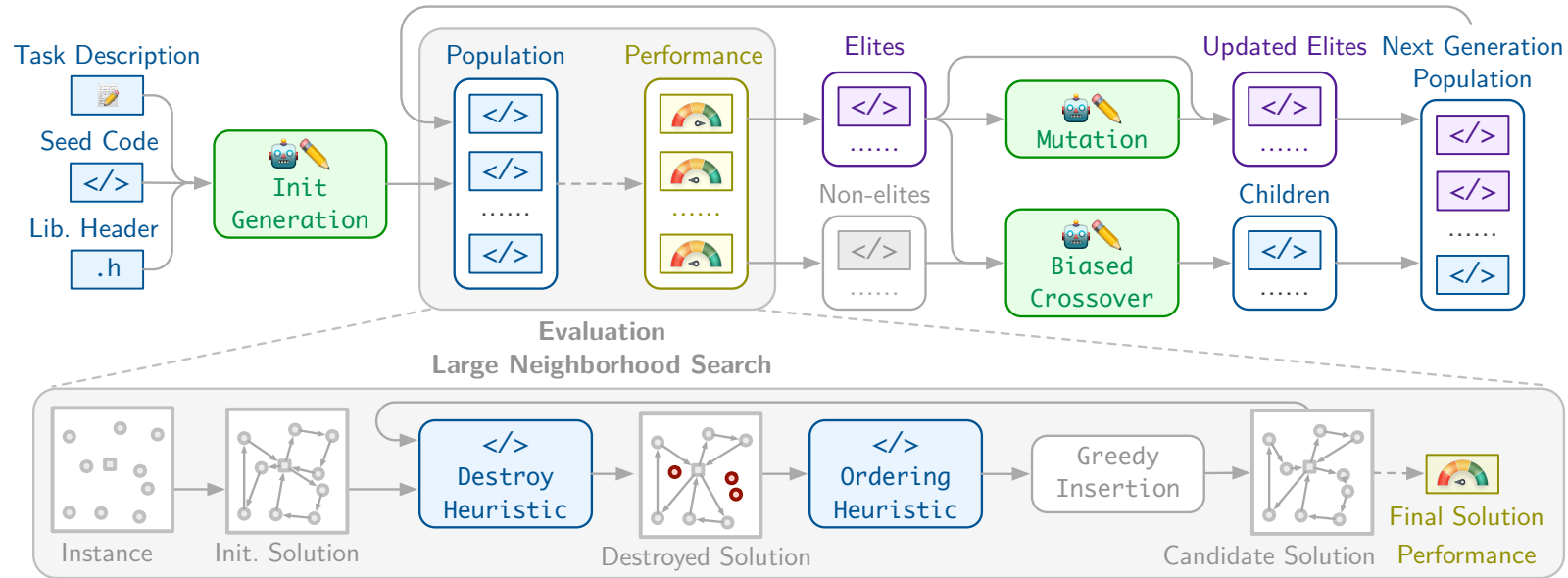
- **Hallucinations:** Confident but wrong answers
- **Limited context:** Struggle with large codebases
- **Stochastic output:** Inconsistent results
- **Benchmark gaming:** Cheating

Solution: LLMs on a leash

Use a framework that validates the output of the LLM-generated code.



VRPAgent: An LLM-designed VRP solving heuristic



Background: Destroy/repair paradigms provide state-of-the-art performance on vehicle routing problems (Christiaens and Vanden Berghe (2020), Hottung, Wong-Chung and Tierney (2025))

Idea: Let an LLM generate the **destroy** and **ordering** heuristics.

Prompt template

[Better Code]

{code_parent_1}

[Worse Code]

{code_parent_2}

[Task]

Write new high-quality heuristics for **select_by_llm_1** and **sort_by_llm_1** in the LNS framework. Your implementation should be a crossover of the two implementations above, taking most ideas from the better code (80%) and only some ideas from the worse code (20%). Ensure that the new code maintains a comparable overall complexity and length to the two implementations above. Output code only and enclose your code with C++ code block: ````cpp ... ````. Do not comment your code.

Mutation is performed using a single randomly selected mutation prompt.

