WikiEvolve: Coupling Foundational and Experiential Knowledge in Self-Evolving Agents

Anonymous authors
Paper under double-blind review

Abstract

Self-evolving agents can improve solutions for various tasks by iteratively proposing them based on past experience (e.g., proposed algorithms and their utilities), evaluating their utilities, and using evolutionary search to select the best solution. However, as these agents might only make minor modifications and do finite exploration, useful solutions can remain undiscovered, limiting the effectiveness of experience-based evolution. To address this limitation, we introduce WikiEvolve, a novel framework that couples a fixed *foundational wiki* with an accumulating *experiential memory*. The foundational wiki provides established foundational knowledge from literature to enrich candidate solutions, while the experiential memory records past experience to guide future selection. Across distributed constraint optimization (DCOP) and Bayesian optimization (BO) tasks, our framework usually improves the final solution utility over experience-based evolution. Moreover, our experiments show how the amount of foundational knowledge influences the final solutions. These results demonstrate that combining a structured foundational wiki with an experiential memory enables self-evolving agents to break out of trial-and-error traps and discover solutions beyond what pure experience can reach.

1 Introduction

By pretraining on massive corpora, large language models (LLMs) acquire broad knowledge and capabilities, but their parameters typically remain fixed after deployment, limiting their continual adaptation to new tasks and environments. This limitation is consequential in open-ended tasks that require iterative improvement, such as algorithm discovery and code optimization (Novikov et al., 2025) or lifelong embodied control (Wang et al., 2024). It has motivated *self-evolving agents*, which keep the LLM fixed but let the agent learn from its own *experiences* (Fang et al., 2025). An experience in *experiential memory* (Hu et al., 2025) records what the agent tried and what happened. For example, Reflexion stores verbal reflections on failed attempts and uses them in later trials (Shinn et al., 2023), and FunSearch stores evaluated programs and their scores, and prompts the LLM with high-scoring ones to propose new programs (Romera-Paredes et al., 2024).

Relying primarily on past experiences, however, limits what such agents can discover, especially in knowledge-intensive domains (Trehan & Chopra, 2026). The experiential memory carries no information about solutions or modifications that the agent has not explored. Moreover, feedback is often limited to a scalar utility or success signal, which is too sparse to reveal which part of a modification caused the observed outcome (Che et al., 2026). The agent may consequently over-exploit existing candidates, make ineffective modifications, or repeat unproductive experiments. Human researchers, in contrast, do not only rely on their own trial and error. High-impact science typically builds on well-established prior work while combining it in new ways (Uzzi et al., 2013). Literature itself encodes knowledge that anticipates future discoveries (Tshitoyan et al., 2019). This motivates our central question:

Can self-evolving agents similarly benefit from foundational knowledge, i.e., the established algorithms, principles, and theory documented in the literature?

In principle, LLM agents already draw on foundational knowledge through their pretrained parameters and reasoning (Gekhman et al., 2026), or external documents retrieved by using retrieval-augmented generation (RAG) (Lewis et al., 2020). However, these methods do not explicitly relate pieces of knowledge to one another or to the algorithm being improved. At every iteration, the agent must

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

therefore work out again which and how knowledge relates to its current solution, guided only by sparse feedback. Retrieving raw documents by surface similarity alone forces the agent to reread the same sources at every query. Sec. 5 provides a deeper comparison with related work. We therefore propose to give the agent a *foundational wiki*: a fixed set of interlinked pages, written before self-evolution begins, that distill papers and textbooks of the target domain into summaries of individual methods and overviews of related methods (Karpathy, 2026). This raises our research question: *How should a foundational wiki be structured, and how should an agent combine it with its experiential memory to guide self-evolution?*

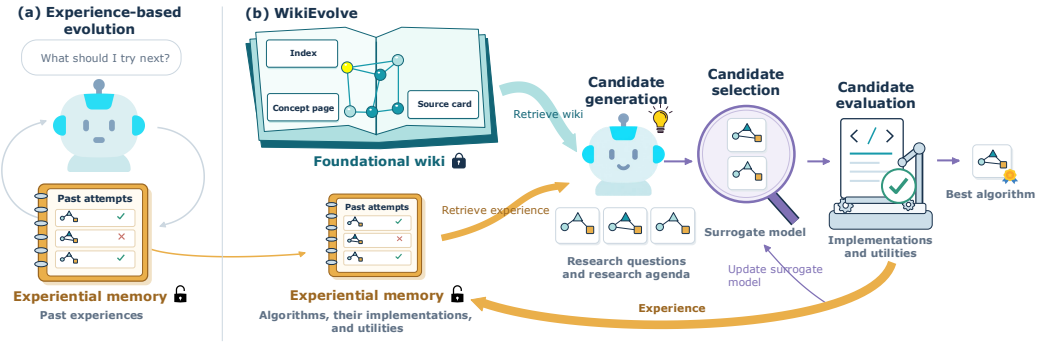


Figure 1: Overview of WikiEvolve. (a) Experience-based evolution proposes new algorithms only from its experiential memory of past attempts. (b) WikiEvolve consults both a foundational wiki (containing index, source cards and concept pages) that stays fixed during self-evolution and its accumulating experiential memory. In each iteration, the agent retrieves relevant wiki and past experiences, updates its research agenda and research questions, and generates candidate algorithms. A surrogate model selects the promising candidates for implementation and evaluation, and their implementations and utilities update the experiential memory and the surrogate model. Finally, WikiEvolve returns the best-found algorithm over all iterations.

To this end, we introduce WikiEvolve, a framework that couples a fixed foundational wiki with an evolving experiential memory (Fig. 1). The two sources are complementary: the wiki suggests *what* to try beyond the agent's own experience, while the experiential memory records *how well* these ideas performed once implemented and evaluated, guiding what to try next. We address three successive challenges that occur in every self-evolution iteration:

(Ch1) Research planning. Scalar environment feedback alone does not indicate which direction to pursue next. Inspired by how researchers work, the WikiEvolve agent first reviews relevant experiences, updates a research agenda, and poses specific research questions.

(Ch2) Finding relevant knowledge. Given these questions, the WikiEvolve agent navigates the wiki much as a person browses an encyclopedia, from pages that match its questions to linked pages on related methods. Each proposed algorithm cites the pages it builds on, so its experience records which knowledge it used.

(Ch3) Expensive implementation and evaluation. Implementing and evaluating every idea is expensive, and LLM agents often fail to implement complex ideas faithfully (Tang et al., 2025; Kon et al., 2026). Hence, the WikiEvolve agent proposes small changes to one part of its current best algorithm rather than rewriting it (Romera-Paredes et al., 2024; Nam et al., 2025), and learns from its experiences to predict and implement promising candidates.

In summary, our contributions are as follows. **(C1)** We identify a key limitation of experience-based self-evolving agents and propose a new perspective: agents should also learn from foundational knowledge in the literature, through a structured foundational wiki (Sec. 2, Sec. 3.1). **(C2–C4)** We introduce WikiEvolve, which addresses challenges Ch1–Ch3 through **(C2)** research planning informed by experiential memory (Sec. 3.2), **(C3)** retrieval that traverses the graph-structured wiki in addition to matching by similarity (Sec. 3.2), and **(C4)** surrogate-guided selection of candidate algorithms before implementation (Sec. 3.3). **(C5)** We evaluate WikiEvolve on five algorithm-improvement tasks from two domains, distributed constraint optimization (DCOP) (Fioretto et al., 2018) and Bayesian optimization (BO) (Chew et al., 2026), with two LLMs (Sec. 4). Compared

with experience-based self-evolving agents, such as OpenEvolve (Sharma, 2025), WIKIEVOLVE finds the best algorithm on all DCOP tasks, and on all BO tasks when the algorithms are subsequently evaluated on a larger deployment budget. The amount of knowledge matters, however: on Graph Coloring, retrieving a moderate part of the wiki nearly *doubles* the improvement obtained without the wiki (21.93% vs. 11.62%), whereas providing the entire wiki performs worse than using no wiki at all (6.76%).

2 PRELIMINARIES

Problem setup. We consider a self-evolving agent that iteratively improves an algorithm for a given task, such as graph coloring, where the goal is to color nodes such that no two adjacent nodes share the same color. The agent is initialized with an algorithm a', represented as a textual description of a solution strategy, e.g., "use backtracking on ordered nodes, assign a random feasible color, and undo the previous assignment when a constraint is violated." Each algorithm a is implemented as executable code c_a, whose performance is evaluated on a set of task instances to obtain the utility u_a.

Experience-based evolution. At each iteration t, the self-evolving agent generates a pool of candidate algorithms A_t, executes their corresponding implementations, and evaluates their utilities. The resulting experiences of iteration t, i.e., candidate algorithms, their implementations, and utilities, $\mathcal{E}_t = \{(a, \mathsf{c}_a, u_a) : a \in A_t\}$, are stored to an evolving experiential memory $\mathcal{E}$. In iteration $t+1$, the agent uses the current $\mathcal{E} = \bigcup_{t'=1}^{t} \mathcal{E}_{t'}$ to guide the generation of A_{t+1}.

For example, it may generate each new candidate by making an incremental modification to the highest-utility algorithm so far. After T iterations, the agent returns the highest-utility algorithm, $a^\star \in \arg\max_{a \in \bigcup_{t=1}^{T} A_t} u_a$. We call this process evolution because, as in evolutionary search, high-utility algorithms are selected and modified to produce new candidates (Romera-Paredes et al., 2024; Liu et al., 2024a).

Foundational knowledge. In addition to experiential knowledge accumulated during self-evolution, the agent can draw on foundational knowledge, containing established algorithms, principles for algorithm selection, and theoretical results documented in the literature on the target domain. For the graph coloring example, this includes vertex-ordering heuristics, constraint-propagation techniques, and pruning rules for backtracking search, etc. Although this knowledge may be available in external documents or encoded in the parameters of the underlying LLMs through pretraining, identifying connections among relevant ideas and determining how they relate to the current algorithm can remain challenging, as discussed in Sec. 1.

3 WIKIEVOLVE

Ideally, a self-evolving agent should return an algorithm $a^\star$ with higher utility. To do so, we propose coupling experiential memory $\mathcal{E}$ and foundational knowledge to generate candidate algorithms, such as A_t in iteration t, based on both previously explored solutions and unexplored foundational knowledge. WIKIEVOLVE involves defining a structured foundational wiki to support efficient retrieval and traversal (Sec. 3.1), planning and retrieval (Sec. 3.2), and steps to generate and efficiently evaluate candidate algorithms (Sec. 3.3). The framework is summarized in Fig. 1 and Alg. 1.

