

Retrieval is Enough: A Memory-Augmented Agent for Algorithm Selection

Weizhou Zhang Lixing Lyu

Institute of Operations Research and Analytics
National University of Singapore, Singapore
{weizhouz, lyulixing}@u.nus.edu

Abstract

Selecting the best algorithm for each optimization instance is difficult because algorithm performance can vary substantially across instance structures and parameter settings. Existing learned selectors often encode historical experience in fixed model parameters, making new performance records costly to incorporate. We introduce a memory augmented agent for instance wise algorithm selection based on retrieval augmented generation. Given a new instance, the agent retrieves structurally similar historical cases, each containing the complete performance profile of all candidate options rather than only the winning label. It then reasons over the retrieved objective gaps and rankings before invoking an algorithm and configuration. New verified records can be added directly to the external memory without retraining the selector. We evaluate the proposed approach on 10,898 geographically grounded hub location instances, using 8,718 instances for retrieval and 2,180 for testing across five algorithm and configuration options. The proposed method achieves 68.35% selection accuracy and a 0.549% objective gap relative to the virtual best solver, outperforming the single best solver and retrieval only label voting baselines. These results demonstrate the value of combining retrieval with reasoning over complete solver performance records.

1 Introduction

Optimization problems arise whenever a decision must satisfy a set of constraints while optimizing an objective. A single problem class may contain instances with very different sizes, structures, and cost patterns. These differences matter in large integer and combinatorial optimization, where an exact solver can be highly effective on one instance but computationally prohibitive on another, while faster heuristics often trade solution quality for scalability. As a result, no candidate algorithm is ex-

pected to dominate uniformly, and the choice of algorithm becomes part of the solution process.

We study instance wise algorithm selection for parameterized optimization problems. Given a new instance and a fixed collection of candidate algorithms, the goal is to select the algorithm expected to achieve the best empirical performance, together with its configuration when needed. Classical selection methods learn this decision from pairs of instance features and observed solver performance (Rice, 1976; Kerschke et al., 2019). Although effective, a fitted selector usually represents historical experience through fixed model parameters. Incorporating new benchmark records may therefore require another training cycle, and a predicted label need not reveal which prior cases support the decision.

Retrieval augmented generation (RAG) offers a different way to use historical experience (Lewis et al., 2020). Instead of encoding all evidence in model parameters, a system can retrieve relevant records when a decision is made and place them in the context of a large language model (LLM). This is particularly natural for algorithm selection because historical benchmarks contain direct evidence about how candidate algorithms behave on different instances. Yet most LLM research in optimization focuses on translating natural language into formal models or solver calls, while existing LLM approaches to algorithm selection emphasize learned representations of algorithm code. Neither line directly treats the complete performance records of similar solved instances as decision evidence.

We propose a memory augmented agent built on retrieval augmented generation for instance wise algorithm selection. The agent stores historical optimization instances together with the observed performance of every candidate algorithm. For a new instance, it retrieves structurally similar records and constructs a prompt that combines the target

instance, candidate descriptions, retrieved performance profiles, and user instructions. The LLM compares the candidates using this evidence and returns an algorithm, its configuration when applicable. The selected algorithm is then run without any modification to the underlying solver. This design separates selection from solution: existing algorithms retain their original implementations, while the external memory can grow as new verified benchmark records become available.

1.1 Main Contributions

Retrieval Augmented Algorithm Selection. Building on the classical algorithm selection setting, we condition each selection on a natural language instruction in addition to the target instance, candidate algorithms, and historical observations. The framework returns an algorithm and its configuration when applicable, while leaving the underlying algorithms unchanged.

Retrieval without Retraining. New observations can be added to the external memory and used immediately without retraining the selector. For each retrieved instance, we provide the full performance vector over all candidate algorithms, rather than only its winning label. The model can therefore compare performance gaps and rankings across several retrieved instances instead of following the label returned by a one nearest neighbor rule.

Numerical Evaluation. We construct a dataset containing 10,898 hub location instances from Chinese city locations. A random 80/20 split produces 8,718 historical records for retrieval and 2,180 held out instances for evaluation. The proposed method achieves 68.35% selection accuracy, compared with 52.80% for the single best solver and 61.74% for the strongest retrieval only baseline, kNN top 7. Its average objective gap relative to the virtual best solver is 0.549%, compared with 1.298% for kNN top 7.