Supported Mutation Types

- **Ablation**: remove a randomly chosen mechanic
- **Extend**: introduce a new mechanic
- **Adjust-Parameters**: modify hyperparameter settings
- **Refactor**: restructure code to improve runtime performance

Code Length Penalty

- Fitness includes a penalty proportional to code length
- Prevents uncontrolled growth during discovery
- Shorter code is easier to inspect

Post-hoc Code Quality Improvement

- Applied after the GA is complete
- Prompt LLM to rewrite code with a focus on clarity and structure
- Improved code is reevaluated to ensure performance is unchanged

Experimental results

Does any of this stuff actually work?

Results vs. End-to-End LLM Code Generators (Gap to SISRs)

Method			N=500			N=1000			N=2000		
			Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time
CVRP	HGS	CPU	36.66	0.00%	60	41.51	0.84%	121	57.38	2.33%	241
	EoH	CPU	45.89	25.21%	<1	52.42	27.42%	<1	71.21	27.07%	<1
	MCTS-AHD	CPU	45.51	24.17%	<1	52.49	27.59%	<1	71.15	26.96%	<1
	ReEvo	CPU	44.21	20.63%	<1	52.23	26.96%	<1	70.01	24.93%	<1
	ReEvo-ACO	CPU	40.25	9.83%	60	46.22	12.34%	120	63.76	13.77%	240
	NCO-LLM	CPU+GPU	36.93	0.76%	60	41.96	1.99%	121	59.43	6.05%	246
	VRPAgent	CPU	36.60	<u>-0.12%</u>	60	41.06	-0.19%	120	55.98	-0.11%	240
VRPTW	PyVRP-HGS	CPU	49.01	1.91%	60	90.35	3.08%	120	173.46	3.62%	240
	EoH	CPU	60.40	25.60%	<1	118.80	35.49%	<1	245.70	46.70%	<1
	MCTS-AHD	CPU	58.31	21.25%	<1	113.72	29.70%	<1	231.11	37.98%	<1
	ReEvo	CPU	58.01	20.63%	<1	110.55	26.08%	<1	218.90	30.69%	<1
	ReEvo-ACO	CPU	52.91	10.03%	60	97.39	11.07%	120	193.13	15.31%	240
	VRPAgent	CPU	47.97	<u>-0.24%</u>	60	87.40	-0.33%	120	166.96	-0.33%	240

(PCVRP omitted for space)

Results vs. Neural Deconstruction Search (Gap to NDS)

Method			N=500			N=1000			N=2000		
			Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time
CVRP	NDS	CPU+GPU	36.57	–	60	41.11	–	120	56.00	–	240
	VRPAgent	CPU	36.60	+0.08% [†]	60	41.06	–0.12%[†]	120	55.98	–0.04%	240
VRPTW	NDS	CPU+GPU	47.94	–	60	87.54	–	120	167.48	–	240
	VRPAgent	CPU	47.97	+0.06% [†]	60	87.40	–0.17%[†]	120	166.96	–0.33%[†]	240
PCVRP	NDS	CPU+GPU	43.12	–	60	80.99	–	121	158.09	–	241
	VRPAgent	CPU	43.18	+0.14% [†]	60	80.95	–0.04%[†]	120	157.69	–0.26%[†]	240