3.1 FOUNDATIONAL WIKI

The foundational wiki $\mathcal{W}$ represents foundational knowledge as an undirected graph $\mathcal{W} = (N, E)$ with node set N and edge set E that consolidates information from research papers, tutorials, and other external resources. A visualized example is given in Fig. 3. Each node in N corresponds to a textual Markdown file in the wiki. The nodes $N = S \cup C \cup \{n_{\text{idx}}\}$ comprises the source card set S, synthesized concept page set C, and an index node n_{idx} for high-level navigation. Each source card in S contains textual knowledge automatically extracted from an individual source, such as a single research paper or tutorial, according to a predefined schema. Each concept page in C synthesizes knowledge across source cards and covers a topic such as (a) an algorithm family (e.g., multi-fidelity Bayesian optimization) or (b) a comparison criterion (e.g., theoretical guarantees of BO methods). The edge set E is induced by internal Markdown hyperlinks present in the wiki files. The index node

n_{idx} links to the concept pages in C. Each concept page in C links to source cards it synthesizes and related concept pages. A source card in S may link to related source cards (e.g., similar methods). Further details on wiki construction are provided in App. A. In Alg. 1, we denote $C_t \subseteq C$ and $S_t \subseteq S$ as the subsets of concept pages and source cards retrieved in iteration t, respectively.

3.2 Planning and Retrieval

This section specifies how the self-evolving agent should use $\mathcal{E}$ and the wiki $\mathcal{W}$ based on the current *research agenda* and *research questions*. A research agenda provides a high-level description of the research direction, including the hypothesis, problem definition, research objectives, and research plan. Research questions identify specific issues that need to be addressed to advance the agenda. Concrete examples of the agenda and research questions are given in App. H. Intuitively, WikiEvolve mirrors the research process of first identifying a promising direction (agenda) and specific questions before committing to a concrete solution. This process enables more targeted knowledge retrieval, so that the agent is not limited to local algorithm modifications.

In iteration t, the agent retrieves from $\mathcal{E}$ the most lexically similar experiences E_{t-1} to the previous agenda G_{t-1} (see App. B.1). Then during *research planning*, the agent inputs these experiences E_{t-1}, agenda G_{t-1}, and the current best algorithm $a^\star$ with its utility $u_{a^\star}$ to the LLM, which will output the new research agenda G_t and research questions Q_t.

Based on the research questions Q_t, the agent retrieves the top-R most lexically similar nodes in the set $\{n_{\text{idx}}\} \cup C$ and the top-x source cards with the highest relevance. The relevance is higher when the source card's content is more lexically similar or has a shorter path to the top-R most lexically similar concept pages. Subsequently, the recently retrieved source cards are penalized to reduce repeated use of the same source cards, and encourage exploration of the wiki. More details are provided in App. B.2. Then the agent retrieves from $\mathcal{E}$ the most lexically similar experiences E_t to the new agenda G_t (see App. B.1).

This planning and retriever design is motivated by prior work on feedback-driven reflection (Shinn et al., 2023) and experience reuse (Zhao et al., 2024) for improving subsequent agent decisions. In Sec. 4.5, we present an ablation study comparing WikiEvolve with variants that remove the research agenda G_t and the research questions Q_t, to assess their importance. In Sec. 4.3, we show that using the entire wiki instead of target retrieval can substantially degrade performance.

Remark on experience. Each experience records the algorithm a, code $\mathtt{c}_a$ and the utility u_a. The algorithm also records (i.e., cites) the source cards used for its generation to directly connect relevant foundational knowledge to experiences. This design help to agent to navigate the wiki efficiently and reassess earlier research agenda.

3.3 Candidate Algorithm and Selection

Candidate generation. Next, in each iteration t, the agent uses the LLM to propose a pool of candidate algorithms A_t conditioned on the research agenda G_t, research questions Q_t, the parent algorithm $a^\star$ (initialized as a' in the first iteration), the retrieved foundational wiki knowledge and experiences E_t. Rather than substantially redesigning the parent algorithm $a^\star$, the agent adapts mechanisms from its cited foundational knowledge and experience to modify a specific part while preserving the overall algorithmic structure in each $a \in A_t$. This local refinement strategy follows evidence that focusing evolution on critical components can reduce unnecessary reconstruction and enable deeper, more reliable search than coarse whole-program modification (Romera-Paredes et al., 2024; Nam et al., 2025).

Candidate selection for evaluation. The self-evolving agent requires the utility u_a of each candidate algorithm a to update the best algorithm $a^\star$ and its experiential memory $\mathcal{E}$. How can u_a be known?

It is unreliable to rely on the same LLM to judge the utility of a without implementation as research agents often fail to faithfully implement proposed methods (Kon et al., 2026), while LLM judges can exhibit self-preference when evaluating their own outputs (Pombal et al., 2026). Thus, the actual implementation $\mathtt{c}_a$ should be evaluated. Tang et al. (2025) report difficulties in correctly implementing complex, self-generated ideas, while EXP-Bench identifies missing essential implementation components as a major failure mode (Kon et al., 2026). These suggest that candidate

Algorithm 1 WIKIEVOLVE: Knowledge-Guided Self-Evolution

Require: Initial algorithm a', Foundational wiki $\mathcal{W}$, Initial research agenda G_0, Utility function u, Surrogate Model φ
Ensure: Best algorithm $a^\star$

Current best algorithm $a^\star \leftarrow a'$; Experiential memory $\mathcal{E} \leftarrow \emptyset$; $\mathcal{E}_0 \leftarrow \emptyset$;
for iteration $t = 1, \ldots, T$ **do**
  $\mathcal{E}_t \leftarrow \emptyset$;
  $\mathrm{E}_{t-1} \leftarrow \textsc{RetrieveExperience}(a^\star, \mathcal{E}_{t-1}, \mathcal{E}, \mathrm{G}_{t-1})$ {App. B.1}
  (Research Agenda G_t, Questions Q_t) $\leftarrow \textsc{ResearchPlanning}(a^\star, \mathrm{E}_{t-1}, \mathrm{G}_{t-1})$ {LLM}
  $(C_t, S_t) \leftarrow \textsc{Retrievewiki}(\mathrm{Q}_t, \mathcal{W})$ {App. B.2}
  $\mathrm{E}_t \leftarrow \textsc{RetrieveExperience}(a^\star, \mathcal{E}_{t-1}, \mathcal{E}, \mathrm{G}_t)$ {App. B.1}
  Candidate algorithms $A_t \leftarrow \textsc{GenerateCandidates}(a^\star, \mathrm{E}_t, \mathrm{G}_t, \mathrm{Q}_t, C_t, S_t)$ {LLM}
  $A_t^{\mathrm{sub}} \leftarrow \textsc{SelectCandidates}(A_t, \varphi)$ {App. D}
  for $a \in A_t^{\mathrm{sub}}$ **do**
    $\mathtt{c}_a \leftarrow \textsc{Implement}(a, S_t, \mathtt{c}_{a^\star}, \mathrm{E}_t)$ {LLM}
    $\mathcal{E}_t \leftarrow \mathcal{E}_t \cup \{(a, \mathtt{c}_a, u_a)\}$
  end for
  $\varphi \leftarrow \textsc{UpdatePredictionModel}\left(\{(a, u_a) : a \in A_t^{\mathrm{sub}}\}\right)$ {App. C}
  $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}_t$
  $a^\star \leftarrow \arg\max_{a \in A_t^{\mathrm{sub}} \cup \{a^\star\}} u_a$
end for

selection can be used to reduce the cost of implementations and evaluations and improve success rates, for example, by leaving out algorithms whose previous implementations have resulted in low utility.

WIKIEVOLVE proposes to learn a surrogate model φ to predict the expected utility of each candidate before implementation and select a promising subset. Specifically, the surrogate model maps each candidate algorithm (represented by its embedding produced by a fixed text encoder) to the utility measured after implementation. The model is trained on previously evaluated candidates and their measured utilities (see App. C). We use Gaussian Process as the surrogate model as it is flexible, provides uncertainty estimates, and performs sufficiently well in Sec. 4. This component can be replaced by other surrogate models that operate on candidate representations and provide both utility predictions and uncertainty estimates for candidate selection.

We select the promising candidates $A_t^{\mathrm{sub}} \subseteq A_t$ for implementation based on three features: contextual relevance to the research questions Q_t, expected improvement in utility, and uncertainty in the utility estimates produced by the surrogate model φ. Each feature independently selects several candidates, and their union forms the final candidate set A_t^{sub}. App. D describes how this selection balances contextual relevance, exploitation of candidates with higher predicted utility with exploration of candidates with uncertain utility by leveraging acquisition functions from Bayesian optimization. We provide an additional study on the effect of the surrogate model in App. E.4.

Candidate evaluation and experience accumulation. The agent implements each candidate algorithm a in A_t^{sub} while referencing the retrieved source cards S_t from $\mathcal{W}$, retrieved experiences E_t and implementation of the parent algorithm $\mathtt{c}_{a^\star}$. The resulting utility u_a is used to update the experiential memory $\mathcal{E}$ and surrogate model φ to improve the proposed algorithm and utility prediction in future iterations.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We evaluate Qwen3.8-27B (Qwen Team, 2026) in non-thinking mode and DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026) with low reasoning effort. We use temperature 0 by default and 0.7 for candidate generation.

Tasks. We evaluate WIKIEVOLVE on five tasks from two domains. In each task, the utility function is given and the agent may only modify the algorithm. We consider tasks from distributed constraint

optimization (DCOP) and Bayesian optimization (BO) as they represent challenging NP-hard and black-box optimization problems, respectively, while providing automated evaluation for candidate algorithms. For DCOP, we use Graph Coloring, Max-DisCSP, and finite-cost Weighted DCOP (Zhang et al., 2005; Yokoo & Hirayama, 2000; Fioretto et al., 2018), each with 18 frozen instances spanning six structural groups (e.g., graph density and constraint tightness). Graph Coloring and Weighted DCOP start from DSA-B (Zhang et al., 2005), and Max-DisCSP from MGM (Maheswaran et al., 2004) in FRODO (Léauté et al., 2009). For BO, we use two BoLT tasks (Chew et al., 2026): Hyperparameter Optimization (HPO), which tunes LoRA (Hu et al., 2022) fine-tuning of Qwen3-8B-Base (Yang et al., 2025) over three instances, and Data Mixture (DM), which searches a 6-dimensional training-data mixture for Qwen3-4B-Base over a single instance. Both tasks are initialized with the GP-EI (Jones et al., 1998) algorithm from BoTorch (Balandat et al., 2020). Further task and utility details are provided in App. E.1–App. E.3.

Utility definition. The utility u_a of an algorithm measures how much it reduces the cost of the initial algorithm, averaged over the task instances and over repeated stochastic runs of the algorithm. The cost is the final constraint cost in DCOP and the simple regret after the self-evolution budget of 60 BO evaluations. We report the improvement (%) of the best-found algorithm as its cost reduction relative to the cost of the initial algorithm. The initial algorithm a' has $u_{a'} = 0$, and a positive utility means a lower cost. For BO, following BoLT, the improvement is the reduction in geometric-mean simple regret, reported both at the self-evolution budget and, to test generalization, at a larger budget of 200 BO evaluations.

Configuration and baselines. All results are reported as mean $\pm$ std over three independent self-evolution runs under different seeds. WikiEvolve runs $T = 50$ iterations with up to $k = 5$ implemented candidates each (App. D). We compare WikiEvolve against **(A)** methods without experiential memory $\mathcal{E}$ and **(B)** experience-based self-evolving agents without foundational knowledge. **(A) Single-Iteration (No / Given wiki)** runs 50 independent one-step modifications of a', without or with wiki retrieval; we report their mean (Pass@1) and the best utility (Best-of-50). **(B) Experience Only** runs WikiEvolve without the foundational wiki; **ReAct + Partial History** (Yao et al., 2023) edits, evaluates, and restores versions of the algorithm, remembering its last eight interactions; and **OpenEvolve** (Sharma, 2025) runs AlphaEvolve-style evolutionary search, whose program database serves as experiential memory.