1.2 Related Work

Algorithm selection and configuration. The classical algorithm selection problem chooses an algorithm for each instance to optimize a target measure (Rice, 1976). Modern work studies selection together with configuration and resource allocation (Kerschke et al., 2019). SATzilla predicts solver performance from instance features (Xu et al., 2008, 2012); ParamILS and ISAC optimize

algorithm parameters (Hutter et al., 2009; Kadioglu et al., 2010); and Hydra and AutoFolio configure selection systems (Xu et al., 2010; Lindauer et al., 2015). Other methods allocate computation time among several algorithms (Kadioglu et al., 2011; Amadini et al., 2014), or learn instance representations in place of manually designed features (Loreggia et al., 2016). ASlib provides standard benchmarks and motivates comparisons with the single best solver and virtual best solver (Bischl et al., 2016). These approaches establish the value of matching algorithms to instances, but most encode the selection rule in a trained predictor or a configured allocation policy. Our framework instead retrieves solver performance records when each selection is made, so new records can be used without fitting the selector again.

Retrieval augmented selection. Case based reasoning (CBR) reuses previously solved instances in optimization. Kraay and Harker (1996) apply CBR to repetitive combinatorial optimization, and later work uses related ideas for constraint solving and satisfiability (O’Mahony et al., 2008; Hurley and O’Sullivan, 2012). RAG supplies an LLM with information retrieved from an external memory (Lewis et al., 2020), and CBR has recently been used as a structured retrieval mechanism for LLM agents (Wiratunga et al., 2024). In a simple retrieval rule, a recommendation may follow the decision attached to the nearest stored case. Our framework instead retrieves several similar instances, and each record contains the observed performance of every candidate rather than only a past solution or winning label. The retrieved cases therefore reveal performance gaps, rankings, and agreement across neighbors. This evidence supports a direct comparison of the candidates before an algorithm and configuration are selected.

Language models for optimization. Chain of Experts, OptiMUS, and ORLM use language models to generate formal optimization models or solver instructions from text (Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Huang et al., 2024). Related work considers natural language assistance and interactive preference elicitation for decision optimization (Wasserkrug et al., 2024; Lawless et al., 2024b). The closest selection work derives algorithm representations from source code and aligns them with problem representations (Wu et al., 2024). Its main objective is to improve how algorithms are represented, rather than to retrieve

historical solver performance for the target instance. Other studies use LLMs to configure cutting plane separators under limited data or learn representations of mixed integer linear programs for downstream prediction (Lawless et al., 2024a; Li et al., 2024). In contrast, we assume that the optimization model and candidate algorithms already exist. We leave them unchanged and use retrieved empirical performance, together with the user prompt, as evidence for choosing an algorithm and its configuration.

2 Framework Formulation

For any given optimization problem class with a collection of candidate algorithms, the proposed framework can be instantiated as an instance-wise algorithm selector. Let $\mathcal{C}$ denote an optimization problem class, and let $\Theta_{\mathcal{C}}$ be the instance space associated with $\mathcal{C}$. For each instance $\theta \in \Theta_{\mathcal{C}}$, we consider an optimization problem of the form

$$\min_{x \in \mathcal{X}_{\mathcal{C}}(\theta)} f_{\mathcal{C}}(x; \theta), \quad (1)$$

where $f_{\mathcal{C}}(x; \theta)$ is the objective function defined by problem class $\mathcal{C}$ and instance θ , and $\mathcal{X}_{\mathcal{C}}(\theta)$ is the corresponding feasible region for the decision variable x .

For problem class $\mathcal{C}$, suppose that we are given a collection of practical solution methods,

$$\mathcal{A}^{\mathcal{C}} = \{A_1^{\mathcal{C}}, A_2^{\mathcal{C}}, \dots, A_K^{\mathcal{C}}\}.$$

These methods may include exact algorithms, approximation algorithms, heuristics, local search procedures, decomposition-based methods, or domain-specific solvers. Given an instance θ , each method $A_k^{\mathcal{C}}$ can be applied to produce a feasible or candidate solution for (1).

In addition to the candidate algorithms, we assume access to a collection of historical optimization records for problem class $\mathcal{C}$,

$$\mathcal{H}_{\mathcal{C}} = \{(\theta_i, \mathbf{y}_i)\}_{i=1}^{|\mathcal{H}_{\mathcal{C}}|},$$

where $\theta_i \in \Theta_{\mathcal{C}}$ denotes a previously encountered problem instance and

$$\mathbf{y}_i = (y_{i1}, \dots, y_{iK})$$

is its complete performance vector, where $y_{ik} = f_{\mathcal{C}}(A_k^{\mathcal{C}}(\theta_i); \theta_i)$ records the objective value achieved by candidate k . The corresponding oracle index is $o_i = \arg \min_{k \in [K]} y_{ik}$. The precise way in which

these historical records are retrieved and used is described in the next section.

The goal of the meta-algorithm is to select a suitable algorithm for a given instance. Formally, given a new instance $\theta \in \Theta_{\mathcal{C}}$, the candidate algorithm set $\mathcal{A}^{\mathcal{C}}$, the historical records $\mathcal{H}_{\mathcal{C}}$, and a user prompt or task-specific instruction ρ , the meta-algorithm outputs

$$\hat{k} = \pi_{\mathcal{C}}(\theta, \mathcal{A}^{\mathcal{C}}, \mathcal{H}_{\mathcal{C}}, \rho),$$

where $\hat{k} \in [K]$ is the selected algorithm index. Equivalently, the selected algorithm is $\hat{A} = A_{\hat{k}}^{\mathcal{C}}$. The final solution is obtained by running the selected algorithm: $\hat{x} = A_{\hat{k}}^{\mathcal{C}}(\theta)$.