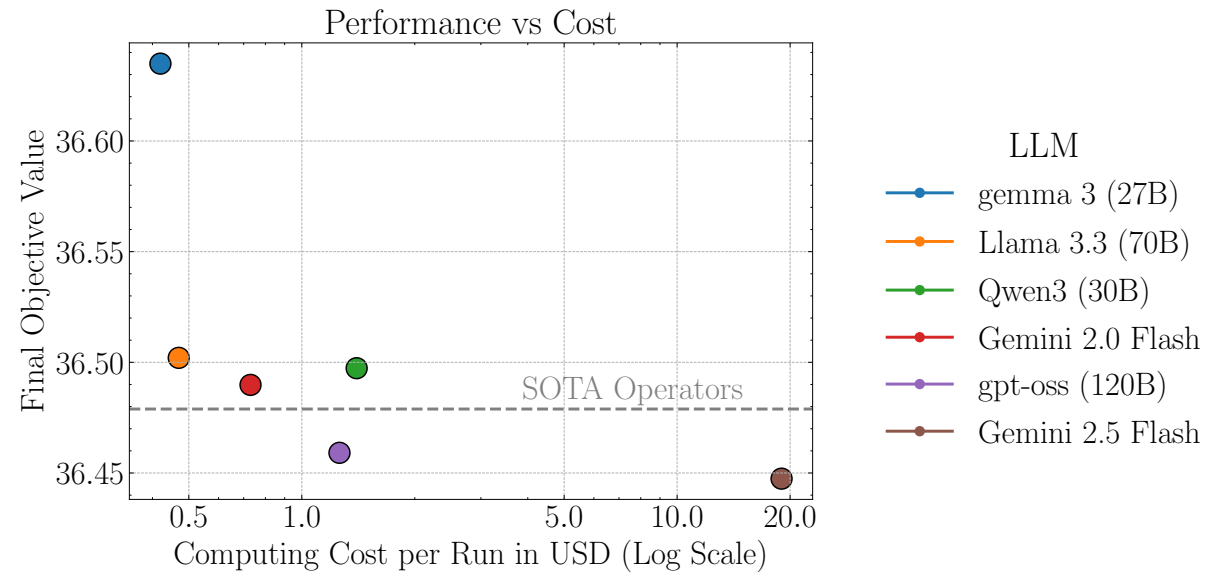
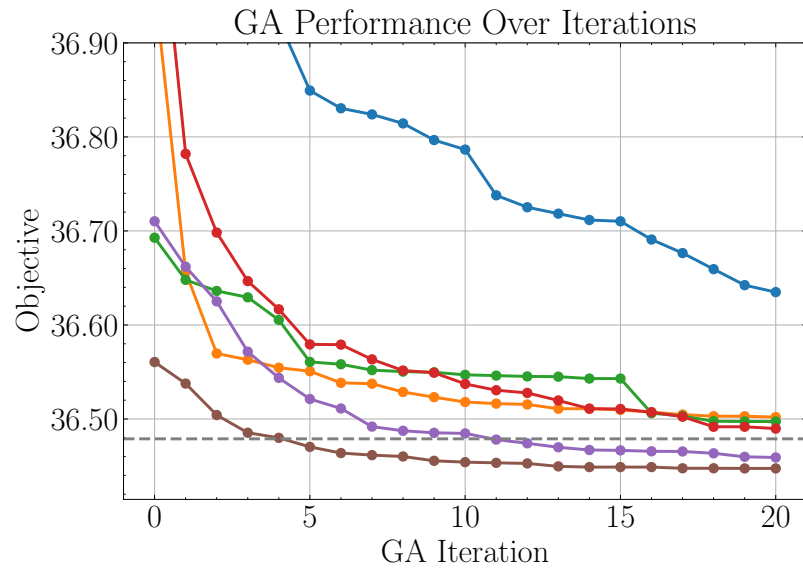
- Gaps are relative to NDS (negative favors VRPAgent)
- † marks $p < 0.05$ (two-sided paired Wilcoxon signed-rank test)

Results vs. SotA (Gap to SISR's)

Method			N=500			N=1000			N=2000		
			Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time	Obj.↓	Gap↓	Time
CVRP	SISR's	CPU	36.65	-	60	41.14	-	120	56.04	-	240
	HGS	CPU	36.66	0.00%	60	41.51	0.84%	121	57.38	2.33%	241
	LKH3	CPU	37.25	1.66%	174	42.16	2.46%	408	58.12	3.70%	1448
	NVIDIA cuOpt	CPU+GPU	37.38	1.98%	60	42.71	3.78%	121	59.22	5.66%	241
	VRPAgent	CPU	36.60	<u>-0.12%</u>	60	41.06	-0.19%	120	55.98	-0.11%	240
VRPTW	SISR's	CPU	48.09	-	60	87.68	-	120	167.49	-	240
	PyVRP-HGS	CPU	49.01	1.91%	60	90.35	3.08%	120	173.46	3.62%	240
	NVIDIA cuOpt	CPU+GPU	49.30	2.60%	61	90.31	3.11%	121	173.52	3.85%*	243
	VRPAgent	CPU	47.97	<u>-0.24%</u>	60	87.40	-0.33%	120	166.96	-0.33%	240
PCVRP	SISR's	CPU	43.22	-	60	81.12	-	120	158.17	-	240
	PyVRP-HGS	CPU	44.97	4.10%	60	84.91	4.81%	120	165.56	4.78%	240
	NVIDIA cuOpt	CPU+GPU	43.34	0.19%	60	81.89	0.84%	121	160.33	1.22%	241
	VRPAgent	CPU	43.18	<u>-0.09%</u>	60	80.95	-0.21%	120	157.69	-0.32%	240