4.2 Main Results

DCOP. In Tab. 1, WikiEvolve achieves the highest mean improvement across all methods on both models and all tasks. WikiEvolve outperforms Experience Only, OpenEvolve and ReAct with partial history, showing that foundational knowledge provides further gains beyond experience-only self-evolution. Foundational knowledge alone is also insufficient. Without experience, single-iteration generation given wiki does not consistently improve over generation with no wiki, and even its Best-of-50 improvement remains below WikiEvolve across all settings.

BO. In each BO task, the BO algorithm's utility is evaluated after 60 objective evaluations. In practice, a user may wish to use the algorithm on more evaluations (e.g., 200) but use a smaller budget for tuning. Hence, Tab. 1 reports the mean improvement over GP-EI under both the tuning and larger deployment budget.

Improving GP-EI requires experiences. A single LLM-proposed modification yields small or inconsistent gains (Pass@1), and at $B = 60$, even the Best-of-50 improvement falls short of Experience Only and WikiEvolve. ReAct with partial history and OpenEvolve also struggle; on HPO, they rarely improve on GP-EI, and on DM, they fall short of Experience Only and WikiEvolve.

Experience only performs sufficiently well for the tuning budget. At $B = 60$, WikiEvolve and Experience Only perform similarly, each winning on two of the four task–LLM pairs, mostly by margins within run-to-run noise. The wiki's benefit here is reliability rather than peak performance: WikiEvolve tends to vary less across runs in three of the four pairs.

WikiEvolve generalizes better than Experience Only to the larger budget. At $B = 200$, WikiEvolve is best across all four task–LLM pairs. The contrast is sharpest on DM: at $B = 200$, every baseline is *worse* than GP-EI, including Experience Only, whose algorithms improve on GP-EI by 70.63% and 80.03% at $B = 60$, sugggesting that its gains came from overfitting to the tuning budget.

Table 1: Improvement (%) of the best-found algorithm $a^\star$ over the initial algorithm, as the mean $\pm$ standard deviation over three self-evolution runs; higher value is better. For BO, (**@60**) denotes the tuning budget $B = 60$, and (**@200**) evaluates generalization to a larger deployment budget $B = 200$. For BO, the mean and standard deviation are computed on log simple regret (App. E.1).

		DCOP			**BO (@60 BO eval)**		**BO (@200 BO eval)**	
Model	Method	Graph Coloring	Max-DisCSP	Weighted DCOP	HPO	DM	HPO	DM
Qwen3.8-27B	*Single-Iteration (No wiki)*							
	Pass@1	$0.10_{\pm 0.42}$	$0.63_{\pm 0.73}$	$0.31_{\pm 0.41}$	$1.84_{\pm 0.61}$	$3.05_{\pm 1.46}$	$0.48_{\pm 1.12}$	$-5.33_{\pm 5.52}$
	Best-of-50	$5.07_{\pm 2.96}$	$11.99_{\pm 3.02}$	$3.36_{\pm 0.98}$	$34.70_{\pm 7.51}$	$50.95_{\pm 11.19}$	$10.22_{\pm 36.29}$	$-0.44_{\pm 15.70}$
	Single-Iteration (Given wiki)							
	Pass@1	$0.21_{\pm 0.43}$	$0.71_{\pm 0.83}$	$0.27_{\pm 0.35}$	$0.58_{\pm 0.55}$	$4.47_{\pm 0.91}$	$-0.16_{\pm 0.21}$	$-5.82_{\pm 4.39}$
	Best-of-50	$10.43_{\pm 3.01}$	$15.05_{\pm 3.32}$	$3.20_{\pm 0.90}$	$22.29_{\pm 21.65}$	$57.81_{\pm 3.18}$	$-6.31_{\pm 12.23}$	$-48.85_{\pm 48.19}$
	Experience Only	$11.62_{\pm 1.41}$	$22.83_{\pm 1.02}$	$5.68_{\pm 0.95}$	$\mathbf{79.13_{\pm 9.39}}$	$70.63_{\pm 13.98}$	$49.71_{\pm 11.01}$	$-67.98_{\pm 41.05}$
	ReAct + Partial History	$12.61_{\pm 3.60}$	$12.16_{\pm 2.84}$	$4.50_{\pm 1.15}$	$2.81_{\pm 4.80}$	$55.68_{\pm 15.89}$	$13.55_{\pm 21.80}$	$-14.75_{\pm 29.27}$
	OpenEvolve	$0.00_{\pm 0.00}$	$18.68_{\pm 1.14}$	$5.06_{\pm 1.47}$	$0.00_{\pm 0.00}$	$38.78_{\pm 15.58}$	$0.00_{\pm 0.00}$	$-65.64_{\pm 113.73}$
	WikiEvolve (ours)	$\mathbf{18.01_{\pm 3.02}}$	$\mathbf{26.74_{\pm 1.82}}$	$\mathbf{6.79_{\pm 2.17}}$	$70.19_{\pm 3.51}$	$\mathbf{71.57_{\pm 7.18}}$	$\mathbf{66.62_{\pm 12.09}}$	$\mathbf{2.88_{\pm 61.75}}$
DeepSeek-V4-Flash	*Single-Iteration (No wiki)*							
	Pass@1	$0.00_{\pm 0.00}$	$7.43_{\pm 1.48}$	$0.37_{\pm 0.22}$	$7.73_{\pm 0.48}$	$16.81_{\pm 4.33}$	$5.61_{\pm 4.69}$	$-26.17_{\pm 24.96}$
	Best-of-50	$0.00_{\pm 0.00}$	$26.44_{\pm 2.02}$	$3.31_{\pm 0.84}$	$49.13_{\pm 6.16}$	$62.79_{\pm 13.55}$	$34.30_{\pm 15.76}$	$-50.92_{\pm 80.57}$
	Single-Iteration (Given wiki)							
	Pass@1	$0.16_{\pm 0.28}$	$3.72_{\pm 0.51}$	$0.33_{\pm 0.20}$	$6.40_{\pm 3.85}$	$5.75_{\pm 2.94}$	$5.80_{\pm 2.51}$	$-13.54_{\pm 7.85}$
	Best-of-50	$8.15_{\pm 14.12}$	$22.95_{\pm 2.15}$	$3.85_{\pm 1.30}$	$51.70_{\pm 26.81}$	$47.39_{\pm 11.68}$	$24.94_{\pm 17.41}$	$-55.50_{\pm 28.24}$
	Experience Only	$14.18_{\pm 12.31}$	$27.87_{\pm 3.97}$	$4.32_{\pm 1.24}$	$65.84_{\pm 2.47}$	$\mathbf{80.03_{\pm 13.44}}$	$44.35_{\pm 24.69}$	$-51.35_{\pm 23.05}$
	ReAct + Partial History	$4.85_{\pm 8.40}$	$22.68_{\pm 2.23}$	$1.69_{\pm 0.42}$	$0.00_{\pm 0.00}$	$43.44_{\pm 33.60}$	$0.00_{\pm 0.00}$	$-30.56_{\pm 38.11}$
	OpenEvolve	$14.55_{\pm 12.60}$	$18.88_{\pm 2.40}$	$4.06_{\pm 0.83}$	$0.00_{\pm 0.00}$	$27.43_{\pm 21.48}$	$0.00_{\pm 0.00}$	$-21.74_{\pm 34.14}$
	WikiEvolve (ours)	$\mathbf{22.52_{\pm 6.25}}$	$\mathbf{28.75_{\pm 2.11}}$	$\mathbf{9.49_{\pm 1.99}}$	$\mathbf{77.92_{\pm 7.21}}$	$78.64_{\pm 6.75}$	$\mathbf{56.57_{\pm 15.54}}$	$\mathbf{21.17_{\pm 36.52}}$

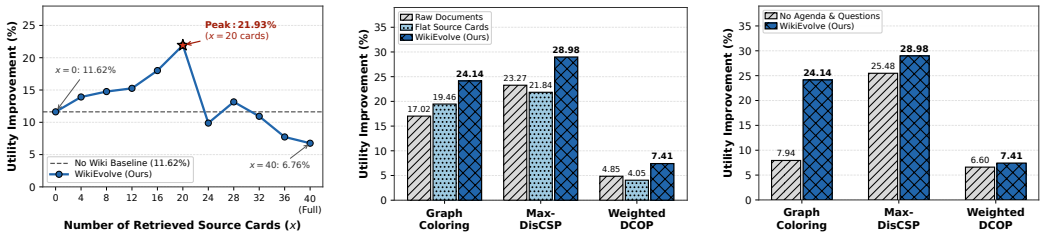
WikiEvolve is the only method that stays at or above GP-EI with both LLMs, albeit with high variance. On HPO, WikiEvolve's gain also holds up: with Qwen it drops by only 3.57 points from $B = 60$ to $B = 200$, whereas that of Experience Only drops by 29.42. Intuitively, trial and error in experience-based self-evolution favors algorithms that specifically works for the tuning budget, while algorithms inspired by BO literature were designed to work in general and thus can better generalize to larger deployment budget. Tab. 8 in App. G qualitatively illustrates this intuition. These results show that neither foundational knowledge nor experiential memory alone accounts for the gains of WikiEvolve. Instead, the improvement comes from coupling both: the agent explores retrieved foundational knowledge, evaluates the resulting candidates, and accumulates their outcomes as experiential memory to guide subsequent self-evolution.

In App. E.5, we examine the utility improvement over self-evolution iterations and show that WikiEvolve continues to discover better algorithms after the baseline methods have largely plateaued. Appendices F and G provide concrete examples of how the best-found algorithm use foundational knowledge.

4.3 The Foundational-Knowledge Goldilocks Zone

We study how the amount of retrieved foundational knowledge affects self-evolution by varying the number of retrieved source cards x, from no retrieval ($x = 0$) to the entire wiki ($x = 40$). Fig. 2a reports the mean utility improvement over the initial algorithm across three seeds for the Graph Coloring task with Qwen3.8-27B model in 50 iterations. The results show a non-monotonic relationship between the amount of retrieved foundational knowledge and performance. Without wiki retrieval ($x = 0$), the utility improvement is 11.62%. Performance gradually increases as more source cards are retrieved and reaches the highest observed value of 21.93% at $x = 20$. However, further increasing the retrieved knowledge substantially degrades performance, reaching 6.76% when the entire wiki ($x = 40$) is provided.

These results exhibit a foundational knowledge *Goldilocks zone*: an intermediate amount of retrieved foundational knowledge performs better than either no knowledge or providing the entire wiki. A small knowledge set may restrict access to useful mechanisms, whereas an overly broad set may introduce competing information and disperse exploration. Thus, more accessible foundational knowledge does not necessarily lead to better self-evolution.