Therefore, the same framework can be instantiated for different problem classes by specifying the corresponding instance space $\Theta_{\mathcal{C}}$, feasible region $\mathcal{X}_{\mathcal{C}}(\theta)$, objective function $f_{\mathcal{C}}(\cdot; \theta)$, candidate algorithm set $\mathcal{A}^{\mathcal{C}}$, and historical records $\mathcal{H}_{\mathcal{C}}$.

3 Memory Augmented Agent for Algorithm Selection

Instance Representation. For each problem class, we map a raw optimization instance into a fixed-length feature vector that captures its scale, structure, cost patterns, and key factors that may affect solver performance. We then apply instance-wise deterministic normalization, avoiding the need to fit a scaler on the historical instance database. This makes the representation of each instance independent of the database used for retrieval.

Offline Memory Construction. We construct an external solver-performance memory from historical benchmark instances. Each memory entry stores the normalized instance representation together with the observed performance profile of candidate algorithms, including the best-performing solver, performance gaps, and relative rankings. This memory provides verified empirical evidence of past solver behavior and serves as the basis for retrieval-augmented algorithm selection.

Online Memory-Augmented Retrieval. Given a new instance, we retrieve similar memory entries using nearest-neighbor search in the normalized feature space and assign each entry a similarity score. The retrieved entries serve as few-shot empirical evidence for the LLM, allowing it to reason from past solver behavior on structurally similar instances.

Few-Shot Prompting. We construct a structured prompt that combines problem context, candidate algorithm descriptions, target-instance features, and retrieved memory entries. Unlike a nearest-neighbor rule that directly follows the top retrieved label, each memory entry provides the full relative performance profile of all candidate algorithms. This allows the LLM to assess whether the retrieved entries consistently support one solver or reveal an ambiguous decision boundary, and then output a recommendation supported by the retrieved evidence.

Agent Workflow. At decision time, the agent observes a new instance, retrieves relevant performance records, reasons over the retrieved evidence, and invokes the selected algorithm: The algorithm returns the solution $\hat{x}$. Because each memory entry contains the complete performance vector $\mathbf{y}_i$, a new entry is added only after all candidate options have been benchmarked and their results have been verified: This asynchronous memory update does not require retraining the selector.

4 Numerical Experiment

4.1 Facility Location Problem

We illustrate the proposed framework using a hub location problem as an example. Let $\mathcal{C}$ denote the class of hub location instances. Each instance $\theta \in \Theta_{\mathcal{C}}$ is represented by

$$\theta = (n, p, (d_{ij})_{i,j \in [n]}, c_1, c_{2,1}, c_{2,2}, v_1, v_2),$$

where n is the number of nodes, p is the number of hubs, d_{ij} is the distance between nodes i and j , c_1 is the hub construction cost, $c_{2,1}$ and $c_{2,2}$ are link maintenance costs, and v_1 and v_2 are the communication speeds for hub–hub and hub–non-hub links, respectively. The notation used in the general framework and the hub location example is summarized in Table 2 at the beginning of the appendix.

Figure 1 illustrates the hub / non-hub structure: hub nodes are fully connected to each other, each non-hub node attaches to exactly one hub, and any communication between two non-hub nodes is forwarded through their hubs.

Optimization Model. The decision variables specify hub selection, node-to-hub assignment, and routing decisions. Let h_i indicate whether node i is selected as a hub, q_{ik} indicate whether node i is assigned to hub k , and $x_{ik\ell j}$ indicate whether the

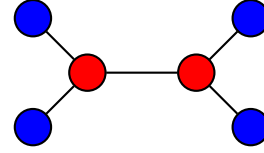


Figure 1: Hub location network (6 nodes with 2 hubs): red points are hub nodes and blue points are non-hub nodes. Any pair of non-hub nodes can only communicate through their hub nodes.