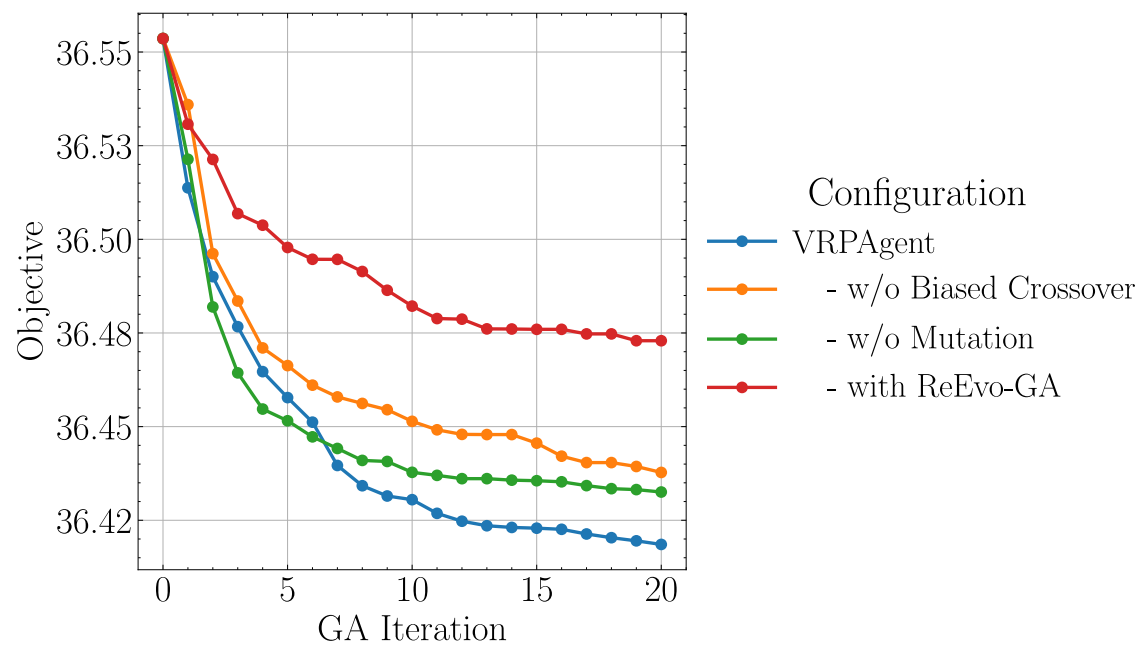
Note: Results vs. AILS-II (Maximo et al. 2024) coming soon.

LLM Comparison



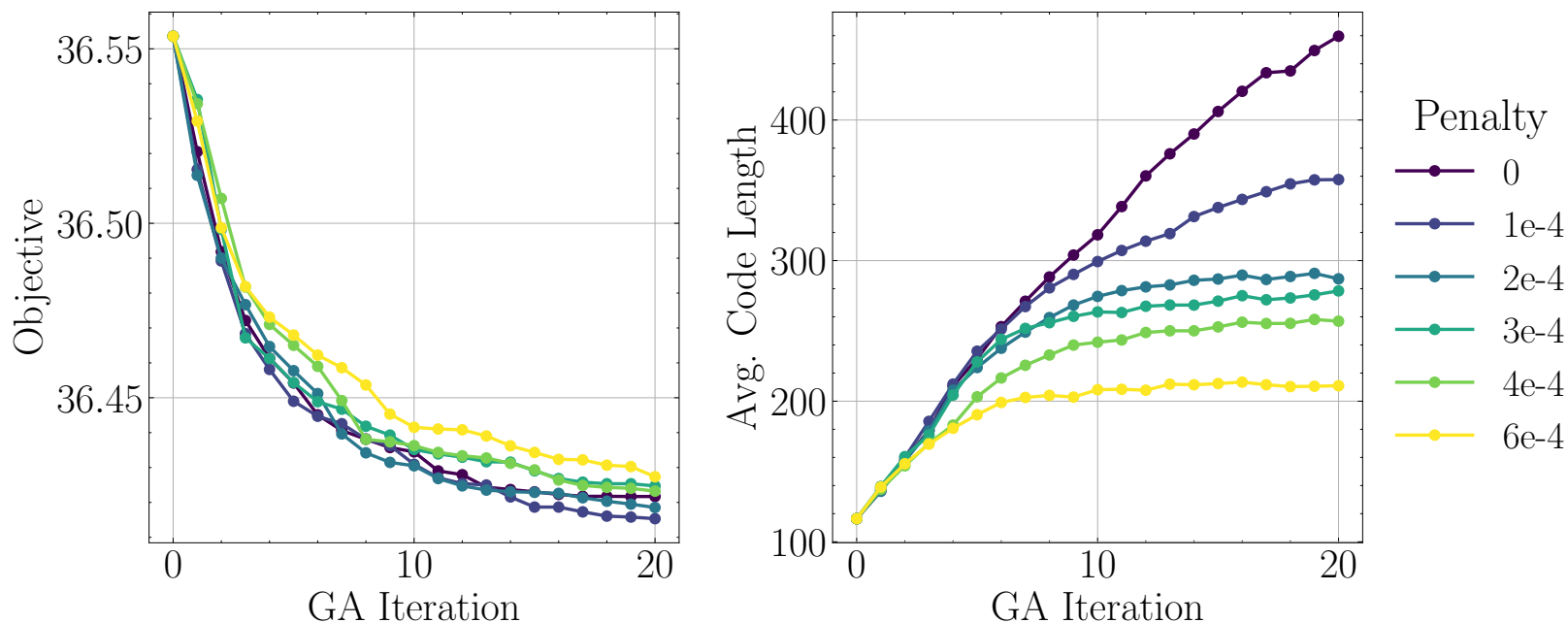
💡 You do not need to use expensive LLMs to get good performance.
Currently showing promise: gemma4, qwen3.8-27b