(a) Goldilocks Zone (b) Knowledge Representation (c) Agenda Guidance

Figure 2: Empirical ablations across DCOP tasks. (a) Retrieved card count $x \in [0, 40]$ exhibits an intermediate Goldilocks zone at $x = 20$ (Sec. 4.3). (b) Knowledge representation comparison across 3 columns per task: Raw Documents, Flat Source Cards, and structured WikiEvolve (Sec. 4.4). (c) Research planning ablation: full pipeline vs. removing research agenda and questions (Sec. 4.5).

4.4 Knowledge Representation and wiki Structure

We examine two related questions: Are compressed source cards as useful as raw documents? Do the structure and connections in our foundational wiki improve utility? In this experiment, Raw Documents replaces each source card with its corresponding raw paper while Flat Source Cards places all source cards in an unstructured pool without connections between them. We report the mean utility improvement of best candidate $a^\star$ over the initial algorithm after 90 iterations averaged across three seeds.

Source cards retain useful knowledge. In Fig. 2b, Flat Source Cards outperforms Raw Documents on Graph Coloring, while Raw Documents only performs slightly better on Max-DisCSP and Weighted DCOP. Thus, source cards relatively preserve useful knowledge for algorithm discovery.

The structured wiki outperforms flat source-card retrieval. In Fig. 2b, WikiEvolve achieves higher utility than Flat Source Cards across all three tasks, with gains ranging from 3.36 to 7.14 percentage points. This suggests that effective self-evolution depends not only on access to the foundational knowledge, but also on how that knowledge is organized and combined with accumulated experimental experience during self-evolution.

4.5 Ablation for Research Agenda and Questions

We ablate the hierarchical research planning stage by removing both the research agenda G_t and research questions Q_t in the 90 self-evolution iterations, while keeping the remaining framework unchanged. This removes the explicit high-level research direction and the targeted questions used to guide wiki retrieval.

As shown in Fig. 2c, removing the agenda and questions reduces utility across all three DCOP tasks, with the largest degradation observed on Graph Coloring. The full WikiEvolve consistently outperforms the ablated variant.

These results suggest that separating high-level research planning from candidate generation is useful for self-evolution. The research agenda maintains a coherent direction across iterations, while the research questions provide more focused retrieval targets. Without these intermediate planning steps, the search is less effectively guided by an explicit research objective and targeted knowledge retrieval.

5 Related Work

Experience-based self-evolving agents. Reflexion stores verbal reflections on failed attempts and conditions later trials on them (Shinn et al., 2023); ExpeL extracts natural-language insights from successful and failed trajectories and recalls past successes as examples (Zhao et al., 2024); Agent Workflow Memory induces reusable workflows from past trajectories (Wang et al., 2025b); ReasoningBank distills reasoning strategies from both successes and failures (Ouyang et al., 2026); and Voyager accumulates a library of executable skills (Wang et al., 2024). The Darwin Gödel

Machine goes further and modifies its own code, tools, and workflows, validating each variant on coding benchmarks (Zhang et al., 2026). These works only learn from the agent's own trajectories and experiences, and propose solutions that concentrate around previously explored solutions. In contrast, WIKIEVOLVE can additionally learn from established foundational knowledge and propose different solutions based on them.

LLM-based algorithm discovery. FunSearch pairs an LLM with an automatic evaluator and evolves algorithms sampled from a database of high-scoring ones, discovering new mathematical constructions and bin-packing heuristics (Romera-Paredes et al., 2024). Evolution of Heuristics evolves natural-language descriptions of algorithms together with their code for online bin packing problems (Liu et al., 2024a); AlphaEvolve scales this loop to larger code bases and scientific and systems-optimization problems (Novikov et al., 2025), with OpenEvolve as an open-source implementation (Sharma, 2025); and AFlow applies search to agentic workflows represented as code (Zhang et al., 2025). These methods propose new programs from previously evaluated ones and the LLM's parametric knowledge. They do not consult established foundational knowledge and may implement and evaluate every candidate. In contrast, WIKIEVOLVE explicitly references relevant foundational knowledge in each candidate and prioritizes promising proposals for implementation.

Foundational knowledge and structured retrieval. Retrieval-augmented generation retrieves raw documents by surface similarity (Lewis et al., 2020), which may be unsuitable for algorithm and code-based foundational knowledge (Wang et al., 2025a) as it does not capture how algorithms relate (e.g., whether the prerequisites match). LLM-maintained wikis distill documents into interlinked pages (Karpathy, 2026; Ming et al., 2026; Singh et al., 2026) but are not optimized to aid knowledge retrieval when solving different tasks. These methods do not consider how to couple foundational knowledge with experiential memory. Conversely, WikiSkill compiles an agent's execution traces into a wiki of failure patterns and strategies (Tang et al., 2026). However, it is based on the agent's experiences instead of foundational knowledge as in WIKIEVOLVE.

Algorithm selection. Selecting algorithm is a classic problem (Rice, 1976) of predicting the best member of a fixed portfolio from hand-crafted (Xu et al., 2008) or learned features (Loreggia et al., 2016), and LLMs have recently been used to extract high-dimensional algorithm representations for selection (Wu et al., 2024). Bayesian optimization has also been applied over learned representations of structured objects (Tripp et al., 2020; Maus et al., 2022), over embeddings of instructions (Chen et al., 2024; Lin et al., 2024), and with LLMs in the loop (Liu et al., 2024b). LLM-based program evolution, in contrast, typically evaluates every generated program (Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024). WIKIEVOLVE tackles this by embedding the textual descriptions of candidate algorithms with a fixed text encoder and using a surrogate model to select promising candidates to implement first.

6 CONCLUSION

This paper raises the perspective that self-evolving agents, like human researchers, can benefit from established foundational knowledge instead of solely relying on trial and error. We introduced WIKIEVOLVE, which couples a fixed foundational wiki with an evolving experiential memory. In each iteration, the agent plans its research with an agenda and questions, retrieves knowledge by traversing the wiki graph, generates candidates, and implements only the promising ones. Our empirical results point to three interesting lessons. First, foundational knowledge is more useful when coupled with learning from experience. Second, targeted retrieval of structured knowledge based on research agenda and questions may outperform retrieval of the entire wiki with more knowledge. Third, algorithms designed with foundational knowledge can better generalize beyond the conditions they were tuned on: in BO tasks, algorithms proposed by WIKIEVOLVE still perform better under a different evaluation budget.

Limitations and future work. As our wikis are built once per task and remain fixed, they do not absorb what the agent discovers. Verifying discovered algorithms and promoting them into the wiki is a natural next step. It remains an open problem to extend WIKIEVOLVE (e.g., improve the wiki design) to other domains with extensive literature such as reinforcement learning.

AI use statement

Generative AI tools were used to assist with drafting, editing, code review, and mathematical consistency checks. All AI-assisted text, code changes, and derivations were reviewed by the authors, who take responsibility for the final content and claims.

Ethics Statement

This paper proposes methods for large language models and does not involve human subjects, collection of personal or sensitive information, or the introduction of new datasets containing such information. Our experiments use existing publicly available benchmarks and pretrained language models.

Reproducibility Statement

The details of our experimental setup are provided in Sec. 4 and the App. E. Additionally, the code used in our experiments has been included in the supplementary material, enabling full reproduction of the results reported in this paper.

References

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Proc. NeurIPS*, pp. 21524–21538, 2020.

Lirong Che, Yuzhe Yang, Peiwen Lin, Chuang Wang, Xueqian Wang, and Jian Su. DemoEvolve: Overcoming sparse feedback in agentic harness evolution with demonstrations. *arXiv:2605.24539*, 2026.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. InstructZero: Efficient instruction optimization for black-box large language models. In *Proc. ICML*, pp. 6503–6518, 2024.

Ruth Wan Theng Chew, Zhiliang Chen, Apivich Hemachandra, and Bryan Kian Hsiang Low. BoLT: A benchmark to democratize black-box optimization research for expensive LLM tasks. In *ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning*, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Proc. NeurIPS*, 2019.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv:2508.07407*, 2025.

Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018.

Zorik Gekhman, Roee Aharoni, Eran Ofek, Mor Geva, Roi Reichart, and Jonathan Herzig. Thinking to recall: How reasoning unlocks parametric knowledge in LLMs. In *Proc. COLM*, 2026.

Daisuke Hatano and Yuichi Yoshida. Distributed multiplicative weights methods for DCOP. In *Proc. AAAI*, pp. 2074–2080, 2015. doi: 10.1609/aaai.v29i1.9425.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proc. NeurIPS Datasets and Benchmarks Track*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proc. ICLR*, 2022.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of AI agents. *arXiv:2512.13564*, 2025.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Andrej Karpathy. LLM Wiki. GitHub Gist, 2026. URL `https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f`.

Patrick Tser Jern Kon, Qiuyi Ding, Jiachen Liu, Xinyi Zhu, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, Matei Zaharia, and Ang Chen. EXP-Bench: Can AI conduct AI research experiments? In *Proc. ICLR*, 2026.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Proc. COLM*, 2025.

Thomas Léauté, Brammert Ottens, and Radoslaw Szymanek. FRODO 2.0: An open-source framework for distributed constraint optimization. In Katsutoshi Hirayama, William Yeoh, and Roie Zivan (eds.), *Proc. IJCAI Distributed Constraint Reasoning Workshop*, pp. 160–164, 2009.

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. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. NeurIPS*, pp. 9459–9474, 2020.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *Proc. ICLR*, 2024.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Use your INSTINCT: INSTruction optimization for LLMs usIng neural bandits coupled with transformers. In *Proc. ICML*, pp. 30317–30345, 2024.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proc. ICML*, 2024a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Proc. NeurIPS*, pp. 21558–21572, 2023.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance Bayesian optimization. In *Proc. ICLR*, 2024b.

Andrea Loreggia, Yuri Malitsky, Horst Samulowitz, and Vijay Saraswat. Deep learning for algorithm portfolios. In *Proc. AAAI*, 2016. doi: 10.1609/aaai.v30i1.10170.

Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proc. PDCS*, pp. 432–439, 2004.

Natalie Maus, Haydn Jones, Juston Moore, Matt J. Kusner, John Bradshaw, and Jacob Gardner. Local latent space Bayesian optimization over structured inputs. In *Proc. NeurIPS*, pp. 34505–34518, 2022.

Haoliang Ming, Feifei Li, Xiaoqing Wu, and Wenhui Que. Retrieval as reasoning: Self-evolving agent-native retrieval via LLM-Wiki. *arXiv:2605.25480*, 2026.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arık, and Tomas Pfister. MLE-STAR: Machine learning engineering agent via search and targeted refinement. In *Proc. NeurIPS*, pp. 116692–116712, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *Proc. ICLR*, 2026.

José Pombal, Ricardo Rei, and André F. T. Martins. Self-preference bias in rubric-based evaluation of large language models. In *Proc. COLM*, 2026.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, 2026. URL `https://qwen.ai/blog?id=qwen3.8`.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005. ISBN 9780262256834. doi: 10.7551/mitpress/3206.001.0001. URL `https://doi.org/10.7551/mitpress/3206.001.0001`.

John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pp. 65–118. Elsevier, 1976.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Olivier Roussel and Christophe Lecoutre. XML representation of constraint networks: Format XCSP 2.1. *arXiv preprint arXiv:0902.2362*, 2009.