communication between nodes i and j is routed through hubs k and ℓ . For a given instance θ , the hub location problem is formulated as

$$\begin{aligned} \min \quad & \frac{1}{n(n-1)} \sum_{\substack{i,j \in V \\ i \neq j}} \sum_{k,\ell \in V} x_{ik\ell j} \\ & \times \left(\frac{d_{ik} + d_{j\ell}}{v_2} + \frac{d_{k\ell}}{v_1} \right) \\ \text{s.t.} \quad & \sum_{k \in V} h_k = p, \end{aligned} \tag{H1}$$

$$q_{ik} \leq h_k, \quad \forall i, k \in V, \tag{H2}$$

$$\sum_{k \in V} q_{ik} = 1, \quad \forall i \in V, \tag{H3}$$

$$x_{ik\ell j} \leq \frac{1}{2}(q_{ik} + q_{j\ell}), \quad \forall i \neq j, k, \ell \in V, \tag{H4}$$

$$\sum_{k,\ell \in V} x_{ik\ell j} = 1, \quad \forall i \neq j, \tag{H5}$$

$$h_i, q_{ik}, x_{ik\ell j} \in \{0, 1\}. \tag{H6}$$

Here, the objective minimizes the average communication time between all node pairs. The feasible region defined by the above constraints corresponds to $\mathcal{X}_{\mathcal{C}}(\theta)$ in the general framework. Constraint (H1) selects exactly p hubs. Constraints (H2)–(H3) require that each node is assigned to exactly one selected hub. Constraints (H4)–(H5) require that each node pair (i, j) is routed through exactly one hub pair (k, ℓ) consistent with its assignments. Constraint (H6) imposes binary restrictions on all decision variables.

Reference Formulation and Candidate Options.

The full MILP provides a reference formulation of the hub location problem but is not included in the experimental selection pool because its four index routing variables become computationally expensive as n grows. Its complete formulation is given in Appendix A.2. The selector instead chooses among five algorithm and configuration

options:

$$\begin{aligned} \mathcal{A}^C &= \{A_{\text{cluster}}^C\} \\ &\cup \{A_{\text{prop},\alpha}^C : \alpha \in S\}, \\ S &= \{0.6, 0.7, 0.8, 0.9\}. \end{aligned}$$

At the system level, instances with $n < 10$ are routed directly to the full MILP and are not included in the algorithm selection dataset. Every instance with $10 \leq n \leq 30$ is handled by the five option selector; no size based shortcut is applied within this range.

Cluster Presolve Method. The cluster option, A_{cluster}^C , first partitions the network into p clusters, for example using k -means, and then selects one hub within each cluster by solving a reduced one-hub subproblem. This method exploits the spatial structure of the instance and is more scalable than the full MILP formulation. The clustering and hub selection models are given in Appendix A.3.

Proportion Decision Method. Each proportion option, $A_{\text{prop},\alpha}^C$, approximates the relative contribution of hub–hub and hub–non-hub links by a parameter $\alpha \in [0, 1]$, and then solves a reduced formulation without the full four-index routing variables. This method is computationally efficient, but its performance depends on the choice of α . Its reduced model is given in Appendix A.4. All five candidate options are evaluated using the common objective defined in Appendix A.

4.2 Prompt Engineering

Our prompt follows a system-user separation scheme. The system prompt is composed of five parts.

1. **Role:** we specify the agent as an algorithm selector specialized in the facility location problem.
2. **Task:** it instructs the agent to recommend the best option from the candidate pool given the input problem features and historical cases.
3. **Algorithm Pool:** it enumerates the candidate options, each with a description of the instance characteristics under which it tends to perform well.
4. **Reasoning Guideline:** it specifies rules for the thought process, guiding the agent to first inspect and analyze the input query features,

then compare them against the retrieved similar cases, and finally make a selection based on this reasoning.

5. **Output Format:** it requires the agent to return one option from the candidate pool in a predefined format.

On the other hand, the user prompt is constructed for each input instance. First, it provides the extracted input feature set, shown in both raw dimensional form and normalized form. Second, it supplies the top-7 retrieved cases obtained through FAISS similarity search. Each case includes its instance features, similarity score, objective gap for every candidate option, complete ranking, and empirically best option. Finally, the agent reasons over the input features and the retrieved performance profiles jointly, and then outputs a recommended option.

4.3 Results

4.3.1 Data Generation

The underlying data consist of a city file containing coordinates and integer city indices and a precomputed haversine distance matrix indexed by the same identifiers. Sampling is restricted to eleven province level regions: Jiangsu, Fujian, Guangdong, Hubei, Zhejiang, Jiangxi, Hunan, Yunnan, Guangxi, Shanghai, and Anhui.

For each $n \in \{10, \dots, 30\}$, the generator first randomly selects a region containing at least n cities and then samples n cities without replacement from that region. Consequently, every instance represents a within region distribution network. Duplicate city combinations are removed during generation. The remaining parameters satisfy

$$\begin{aligned} p &\in \{2, 3, 4, 5\}, \\ v_1 &\in \{1.5, 2.0, 2.5, 3.0, 4.0\}, \\ v_2 &= 1.0. \end{aligned}$$

Using random seed 42, this procedure produces 10,898 instances. We apply a random 80/20 split with random_state=42, yielding 8,718 historical records in $\mathcal{H}_C$ and a held out test set of $N = 2,180$. Only the historical split is included in the FAISS retrieval index; the split is by instance rather than by held out region.