Ablation Study



💡 The search strategy applied to the LLM code influences performance.

Sensitivity Analysis: Code length Penalty



💡 Penalizing the code length has a negligible impact on performance, but significantly shortens the code.

Is the generated code any good?

We had **three heuristics experts** analyze the best heuristics after asking the LLM to clean up the code.

Criteria	CVRP	VRPTW	PCVRP
Readability	✓	✓	✓
Coherence	✓	✓	✓
Maintainability	✓	✓	~
Interpretability	✓	✓	~
Novelty	~	~	~

- The quality of the code was **surprisingly good**, but there is **room for improvement**.
- The code is **novel** – to our knowledge, no existing code matches these approaches

What about timetabling?

AHS for the IHTC 2024 Competition with Qwen3.8

What about timetabling?

Setup

- **IHTC 2024** public dataset
- Qwen3.8 told to make a heuristic
- Worked overnight

Results

- Feasible solutions: **30 / 30**
- Total cost: **+6.40%** over BKS
- Range: **+0.21%** (m17) to **+14.61%** (m07)

Per-instance gap to the BKS



instance	Qwen Heuristic	BKS	gap
m17	22,589	22,542	+0.21%
m04	3,362	3,345	+0.51%
m20	5,971	5,335	+11.92%
m07	8,400	7,329	+14.61%



Not bad for for the investment.

Conclusion & future work

Summary:

- End-to-end LLM code generation (AlphaEvolve & co.) **fails** to match state-of-the art OR performance
- LLMs need **small, clearly defined tasks** to function correctly
- Matching/beating the OR state of the art with LLMs **is possible!**

Limitations:

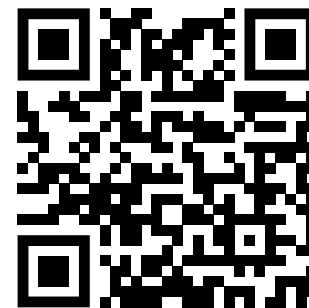
- Unclear how well this will work on novel problems.
- Has anything new actually been created?
- Who cares about these deterministic problems with a few constraints?

Future work:

- LLM generated heuristics for other problems (scheduling, etc.)

Outlook: stronger models (Qwen 3.8, Opus 5, GPT 5.6) lead to better heuristics

LLMs are fundamentally changing the field of OR



Advertisement: EURO PhD School

What: EURO PhD School “Data Science meets Combinatorial Optimization”

When: August 23–27, 2027

Where: University of Vienna, Austria

Topics: DRL, LLMs, Decision Focused Learning, Explainability...

Website coming soon...



© C.Stadler/Bwag; CC-BY-SA-4.0, via Wikimedia Commons

Coming soon: Heurizon



Thank you!

Questions?