Pierre Rust. pyDCOP: Library for research on distributed constraints optimization problems, 2018. URL `https://github.com/Orange-OpenSource/pyDcop`.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proc. NeurIPS*, pp. 8634–8652, 2023.

Guransh Singh, Vishwajeet Kumar, Arkadeep Acharya, Adnan Qidwai, Jaydeep Sen, and Sachindra Joshi. SearchWiki: Learning to build and navigate knowledge wikis for active information seeking. *arXiv:2608.29953*, 2026.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MPNet: Masked and permuted pre-training for language understanding. In *Proc. NeurIPS*, pp. 16857–16867, 2020.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proc. ICML*, pp. 1015–1022, 2010.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous scientific innovation. In *Proc. NeurIPS*, pp. 9481–9520, 2025.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution. *arXiv:2608.27454*, 2026.

Dhruv Trehan and Paras Chopra. Why LLMs aren't scientists yet: Lessons from four autonomous research attempts. *arXiv:2601.03315*, 2026.

Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In *Proc. NeurIPS*, pp. 11259–11272, 2020.

Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Rong, Olga Kononova, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571:95–98, 2019.

Brian Uzzi, Satyam Mukherjee, Michael Stringer, and Ben Jones. Atypical combinations and scientific impact. *Science*, 342(6157):468–472, 2013. doi: 10.1126/science.1240474.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. Recent advances in Bayesian optimization. *ACM Computing Surveys*, 55(13s), 2023.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. CodeRAG-Bench: Can retrieval augment code generation? In *Proc. NAACL Findings*, pp. 3199–3214, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proc. ICML*, 2025b.

Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proc. EASE*, pp. 1–10, 2014.

Xingyu Wu, Yan Zhong, Jibin Wu, Bingbing Jiang, and Kay Chen Tan. Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation. In *Proc. IJCAI*, pp. 5235–5244, 2024. doi: 10.24963/ijcai.2024/579.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi: 10.1613/jair.2490.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proc. ICLR*, 2023.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Proc. NeurIPS*, pp. 43571–43608, 2024.

Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3:185–207, 2000.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *Proc. ICLR*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *Proc. ICLR*, 2025.

Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proc. AAAI*, pp. 19632–19642, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Roie Zivan, Steven Okamoto, and Hilla Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.

A Foundational Wiki Construction

This section describes the construction of the BO and DCOP wikis visualized in Fig. 3 and summarized in Tab. 2.

Each source card summarizes one source, such as a single paper. Each concept page (a) gives an overview of a method-family, such as multi-fidelity BO or pseudo-trees in DCOP, or (b) compares methods based on a criterion, such as theoretical guarantees. Each concept page (orange) links to the source cards it synthesizes. Additionally, the index node (black) links to the concept pages, and each source card links to related cards. Construction proceeds in five steps: (i) assemble a corpus, (ii) extract each source into a source card, (iii) synthesize concept pages across related cards, (iv) cross-reference all pages, and (v) verify them. Unlike LLM-maintained wikis that are continually revised as new documents arrive (Karpathy, 2026), our wiki is built once from a systematically assembled corpus, verified before use, and remains fixed throughout self-evolution.

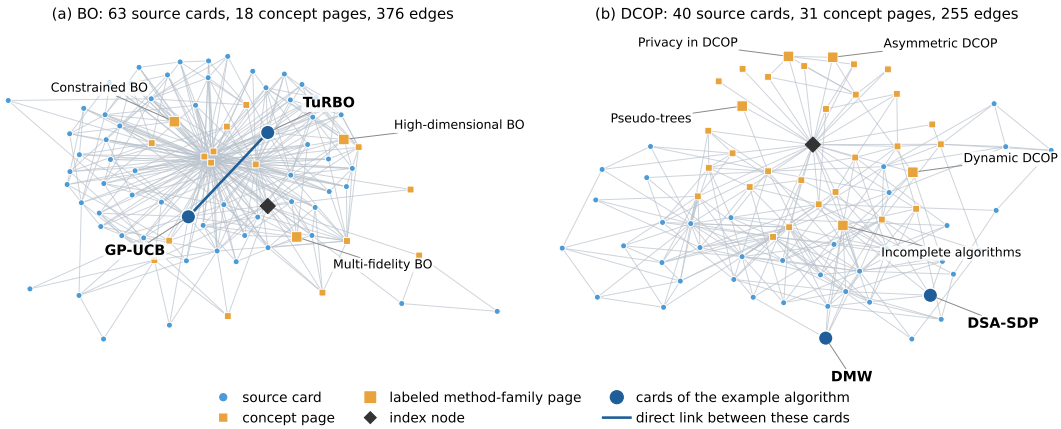


Figure 3: Graphs of the foundational wikis for (left) BO and (right) DCOP: source cards, concept pages, and the index node, connected by their Markdown links (Sec. A). We highlight the two source cards (blue) cited by the best algorithm of App. G and App. F. In the left graph, we highlight the direct link between the TuRBO and GP-UCB source cards. We also clearly annotate some concept pages (orange), such as Multi-fidelity BO.

Table 2: Statistics of the foundational wikis. Concept pages include method-family and comparison pages; edges are those of the undirected graph $\mathcal{W}$ (Sec. A).

	BO	DCOP
Corpus assembly	Snowball sampling	Direct curation
Source cards	63	40
Concept pages	18	31
Edges	376	255
concept–card	237	105
card–card	102	81
concept–concept	19	38
index–concept	18	31

Curator agent. The construction steps are carried out by an LLM agent in Claude Code[1], an agentic coding tool that can read and edit files, run shell commands and scripts, and fetch web pages in a terminal. Specifically, curation involves the following steps: retrieving and archiving papers, reading their full texts, writing and revising many interlinked Markdown files, and running verification scripts. We use Claude Opus 4.6 as the LLM of the curator agent. The curator agent follows a *skill*: a folder of instructions, templates, scripts, and reference documents that an agent loads on demand[2]. Its entry

[1] `https://www.anthropic.com/claude-code`.

[2] `https://docs.anthropic.com/en/docs/agents-and-tools/agent-skills`.

file, `SKILL.md`, specifies the workflow of this section and points to the other files only when a step needs them. The skill is domain-agnostic and works for any domain by specifying the sources (e.g., field of research papers).

Corpus. The skill supports two ways of assembling a corpus: (1) citation snowball sampling and (2) direct curation. The BO corpus is assembled by (1) citation snowball sampling (Wohlin, 2014) from a survey (Wang et al., 2023): from the works the survey cites, the curator agent selects sources that cover every sub-area required by the target tasks, i.e., prompt, hyperparameter, and data-mixture optimization (Sec. 4.1), favouring canonical methods. The corpus is complemented and cross-checked with two community-curated lists of BO papers[3][4], which add work that the survey predates, such as LLM-guided BO; of 29 papers sampled from these lists, 22 (76%) were already in the corpus. Of the 63 BO sources, 60 are research papers whose full text is archived and hash-verified, and three are pedagogical references (a tutorial, a textbook, and a technical note). The DCOP corpus is (2) directly curated to cover the taxonomy of the survey of Fioretto et al. (2018), and each source is checked against its primary paper, venue, or arXiv page.

Source cards and concept pages. For each source, the curator agent reads the full text and writes one source card following the schema in Fig. 4 (left). A card states the source's specific mechanism, including its formula or pseudocode when available, how it differs from the nearest alternative, its experimental findings and limitations, and its verified bibliographic record. A method-family concept page (Fig. 4, right) synthesizes the cards of one family: it tabulates the methods along their distinguishing dimensions, explains why commonly confused methods are not interchangeable, lists typical misapplications, and maps diagnosed failure modes to specific cards. Method-family pages cover, e.g., high-dimensional, multi-fidelity, and constrained BO, or pseudo-trees and dynamic, asymmetric, and privacy-preserving DCOPs. The other concept pages are comparison pages (algorithm selection, guarantees and assumptions, experimental protocols, and model variants).

Source Card Template (BO)

```
---
type: source-card
status: extracted | reviewed
---
# [Author(s) Year -- Short Name]
## Bibliographic Record
## Summary
## Algorithm Overview  (formula /
     pseudocode; nearest alternative)
## Formal Properties
## When to Use
## Known Limitations / Failure Modes
## Experimental Findings and
   Extension Ideas
## Related Cards  (>= 1 link)
## References
```

Concept Page Template (BO)

```
---
type: concept
---
# [Concept Name]
## [Method Family] Taxonomy
     (one row and one link
      per source card)
## Why [A] and [B] Are Not
   Interchangeable
## Common Errors
## When to Use
     (diagnosed failure -> card)
## References  (one link per row)
```

Figure 4: Schemas of a BO source card (left) and a BO method-family concept page (right). Bracketed headings are filled in per page; e.g., the taxonomy section of the multi-fidelity page is titled "The Model-Free vs. Model-Based Spectrum".

Cross-references. $\mathcal{W}$'s nodes are cross-referenced by Markdown hyperlinks of four kinds (Tab. 2). *Index* links connect the index node to every concept page, providing the overview of the concept pages taxonomy. *Concept–concept* links are added by the curator agents between related concept pages. For example, the BO concept page on algorithm selection links to the concept pages on multi-fidelity, high-dimensional, and constrained BO. *Concept–source* links connect each row of a concept page's taxonomy to the corresponding source card; for example, the BO concept page on multi-fidelity BO links to the cards of Hyperband, Bayesian Optimization and HyperBand (BOHB), and multi-fidelity GP methods. *Source–source* links follow a fixed rule of the card schema (Fig. 4): the *Related Cards* section of each source card links (i) the nearest alternative method, with the reason for the comparison, and (ii) the predecessor or successor method that the source extends or is extended by, both extracted from the raw source. For example, the Hyperband card links to its successor, BOHB. In BO, 53% of

[3] `https://github.com/richardcsuwandi/awesome-bo`

[4] `https://github.com/yiminzme/awesome-bayesian-optimization`.

these links coincide with a citation between the two papers; the others relate methods that do not cite each other directly, using the curator agent. Through these links, a research question that matches the multi-fidelity page raises the scores of the Hyperband and BOHB cards through the graph term in Eq. (2).

Verification. The curator agent verifies each source card against its source (the raw full text). A card that passes is marked `reviewed`, and a card that cannot be fully verified remains `extracted`. Concept pages are checked against the cards they link to. In addition, a deterministic linter flags generic or duplicated text, cards without links to other cards, broken links, and unlinked references, and a script checks the archived full texts against their recorded hashes.

Wiki graph. The resulting foundational wiki is the undirected, unweighted graph $\mathcal{W} = (N, E)$ of Sec. 3.1, where $N = S \cup C \cup \{n_{\mathrm{idx}}\}$ and each Markdown file is exactly one node: the index file is n_{idx}, each file extracted from a paper is a source card, and every other page is a concept page. Two nodes are adjacent if either links to the other:

$$E = \big\{\{n_u, n_v\} : n_u, n_v \in N,\ n_u \neq n_v,\ n_u \text{ links to } n_v \text{ or } n_v \text{ links to } n_u\big\}.$$

External links and self-links are ignored, and the wiki remains fixed during self-evolution. Fig. 3 illustrates both graphs. In BO, the TuRBO and GP-UCB cards are linked directly; in DCOP, the DMW and DSA-SDP cards are two hops apart, through the method-family page on incomplete algorithms, so a research question matching that page raises the scores of both cards through the graph term in Eq. (2).