For every instance, we run all five algorithm and configuration options to obtain the complete performance vector $\mathbf{y}_i$. Its oracle label is $o_i = \arg \min_{k \in [K]} y_{ik}$, and the relative objective gaps

and ranking are derived from the same vector. The MILP subproblems are implemented in PuLP and solved with CBC. The held out performance vectors are used only to determine oracle labels and evaluation metrics and are never included in the retrieval memory.

4.3.2 Numerical Setup

Setup. We instantiate the framework of Section 2 as follows: each instance is encoded by a 32-dimensional feature vector normalized instance-wise with fixed clip/tanh bounds (no database statistics); retrieval uses a FAISS flat $L2$ index returning the top-7 neighbors, each with its full performance profile over all options; and the LLM backbone is DeepSeek-Chat at temperature 0.2, prompted as in Section 4.2. The selection target is one of five (algorithm, configuration) options: CLUSTER_PRESOLVE and PROPORTION_SOLVE(α) for $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$. The option with the smallest objective on an instance is its oracle label (Section 4.3.1).

Baselines. On the same held out test set ($N=2,180$) we compare **RAG** against two families of baselines:

- **Single Option:** ALWAYS-CLUSTER and ALWAYS-PROP.(α), plus a RANDOM lower bound.
- **Retrieval Only:** KNN TOP-1 and KNN TOP-7 apply majority voting to the oracle labels of the same retrieved neighbors. Unlike RAG, these baselines do not use the complete performance profiles. A top-30 sensitivity result is reported in Table 3 in the appendix.

Metrics. We report two metrics, averaged over the test set:

- **Accuracy (%)**: fraction of instances whose selected option matches the oracle label.
- **Objective Gap (%)**: relative gap of the selected option to the best objective,

$$f_C^*(\theta) = \min_{k \in [K]} f_C(A_k^C(\theta); \theta),$$

$$\text{gap}(\theta) = 100 \frac{f_C(\hat{A}(\theta); \theta) - f_C^*(\theta)}{f_C^*(\theta)} \%.$$

4.3.3 Main Results

Table 1 reports the overall accuracy and objective gap on the held out test set, and Figure 2 shows

accuracy stratified by instance size. The proposed **RAG** agent performs best on both metrics: it attains the highest selection accuracy (68.35%) and the smallest objective gap (+0.549%), improving over the best single-option policy (ALWAYS-CLUSTER, 52.80%) by more than 15 accuracy points. RAG also outperforms the retrieval-only baselines built on the same neighbors (61.74% for KNN TOP-7 and 59.91% for KNN TOP-1), raising accuracy by about 6.6 points and more than halving the objective gap relative to KNN TOP-7. This comparison measures the combined benefit of exposing the complete performance profiles and using the LLM to synthesize them, rather than the effect of the LLM alone. Figure 2 shows that RAG leads at 16 of the 21 tested instance sizes; it is not the best at four sizes and ties at $n = 30$.

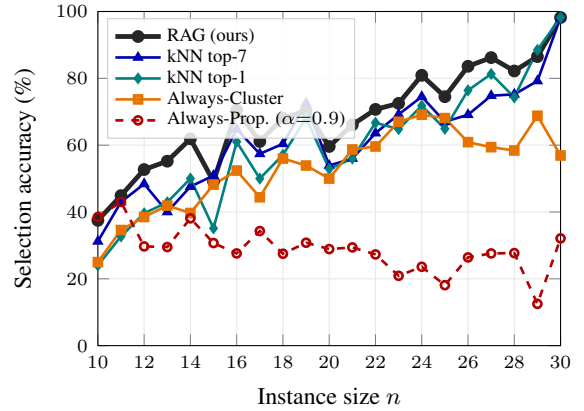


Figure 2: Accuracy vs Instance Size n

Table 1: Overall accuracy and objective value gap on the held out test set ($N=2,180$).

Method	Accuracy (%)	Obj. Gap (%)
Always-Cluster	52.80	+1.792
Always-Prop. ($\alpha=0.6$)	3.49	+51.352
Always-Prop. ($\alpha=0.7$)	6.88	+38.837
Always-Prop. ($\alpha=0.8$)	7.84	+22.787
Always-Prop. ($\alpha=0.9$)	28.99	+7.588
Random	20.55	+24.707
kNN top-1 (majority vote)	59.91	+2.265
kNN top-7 (majority vote)	61.74	+1.298
RAG	68.35	+0.549

Limitations

We evaluate on a single problem class with a portfolio of five closely related options, and our random 80/20 split shares regions between memory and test set, which favors nearest-neighbour retrieval; a held-out-region split is left to future work. Our

baselines also do not isolate the source of the gain, since we compare against neither a non-LLM aggregator over the same retrieved profiles nor a conventional supervised selector on the same features. We further report point estimates from one split under stochastic decoding, and do not measure the latency or cost of the selection step.

References

- Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *International Conference on Machine Learning (ICML)*.
- Roberto Amadini, Maurizio Gabbriellini, and Jacopo Mauro. 2014. SUNNY: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, 14(4–5):509–524.
- Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchet, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. 2016. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58.
- Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2024. ORLM: A customizable framework in training large models for automated optimization modeling. *arXiv preprint arXiv:2405.17743*.
- Barry Hurley and Barry O’Sullivan. 2012. Adaptation in a CBR-based solver portfolio for the satisfiability problem. In *International Conference on Case-Based Reasoning (ICCBR)*, pages 152–166.
- Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stütze. 2009. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.
- Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. 2011. Algorithm selection and scheduling. In *Principles and Practice of Constraint Programming (CP)*, pages 454–469.
- Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC – instance-specific algorithm configuration. In *European Conference on Artificial Intelligence (ECAI)*, pages 751–756.
- Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45.
- David R. Kraay and Patrick T. Harker. 1996. Case-based reasoning for repetitive combinatorial optimization problems, part I: Framework. *Journal of Heuristics*, 2(1):55–85.
- Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. 2024a. LLMs for cold-start cutting plane separator configuration. *arXiv preprint arXiv:2412.12038*.
- Connor Lawless, Jakob Schoeffler, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahar Sarrafzadeh. 2024b. “I want it that way”: Enabling interactive decision support using large language models and constraint programming. *ACM Transactions on Interactive Intelligent Systems*.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. 2024. Towards foundation models for mixed integer linear programming. *arXiv preprint arXiv:2410.08288*.
- Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. 2015. AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778.
- Andrea Loreggia, Yuri Malitsky, Horst Samulowitz, and Vijay Saraswat. 2016. Deep learning for algorithm portfolios. In *AAAI Conference on Artificial Intelligence*.
- Eoin O’Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O’Sullivan. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science (AICS)*.
- John R. Rice. 1976. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier.
- Segev Wasserkrug, Leonard Boussiou, Dick den Hertog, Farzaneh Mirzazadeh, Ilker Birbil, Jannis Kurtz, and Donato Maragno. 2024. From large language models and optimization to decision optimization copilot: A research manifesto. *arXiv preprint arXiv:2402.16269*.
- Nirmalie Wiratunga, Ramitha Abeyratne, Lasal Jayawardena, Kyle Martin, Stewart Massie, Ikechukwu Nkisi-Orji, Ruvan Weerasinghe, Anne Liret, and Bruno Fleisch. 2024. CBR-RAG: Case-based reasoning for retrieval augmented generation in LLMs for legal question answering. In *International Conference on Case-Based Reasoning (ICCBR)*.
- Xingyu Wu, Yan Zhong, Jibin Wu, Bingbing Jiang, and Kay Chen Tan. 2024. Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

- Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. 2024. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations (ICLR)*.
- Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI Conference on Artificial Intelligence*, pages 210–216.
- Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.
- Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 228–241.

A Detailed Formulations of the Reference and Candidate Methods

This appendix specifies the full MILP reference formulation and the two types of methods used to construct the five candidate options. Let $V = [n]$ be the node set, let d_{ij} be the distance between nodes i and j , and let p be the required number of hubs. To avoid confusion between the scalar p and a binary decision variable, we write

$$h_i = \begin{cases} 1, & \text{if } i \text{ is a selected hub,} \\ 0, & \text{otherwise,} \end{cases}$$

and

$$q_{ik} = \begin{cases} 1, & \text{if } i \text{ is assigned to hub } k, \\ 0, & \text{otherwise.} \end{cases}$$

The same variables are used in the main text. Every method returns a hub set and an assignment map $a : V \rightarrow V$, where $a(i)$ is the hub assigned to node i .

A.1 Common Evaluation Objective

All returned solutions are evaluated by the average communication time

$$T(a) = \frac{1}{n(n-1)} \sum_{i \neq j} \left(\frac{d_{i,a(i)}}{v_2} + \frac{d_{a(i),a(j)}}{v_1} + \frac{d_{j,a(j)}}{v_2} \right). \quad (2)$$

This is the objective used to determine the best candidate option and to compute the objective gap in the experiments. Because p is fixed for an instance, the network cost

$$C(p) = pc_1 + \frac{p(p-1)}{2}c_{2,1} + (n-p)c_{2,2} \quad (3)$$

is identical across the methods and does not affect their ranking.