B Retrieval

We retrieve two complementary types of context for candidate generation: previously evaluated candidates and their outcomes from experiential memory, and foundational knowledge from the fixed wiki.

B.1 Experience Retrieval

In iteration t, RetrieveExperience in Alg. 1 selects experiences $\mathrm{E}_t \subseteq \mathcal{E}$. The selection is based on a research agenda $\mathrm{G}_{t'}$ with distinct non-stopword tokens $\mathcal{T}_{\mathrm{G}_{t'}}$.

Recall that an experience $e = (a, \mathtt{c}_a, u_a)$ includes the textual description of the algorithm a (its proposed change, hypothesis, and cited source cards). We define $\mathcal{T}_e$ as the set of distinct non-stopword tokens in a. Each experience $e \in \mathcal{E}$ is scored by its lexical overlap with the agenda $\mathrm{G}_{t'}$,

$$s_{\mathrm{exp}}(e, \mathrm{G}_{t'}) = \frac{|\mathcal{T}_e \cap \mathcal{T}_{\mathrm{G}_{t'}}|}{\sqrt{|\mathcal{T}_e|\,|\mathcal{T}_{\mathrm{G}_{t'}}|}},$$

The retrieved experiences E_t consist of the experience corresponding to the current best algorithm $a^\star$ and the top-3 experiences with the highest $s_{\mathrm{exp}}(e, \mathrm{G}_{t'})$ under the caveat that at most 1 of these experiences can cite the same source and modify the same part of the algorithm code. This caveat is added to promote greater exploration of the wiki.

B.2 Wiki Retrieval

For each research question $q \in Q_t$, we rank nodes in the foundational wiki $\mathcal{W}$ using lexical relevance and graph proximity. As defined in Sec. 3.1, $N = S \cup C \cup \{n_{\mathrm{idx}}\}$ consists of source cards, concept pages, and an index node.

Lexical relevance. For an indexed node $n \in N$, we use a length-normalized TF–IDF-style score:

$$s_{\mathrm{lex}}(n, q) = \frac{1}{1 + \log(1 + L_n)} \sum_{\tau \in \mathcal{T}_q \cap \mathcal{T}_n} f_{\tau,q}(1 + \log f_{\tau,n}) \log\left(1 + \frac{|N|}{1 + \mathrm{df}(\tau)}\right). \tag{1}$$

Here, $\mathcal{T}_q$ and $\mathcal{T}_n$ are sets of distinct tokens; $f_{\tau,q}$ and $f_{\tau,n}$ are the frequencies of token τ in the query and node, respectively; L_n is the node's token count; and $\mathrm{df}(\tau)$ counts indexed nodes containing τ. The sum includes only tokens occurring in both the query and the node.

Graph proximity. We rank the concept and index nodes $C \cup \{n_{\text{idx}}\}$ by $s_{\text{lex}}(n, q)$ and use the top R as traversal seeds. Let $n_q^{(r)}$ be the node/seed at rank r. A source card $n_s \in S$ receives the graph bonus

$$s_{\text{graph}}(n_s, q) = \sum_{\substack{r \in \{1,\dots,R\}: \\ 1 \le \text{dist}(n_q^{(r)}, n_s) \le H}} \frac{\lambda_{\text{graph}}}{r \, \text{dist}(n_q^{(r)}, n_s)}, \tag{2}$$

where dist is shortest-path distance in the unweighted wiki graph $\mathcal{W}$. This favors source cards within H hops of highly ranked seeds, with larger bonuses for shorter paths and higher-ranked seeds. We use $\lambda_{\text{graph}} = 1.5$, $R = 3$, and $H = 2$ for all tasks.

Combining Eq. (1), and Eq. (2), the source card score is calculated by

$$s_{\text{card}}(n_s, q) = s_{\text{lex}}(n_s, q) + s_{\text{graph}}(n_s, q). \tag{3}$$

Merging and repetition penalty. For multiple questions, we merge the per-question concept-page rankings and source card rankings separately by round-robin interleaving with duplicate removal. The stored score of a retained node is its maximum score across the questions. To discourage repeated retrieval, source-card scores in the merged candidate pool are min-max normalized to $\widetilde{s}_{\text{card}}(n_s)$ and adjusted as

$$s_{\text{final}}(n_s) = \widetilde{s}_{\text{card}}(n_s) - \rho_{\text{rep}} h_t(n_s), \qquad \rho_{\text{rep}} = 0.10, \tag{4}$$

where $h_t(n_s)$ counts occurrences of n_s in the supplied recent-source history. If all candidate scores are equal, their normalized scores are set to one. The repetition adjustment is applied only when the recent-source history is nonempty. The highest-ranked nodes supply the concept context $C_t \subseteq C$ and the source-card context $S_t \subseteq S$ used for candidate generation. In the default configuration, the retriever forms a pool the top 16 source cards for DCOP and 35 source cards for BO that having the highest $s_{\text{final}}(n_s)$ scores (Eq. (4)) for candidate generation.

C Surrogate Model Fitting

WikiEvolve decides which candidate algorithms to implement by predicting their utilities before implementation (Sec. 3.3). We view this as learning an unknown function f that maps a candidate algorithm a to its utility, and model f with a Gaussian process (GP) (Rasmussen & Williams, 2005). Each task has its own GP.

Candidate representation. A candidate a is a textual description of its proposed change, the part of the parent algorithm it targets, the mechanism it adapts from its cited source cards, and how the change is integrated into the parent algorithm. The candidate is represented by its ℓ_2-normalized embedding $e(a)$:

$$e(a) = \frac{\text{Enc}(a)}{\|\text{Enc}(a)\|_2} \in \mathbb{R}^{768},$$

where Enc is the frozen `sentence-transformers/all-mpnet-base-v2` encoder (Song et al., 2020).

Why a GP. Candidates that make similar changes to the same part of the parent algorithm have similar embeddings and tend to behave similarly after implementation and achieve similar utilities. Fig. 5 illustrates that the smaller the embedding distance, the smaller the utility difference on average. A GP encodes this belief through its kernel $k(a, a') = \sigma_f^2 \kappa_\ell(e(a), e(a'))$, where σ_f is the signal standard deviation and κ_ℓ is the Matérn-$5/2$ kernel with length scale ℓ (Rasmussen & Williams, 2005),

$$\kappa_\ell(e(a), e(a')) = \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp(-\sqrt{5}r), \qquad r = \frac{\|e(a) - e(a')\|_2}{\ell},$$

which correlates the utilities of two candidates more strongly the closer their embeddings are, relative to ℓ. Beyond predicting the utility, the GP can also quantify the uncertainty in its prediction, which is typically larger for candidates with very different embeddings from previously evaluated candidates. Candidate selection (App. D) uses both the predicted utility and uncertainty estimate.

Noisy utility observations. As algorithms and objectives are stochastic, each candidate a is evaluated on several runs, and its utility u_a (App. E.1) aggregates them and u_a is a noisy estimate of

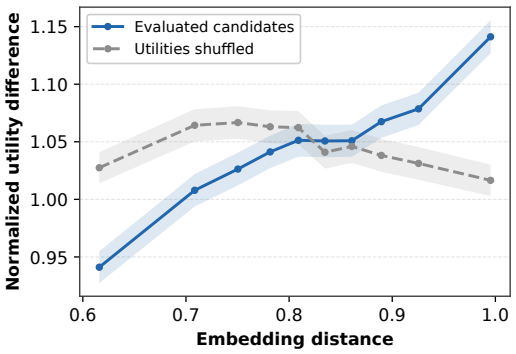


Figure 5: Similar algorithms tend to achieve similar utilities. For all pairs of candidates evaluated within the same self-evolution run of WIKIEVOLVE on HPO and DM (two LLMs, three runs each), the absolute difference of their utilities, divided by the standard deviation of the utilities in that run, against the distance between their description embeddings, averaged in ten bins with equal numbers of pairs (±95% confidence interval). As a reference, the gray curve randomly reassigns the utilities among the candidates of each run, removing any correlation between algorithms and utilities.

$f(a)$. We quantify its precision by the standard error ε_a of u_a, estimated by bootstrapping the runs. Since this precision can differ between candidates, we use a heteroskedastic GP which allows the precision/variance to differ for different candidates. At the end of iteration t, the GP training data $\mathcal{D}_t$ consists of (a, u_a, ε_a) where a is a candidate from the first t iterations whose evaluation completes and whose behavior differs from that of their parent algorithm.

Robust standardization. A poorly performing candidate can have a large negative raw utility and undesirably dominate the GP fit. To avoid this scenario and improve robustness to outliers, we compute a robust location and scale from the fitting set,

$$m = \text{median}_i\ u_{a_i}, \qquad s = \max(c_{\text{MAD}}\,\text{median}_i\ |u_{a_i} - m|,\ 0.05)\,,$$

where $c_{\text{MAD}} = 1/\Phi^{-1}(0.75) \approx 1.4826$ makes the median absolute deviation a consistent estimate of the standard deviation for Gaussian data, and Φ is the standard normal cumulative distribution function. We then lower-clip the utilities at $-c_{\text{clip}}s$ with $c_{\text{clip}} = 3$ and standardize them:

$$y_a = \frac{\max(u_a, -c_{\text{clip}}s) - m}{s}.$$

The clipping threshold is defined on the utility scale, relative to 0, the utility of the initial algorithm, rather than to m: a candidate far worse than the initial algorithm is recorded as clearly bad without dominating the GP fit, and nonnegative utilities are never clipped. In standardized units, the lower bound is $-c_{\text{clip}} - m/s$. The standard errors are rescaled in the same way, $\eta_a = \max(\varepsilon_a/s,\ 0.01)$. The transformation is recomputed from the fitting set at each fit and does not change the utilities stored in the archive.

GP model and fitting. With $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_{a_i}$, we model the standardized utilities as

$$y_{a_i} = \bar{y} + g(e(a_i)) + \xi_i, \qquad g \sim \mathcal{GP}(0, k)\,, \qquad \xi_i \sim \mathcal{N}\left(0,\ \eta_{a_i}^2 + \left(\frac{\rho\nu_0}{s}\right)^2\right),$$

with independent noise terms ξ_i. The first noise term is the candidate-specific standard error; the second is the noise shared by all observations, with base standard deviation $\nu_0 = 0.05$ on the utility scale and a factor ρ, and absorbs variation that the description cannot explain, e.g., two similar ideas implemented differently. Let $\mathbf{y} = [y_{a_1}, \ldots, y_{a_n}]^\top$, $K = [k(a_i, a_j)]_{i,j=1}^n$, and $\Lambda = \text{diag}_i\left[\eta_{a_i}^2 + (\rho\nu_0/s)^2\right] + 10^{-8}I$, where the last term is numerical jitter.