A.2 Full MILP Reference Formulation

The direct method introduces a binary routing variable

$$x_{iklj} = \mathbb{I}\{a(i) = k, a(j) = \ell\}$$

for every ordered pair $i \neq j$ and every pair of candidate hubs $k, \ell \in V$. It solves

$$\min_{h,q,x} \quad \frac{1}{n(n-1)} \sum_{i \neq j} \sum_{k, \ell \in V} x_{iklj} \left(\frac{d_{ik}}{v_2} + \frac{d_{k\ell}}{v_1} + \frac{d_{j\ell}}{v_2} \right) \quad (4)$$

$$\text{s.t.} \quad \sum_{k \in V} h_k = p, \quad (5)$$

$$q_{ik} \leq h_k, \quad \forall i, k \in V, \quad (6)$$

$$\sum_{k \in V} q_{ik} = 1, \quad \forall i \in V, \quad (7)$$

$$x_{iklj} \leq \frac{q_{ik} + q_{j\ell}}{2}, \quad \forall i \neq j, k, \ell \in V, \quad (8)$$

$$\sum_{k, \ell \in V} x_{iklj} = 1, \quad \forall i \neq j, \quad (9)$$

$$h_k, q_{ik}, x_{iklj} \in \{0, 1\}. \quad (10)$$

Constraints (5)–(7) select exactly p hubs and assign every node to one selected hub. Constraints (8)–(9) activate exactly the routing variable associated with the two selected assignments. The model contains $O(n^4)$ routing variables and constraints.

A.3 Cluster Presolve Method

Let $s_i \in \mathbb{R}^2$ denote the geographic coordinates of node i . The first stage partitions the nodes into p sets $G_1, \dots, G_p$ using K means. The associated clustering objective is

$$\min_{\mu, u} \quad \sum_{i \in V} \sum_{r=1}^p u_{ir} \|s_i - \mu_r\|_2^2 \quad (11)$$

$$\text{s.t.} \quad \sum_{r=1}^p u_{ir} = 1, \quad \forall i \in V, \quad (12)$$

$$u_{ir} \in \{0, 1\}, \quad \forall i \in V, r \in [p]. \quad (13)$$

Here, $u_{ir} = 1$ means that node i belongs to cluster G_r , and μ_r is its centroid. After the partition is obtained, the method chooses one hub inside each cluster. For cluster G_r , define a binary variable w_{rk} for each $k \in G_r$. The one hub problem is

$$\min_w \quad \sum_{k \in G_r} \left(\sum_{i \in G_r} d_{ik} \right) w_{rk} \quad (14)$$

$$\text{s.t.} \quad \sum_{k \in G_r} w_{rk} = 1, \quad (15)$$

$$w_{rk} \in \{0, 1\}, \quad \forall k \in G_r. \quad (16)$$

The selected hub is therefore the medoid

$$k_r^* = \arg \min_{k \in G_r} \sum_{i \in G_r} d_{ik}. \quad (17)$$

Every node $i \in G_r$ is assigned to k_r^* . This objective is equivalent, up to the constant factor $2(|G_r| - 1)$, to summing the access distance over all ordered pairs within the cluster. Intercluster communication is not optimized when the medoids are selected,

Table 2: Notation used in the general framework and hub location example.

Symbol	Definition	Symbol	Definition
$\mathcal{C}$	problem class	$\Theta_{\mathcal{C}}$	instance space for class $\mathcal{C}$
θ	problem instance	$\mathcal{X}_{\mathcal{C}}(\theta)$	feasible region of instance θ
$f_{\mathcal{C}}(\cdot; \theta)$	objective function for instance θ	ρ	user prompt or task-specific instruction
$\mathcal{A}^{\mathcal{C}}$	set of K candidate algorithms or configured options	$A_k^{\mathcal{C}}$	candidate option indexed by $k \in [K]$
$\mathcal{H}_{\mathcal{C}}$	historical instance and performance records	$\pi_{\mathcal{C}}$	meta-algorithm for class $\mathcal{C}$
$\mathbf{y}_i$	complete performance vector for instance i	o_i	oracle candidate index for instance i
$\hat{k}, \hat{A}$	selected index and algorithm	$\hat{x}$	solution returned by the selected algorithm
K	number of candidate options	N	number of held out test instances
$V = [n]$	node set	p	required number of hubs
d_{ij}	distance between nodes i and j	$c_1, c_{2,1}, c_{2,2}$	hub and link cost parameters
v_1, v_2	hub–hub and hub–non-hub communication speeds	h_i	indicator that node i is selected as a hub
q_{ik}	indicator that node i is assigned to hub k	$x_{ik\ell j}$	indicator that pair (i, j) is routed through hubs (k, ℓ)
$a(i)$	hub assigned to node i	$T(a)$	common evaluation objective
s_i	coordinates of node i	G_r, μ_r	cluster r and its centroid
u_{ir}	indicator that node i belongs to cluster r	w_{rk}	indicator that node k is selected in cluster r
k_r^*	selected medoid of cluster r	α	weight in the proportion decision surrogate
z_{ij}	indicator that both nodes i and j are hubs	$C(p)$	network cost for a fixed value of p

but it is included when the final solution is evaluated using (2). After clustering, hub selection requires only the within cluster distance sums and has total size $O(n^2)$.