When fitting the GP at the end of iteration t, the hyperparameters ℓ, σ_f, and ρ are chosen from the same grid for all tasks,

$$\ell \in \{0.35, 0.7, 1.4, 2.8\}, \qquad \sigma_f \in \{0.5, 1, 2\} \times q_y, \qquad \rho \in \{0.5, 1\},$$

where $q_y = \max(\mathrm{std}(\mathbf{y}), 0.05)$, to maximize the log marginal likelihood of the fitting set (Rasmussen & Williams, 2005),

$$\log p(\mathbf{y} \mid \ell, \sigma_f, \rho) = -\frac{1}{2}(\mathbf{y} - \bar{y}\mathbf{1})^\top (K + \Lambda)^{-1}(\mathbf{y} - \bar{y}\mathbf{1}) - \frac{1}{2}\log\det(K + \Lambda) - \frac{n}{2}\log(2\pi),$$

which equals the sum of the log predictive densities of the observations, each predicted with Eq. (5) below from the preceding ones. The computation uses the Cholesky factorization of $K + \Lambda$, and configurations for which the factorization fails are discarded.

GP posterior. With the selected hyperparameters and $\mathbf{k}(a) = [k(a_1, a), \ldots, k(a_n, a)]^\top$, the posterior mean and variance of g for a new candidate a follow the standard GP predictive equations with the constant mean $\bar{y}$ (Rasmussen & Williams, 2005), where the diagonal matrix Λ replaces the single noise variance shared by all observations:

$$\mu(a) = \bar{y} + \mathbf{k}(a)^\top (K + \Lambda)^{-1}(\mathbf{y} - \bar{y}\mathbf{1}), \qquad \sigma^2(a) = \max\big(k(a, a) - \mathbf{k}(a)^\top (K + \Lambda)^{-1}\mathbf{k}(a),\ 10^{-10}\big). \tag{5}$$

Undoing the standardization gives the prediction on the utility scale,

$$\mu_t(a) = m + s\,\mu(a), \qquad \sigma_t(a) = s\,\sigma(a).$$

These quantities predict the clipped utility, since the transformation does not undo clipping, and $\sigma_t(a)$ describes the uncertainty about $f(a)$, excluding the noise of a future evaluation. The GP fitted at the end of iteration t is used, unchanged, to score the candidates of iteration $t + 1$ during candidate selection, so each candidate is scored using only candidates evaluated before it. After the selected candidates are evaluated, the new valid observations are added to the archive and the GP is refit; if no new valid observation is added, the existing fit is reused.

D Candidate Selection

Given a candidate pool A_t, we select a subset $A_t^{\mathrm{sub}} \subseteq A_t$ for implementation using contextual relevance, expected improvement, and posterior uncertainty. After warm-up, these criteria receive separate allocations from a fixed implementation budget. Expected improvement (Jones et al., 1998) is a well-known acquisition function in Bayesian optimization that balances exploitation of candidates with high predicted utility and exploration of candidates with high uncertainty, while the uncertainty allocation explicitly further promotes exploration.

Contextual relevance. Let $\mathcal{T}_a$ contain the distinct tokens in the description of a, and let $\mathcal{T}_{Q_t}$ contain the distinct tokens in the concatenated research questions Q_t. For nonempty token sets, we define

$$s_{\mathrm{rel}}(a, Q_t) \triangleq \frac{|\mathcal{T}_a \cap \mathcal{T}_{Q_t}|}{\sqrt{|\mathcal{T}_a|\,|\mathcal{T}_{Q_t}|}}.$$

This score measures lexical overlap with the current questions and is computed independently of the retrieval scores of the sources used to generate a.

Posterior prediction. For each candidate $a \in A_t$, we use the posterior mean $\mu_{t-1}(a)$ and standard deviation $\sigma_{t-1}(a)$ defined in App. C.

Expected improvement. We use expected improvement (EI), a standard acquisition function in Bayesian optimization (Jones et al., 1998), which balances exploitation of candidates with high predicted utility and exploration of candidates with high posterior uncertainty. We use the best observed utility in the full archive, lower-bounded by the utility of the initial baseline algorithm, zero, as the reference:

$$f_t^\star \triangleq \max\left(0, \max_{(a, u_a, \varepsilon_a) \in \mathcal{D}_{t-1}} u_a\right).$$

Let $\Delta_t(a) = \mu_{t-1}(a) - f_t^\star$. For $\sigma_t(a) > 10^{-12}$, the expected improvement under the surrogate posterior is

$$\mathrm{EI}_t(a) = \max(\Delta_t(a)\Phi(z_a) + \sigma_t(a)\phi(z_a), 0), \qquad z_a = \frac{\Delta_t(a)}{\sigma_t(a)},$$

where ϕ and Φ are the standard normal probability density function and cumulative distribution function, respectively. Otherwise, we use the deterministic limit $\mathrm{EI}_t(a) = \max(\Delta_t(a), 0)$. Since

$f_t^\star \geq 0$ and the clipping threshold $-c_{\text{clip}} s$ of App. C is negative, clipping does not alter the reference utility.

Warm-up and budget allocation. Candidates are deduplicated before selection. Let $k' = \min(k, |A_t|)$ be the available implementation budget. While $|\mathcal{D}| < N_{\text{warm}}$, we select the k' candidates with the highest contextual relevance; the default is $N_{\text{warm}} = 20$. After warm-up, we first select

$$k_1 = \min(k_{\text{rel}}, \max(k' - 1, 0))$$

candidates by contextual relevance, then select

$$k_2 = \max(k' - k_1 - 1, 0)$$

remaining candidates by expected improvement. Each stage excludes previously selected candidates, and the union forms A_t^{sub}. With the default $k = 5$ and $k_{\text{rel}} = 2$, a pool of at least five distinct candidates yields two contextual relevance selections, two expected-improvement selections, and one posterior uncertainty selection.

E Experimental Details and Additional Results

E.1 Common Experimental Details

Definition of a task. A task consists of four components: a set of task instances $\mathcal{I}$, an initial algorithm a' and its implementation in one source file, an evaluator that runs an implementation on every instance, and the utility function that aggregates the evaluator's output across instances and runs. Out of these components, the agent may edit only the algorithm's source file. The task instances, evaluator and utility function are fixed and are not visible to the agent as editable code. Each instance is evaluated over multiple runs because both the algorithms (e.g., the random value changes of DSA-B) and the objective (e.g., the observation noise of BoLT) are stochastic. Every run of a candidate uses the same seeds as the corresponding run of a', so the two are comparable run by run.

Utility. Let $c_{a,i,j}$ be the cost of a on instance $i \in \mathcal{I}$ in run $j \in \mathcal{R}_i$ (lower is better; see Appendices E.2 and E.3), and let $s_i = \max\big(|\mathcal{R}_i|^{-1} \sum_{j \in \mathcal{R}_i} |c_{a',i,j}|, 1\big)$ be the mean absolute cost of the initial algorithm. The utility is the mean cost reduction, normalized per instance:

$$u_a = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \frac{1}{|\mathcal{R}_i|} \sum_{j \in \mathcal{R}_i} \frac{c_{a',i,j} - c_{a,i,j}}{s_i},$$

so $u_{a'} = 0$. The normalization keeps large instances from dominating: a cost reduction of 3 counts more when a' has cost 5 than when it has cost 50.

E.2 Distributed Constraint Optimization

DCOP assigns each variable to one agent; agents choose values for their variables and exchange messages only with the agents they share a constraint with, and the goal is to minimize the total cost of the constraints. All three DCOP tasks are evaluated under fixed cycle budgets: 60 cycles for Graph Coloring and Weighted DCOP, and 200 cycles for Max-DisCSP. The evaluator independently recomputes the total constraint cost of the terminal assignment, so improvements are measured under the same configured cycle budget. Tab. 3 summarizes the tasks.

Utility. The cost is the final constraint cost, with 15 runs per instance. We report $100\, u_{a^\star}$ as the improvement (%), as the mean $\pm$ standard deviation over three self-evolution runs. For the Single-Iteration methods, each run has 50 rollouts: Pass@1 is their mean improvement and Best-of-50 the highest.

Graph Coloring. Each variable represents a color drawn from a domain of two to four values, and binary constraints on the graph edges assign costs to pairs of colors. The 18 instances form six groups of three that differ in the cost structure. In three groups, an edge whose endpoints share a color costs 1000: with two colors on bipartite graphs, with two colors on random, grid, and scale-free graphs, and with three or four colors. The bipartite two-color group additionally includes small unary preference costs. In the other three, the cost of equal colors varies across edges (weighted

Table 3: DCOP tasks. Each task has 18 frozen instances in six groups of three; the three instances of a group differ in the number of variables. Every instance is run 15 times (three evaluation seeds with five repetitions each). The initial algorithms are DSA-B with activation probability $p = 0.7$ (Zhang et al., 2005) and MGM (Maheswaran et al., 2004); the agent edits one source file, `dsa.py` in pyDCOP (Rust, 2018) or `MGM.java` in FRODO 2.11.3 (Léauté et al., 2009).

Task	Variables	Domain size	Objective (minimized)	Initial algorithm	Cycles
Graph Coloring	12 / 16 / 20	2-4	constraint cost	DSA-B (pyDCOP)	60
Max-DisCSP	8 / 12 / 16	3 / 4 / 5	violated constraints	MGM (FRODO)	200
Weighted DCOP	12 / 16 / 20	3, 5 or 8	constraint cost (0–9 per pair)	DSA-B (pyDCOP)	60

coloring), each variable additionally has unary costs, or each edge has an arbitrary, asymmetric cost table. Within each group, the three instances have 12, 16, and 20 variables on different topologies (random, grid, bipartite, or scale-free). The initial algorithm is DSA-B: in each cycle, each variable attempts a probabilistic update to a locally cost-minimizing value when this reduces its local cost, or when the local cost is unchanged but at least one incident constraint is not at its individual minimum. The update probability is $p = 0.7$, and ties between eligible values are resolved randomly.

Max-DisCSP. Each of n agents owns one variable that should be assigned a value from the domain $\{1, \ldots, k\}$. Each binary constraint forbids a fraction of the value pairs of its two variables and costs 1 when violated, and the objective is minimizing the number of violated constraints. Instances are generated with FRODO's random Max-DisCSP generator from the number of variables n, the domain size k, the density p_1 (the fraction of variable pairs that share a constraint), and the tightness p_2 (the fraction of forbidden value pairs per constraint). The six groups cover sparse, medium-density, and dense graphs under loose and tight constraints, with group-specific target parameters listed in Tab. 4. Within each group, the three instances have $(n, k) = (8, 3)$, $(12, 4)$, and $(16, 5)$. The initial algorithm is MGM: in each cycle, every agent selects a value that maximally reduces its local cost, exchanges the proposed gain with its neighbors, and changes its value only if the gain is positive and wins the neighborhood comparison, with ties resolved by the implementation's tie-breaking rule (App. F).

Table 4: Max-DisCSP instance groups. p_1 and p_2 denote the generator's target density and tightness, respectively. Constraint counts are listed for $(n, k) = (8, 3), (12, 4), (16, 5)$, respectively, where n is the number of variables and k is the domain size.