A.4 Proportion Decision Method

The proportion method replaces the four index routing variables with a surrogate objective containing two distance components. The first component measures access from each node to its assigned hub. The second measures the distances between selected hubs. Let

$$z_{ij} = h_i h_j,$$

where $z_{ij} = 1$ if both i and j are selected as hubs. For a fixed $\alpha \in [0, 1]$, the implemented model is

$$\min_{h, q, z} \quad \alpha \sum_{i, k \in V} d_{ik} q_{ik} + (1 - \alpha) \sum_{i, j \in V} d_{ij} z_{ij} \quad (18)$$

$$\text{s.t.} \quad \sum_{k \in V} h_k = p, \quad (19)$$

$$q_{ik} \leq h_k, \quad \forall i, k \in V, \quad (20)$$

$$\sum_{k \in V} q_{ik} = 1, \quad \forall i \in V, \quad (21)$$

$$z_{ij} \geq h_i + h_j - 1, \quad \forall i, j \in V, \quad (22)$$

$$z_{ij} \leq \frac{h_i + h_j}{2}, \quad \forall i, j \in V, \quad (23)$$

$$h_i, q_{ik}, z_{ij} \in \{0, 1\}. \quad (24)$$

Constraints (22)–(23) are the binary linearization of $z_{ij} = h_i h_j$. The first sum in (18) is the total

spoke distance, while the second is the total distance over ordered pairs of selected hubs. Thus, α is a weight that controls the relative importance of the two surrogate components. It is not optimized inside the model; each tested value of α defines a separate algorithm and configuration option. The model has $O(n^2)$ variables and constraints. After it is solved, its assignments are evaluated using the original communication time in (2), rather than the surrogate objective in (18).

B Facility Location Test Result by Sample Size

Table 3: Algorithm selection accuracy (%) by instance size n on the held out test set ($N=2,180$).

Panel A: $n = 10$ to 20											
Method	10	11	12	13	14	15	16	17	18	19	20
Always-Cluster	25.0	34.6	38.5	41.9	39.7	48.2	52.4	44.4	56.0	53.9	50.0
Always-Prop. ($\alpha=0.6$)	5.2	1.9	4.4	1.9	2.4	3.5	3.8	5.6	6.6	4.4	5.8
Always-Prop. ($\alpha=0.7$)	13.5	10.3	12.1	14.3	8.7	10.5	6.7	6.5	2.2	3.3	2.9
Always-Prop. ($\alpha=0.8$)	17.7	10.3	15.4	12.4	11.1	7.0	9.5	9.3	7.7	7.7	12.5
Always-Prop. ($\alpha=0.9$)	38.5	43.0	29.7	29.5	38.1	30.7	27.6	34.3	27.5	30.8	28.9
Random	18.8	20.6	14.3	26.7	18.2	20.2	13.3	18.5	23.1	24.2	31.7
kNN top-1	24.0	32.7	39.6	42.9	50.0	35.1	61.0	50.0	57.1	68.1	52.9
kNN top-7	31.2	43.0	48.4	40.0	47.6	50.9	64.8	57.4	60.4	72.5	53.9
kNN top-30	33.3	43.9	37.4	45.7	44.4	51.8	59.0	51.9	67.0	64.8	52.9
RAG	37.5	44.9	52.7	55.2	61.9	49.1	70.5	61.1	68.1	70.3	59.6

Panel B: $n = 21$ to 30											
Method	21	22	23	24	25	26	27	28	29	30	
Always-Cluster	58.7	59.6	67.0	69.1	68.1	60.9	59.4	58.4	68.8	56.9	
Always-Prop. ($\alpha=0.6$)	3.7	2.0	2.2	2.7	0.0	4.5	4.1	4.0	1.0	3.7	
Always-Prop. ($\alpha=0.7$)	3.7	4.0	7.7	0.9	4.3	4.5	5.7	5.9	9.4	7.3	
Always-Prop. ($\alpha=0.8$)	4.6	7.1	2.2	3.6	9.6	3.6	3.2	4.0	8.3	0.0	
Always-Prop. ($\alpha=0.9$)	29.4	27.3	20.9	23.6	18.1	26.4	27.6	27.7	12.5	32.1	
Random	19.3	20.2	20.9	25.4	21.3	18.2	21.9	20.8	17.7	16.5	
kNN top-1	56.0	66.7	64.8	71.8	64.9	76.4	81.3	74.3	88.5	98.2	
kNN top-7	56.0	63.6	69.2	74.5	67.0	69.1	74.8	75.2	79.2	98.2	
kNN top-30	61.5	59.6	68.1	74.5	68.1	73.6	73.2	72.3	76.0	98.2	
RAG	66.1	70.7	72.5	80.9	74.5	83.6	86.2	82.2	86.5	98.2	