Group	p_1	p_2	Constraints
Sparse, loose (S1)	0.20	0.30	6 / 13 / 24
Sparse, tight (S2)	0.25	0.80	7 / 17 / 30
Medium, loose (S3)	0.45	0.35	13 / 30 / 54
Medium, tight (S4)	0.45	0.80	13 / 30 / 54
Dense, loose (S5)	0.70	0.35	20 / 46 / 84
Dense, tight (S6)	0.70	0.80	20 / 46 / 84

The smallest instance, in the sparse, loose group, is stored in FRODO's XCSP 2.1 format (Roussel & Lecoutre, 2009) (abridged below). Relation `r_0` gives cost 1 to the value pairs $(1, 1)$ and $(2, 2)$ and cost 0 to all others, and constraint `c_0` applies it to x_5 and x_1; the agent's algorithm sees only its own variable, its domain, and the constraints it shares with its neighbors.

Example Max-DisCSP instance

```
<instance>
  <presentation maxConstraintArity="2" maximize="false" format="XCSP 2.1_FRODO">
    <stats name="number of variables">8</stats>
    <stats name="domain size">3</stats>
    <stats name="true average density p1">0.214</stats>
    <stats name="true average tightness p2">0.222</stats>
  </presentation>
  <agents nbAgents="8"> <agent name="a0"/> ... <agent name="a7"/> </agents>
  <domains nbDomains="1"> <domain name="D" nbValues="3">1..3</domain> </domains>
  <variables nbVariables="8">
    <variable name="x0" domain="D" agent="a0"/> ...
  </variables>
```

```
  <relations nbRelations="6">
    <relation name="r_0" arity="2" semantics="soft" defaultCost="0"
              nbTuples="2">1: 1 1 |2 2</relation> ...
  </relations>
  <constraints nbConstraints="6">
    <constraint name="c_0" arity="2" scope="x5 x1" reference="r_0"/> ...
  </constraints>
</instance>
```

Weighted DCOP. Each constraint between two variables assigns a cost between 0 and 9 to every pair of values, and the objective is the total cost. The six groups cross two constraint-graph topologies, random and scale-free, with three domain sizes, 3, 5, and 8; within each group, the three instances have 12, 16, and 20 variables and 20 to 36 constraints. The initial algorithm is DSA-B with the same configuration as in Graph Coloring.

Example of a best-found algorithm. App. F shows how WikiEvolve changed MGM on Max-DisCSP: it replaced the greedy choice of the highest-gain value with sampling from a softmax over the gains, combining mechanisms from two retrieved source cards, and kept MGM's neighborhood competition unchanged.

E.3 Bayesian Optimization

A BO task requires the algorithm to maximize an expensive black-box function with a fixed number of queries. In each step, the algorithm chooses the next point x_b from the observations so far and receives a noisy value of $f_i(x_b)$. The initial algorithm is GP-EI in BoTorch (Balandat et al., 2020): it fits a Gaussian process to the standardized observations and queries the point that maximizes expected improvement (Jones et al., 1998). The agent may modify only the functions that define the BO loop, namely the initial design, the choice of the next query, the main loop, and the projection onto the constraints of the search space. The BoLT objectives are emulators, neural networks trained on the results of real LLM training runs, so that a query is cheap while the objective behaves like the real one (Chew et al., 2026).

Utility. Let $f_i^{\max}$ be the maximum of the noiseless objective of instance i, and $r_{a,i,j}(B) = f_i^{\max} - \max_{b \le B} f_i(x_b)$ the simple regret of a after B queries on instance i with seed j, computed on the noiseless objective (Chew et al., 2026). During self-evolution, the cost is $r_{a,i,j}(60)$; since all regrets are below 1, $s_i = 1$. Following (Chew et al., 2026), we report the reduction in geometric-mean regret over GP-EI, $100\,(1 - e^{\Delta(B)})\%$, where $\Delta(B)$ is the mean of $\log r_{a^\star,i,j}(B) - \log r_{a',i,j}(B)$ (regrets floored at 10^{-10}), at the self-evolution budget $B = 60$ and at $B = 200$, which the algorithm never saw during self-evolution. For example, $\Delta(B) = -\log 2$ halves the regret of GP-EI, a reduction of 50%, and -100% doubles it. Over the three runs, we report $100(1 - e^{\bar{\Delta}}) \pm 100\, e^{\bar{\Delta}}\, \mathrm{sd}(\Delta)$, a first-order approximation. For the Single-Iteration methods, Pass@1 uses the mean Δ of the 50 rollouts, counting a crashed or timed-out rollout as $\Delta = 0$, and Best-of-50 is the rollout with the highest utility at $B = 60$.

HPO. The algorithm tunes the LoRA fine-tuning of Qwen3-8B-Base (Yang et al., 2025). The search space has seven parameters (Tab. 5), mixing continuous, integer, and categorical types. HPO has three task instances: this 7-dimensional problem and two 8-dimensional multi-fidelity variants that add a fidelity parameter, either the model size (Qwen3-4B or Qwen3-8B) or the number of training tokens (normalized over 10^5 to 9.1×10^6 tokens).

DM. The algorithm chooses the mixture of three training datasets, instruction following, math, and code (the Tulu-3 persona subsets (Lambert et al., 2025)), for continued training of Qwen3-4B-Base in two stages: the first mixture is used for the first 5M training tokens and the second for the next 5M. The six parameters therefore form two 3-simplices; each group must sum to one, and a query that violates this constraint is rejected by the evaluator. The objective averages the scores on IFEval (Zhou et al., 2023), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), and MBPP+ (Liu et al., 2023).

For example, the HPO query $x = (0,\ 2,\ 2,\ 2,\ 0.5,\ 30,\ 2)$ fine-tunes with the smallest learning rate, batch size, rank, and α, dropout 0.5, LoRA on all 30 layers, and the third target-module option; a DM query $x = (0.2,\ 0.6,\ 0.2,\ 0.1,\ 0.8,\ 0.1)$ trains mostly on math in both stages.

Table 5: Search spaces of the BO tasks. Continuous parameters are normalized to $[0, 1]$ by BoLT.

Task	Parameter	Type	Range
HPO	learning rate	continuous	$[0, 1]$
	batch size	integer	$\{2, 3, 4\}$
	LoRA rank	integer	$\{2, \ldots, 5\}$
	LoRA α	integer	$\{2, \ldots, 5\}$
	LoRA dropout	continuous	$[0, 1]$
	LoRA layers	integer	$\{1, \ldots, 30\}$
	LoRA target module	categorical	4 choices
DM	stage-1 proportions (IF, math, code)	simplex	sum to 1
	stage-2 proportions (IF, math, code)	simplex	sum to 1

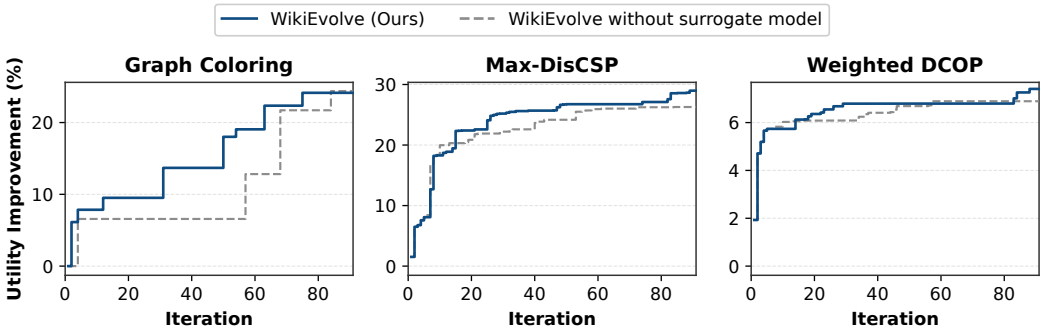


Figure 6: Effect of the surrogate model on Graph Coloring, Max-DisCSP, and Weighted DCOP over 90 iterations with Qwen3.8-27B (mean over three seeds).

Example of a best-found algorithm. App. G shows a returned DM algorithm that replaces expected improvement with GP-UCB and optimizes it inside a TuRBO-style trust region, with its code.

E.4 Effect of the Surrogate Model

We further compare WikiEvolve with a variant that removes the surrogate model from candidate selection (Fig. 6). For the ablation, we select top-k' contextual relevant candidates by using the same method in App. D. On Graph Coloring, the two variants eventually reach similar final utility, but WikiEvolve improves more steadily over the course of self-evolution. On Max-DisCSP and Weighted DCOP, using the surrogate model leads to better final performance. Overall, these results show that the surrogate model helps guide candidate selection during self-evolution.

E.5 Utility against Iterations

Fig. 7 compares how utility improves over successive iterations across three DCOP tasks. WikiEvolve achieves the highest final utility in all three tasks over the baselines: 18.01% vs. 12.61% for the best baseline on Graph Coloring, 26.74% vs. 18.68% on Max-DisCSP, and 6.79% vs. 5.06% on Weighted DCOP. These graphs suggest that combining foundational wiki knowledge with accumulated experimental experience can help the agent discover stronger algorithms over time.

F DCOP Example

We use an evolved algorithm for Max-DisCSP as a concrete example to illustrate how WikiEvolve combines an evolving parent algorithm with mechanisms retrieved from the foundational wiki.

Task and original algorithm. Max-DisCSP is a distributed constraint optimization problem in which agents jointly assign values to variables to minimize the number or cost of violated constraints. We use *maximum gain move* (MGM) as the initial algorithm. In each cycle, an MGM agent evaluates

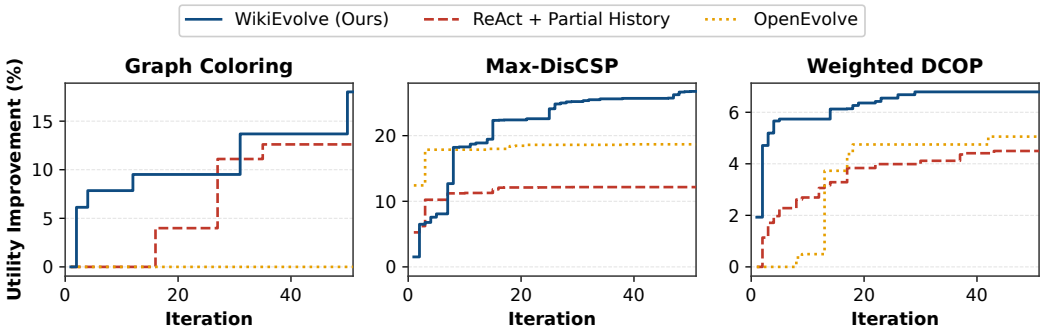


Figure 7: Utility improvement of the best algorithm found so far against self-evolution iterations on the DCOP tasks with Qwen3.8-27B, for WikiEvolve, ReAct + Partial History, and OpenEvolve (mean over three seeds).

Table 6: Three sources of the evolved Max-DisCSP algorithm and their contributions relative to the original MGM baseline.

Source	Mechanism provided	Role in the evolved algorithm
Current parent	MGM communication and neighborhood competition, together with an inherited annealing-based acceptance rule	Provides the algorithmic backbone and controls whether a proposed move is eventually executed
DMW-Game	Represents alternative decisions using a probability distribution	Motivates replacing a single deterministic proposal with a distribution over candidate moves
DSA-SDP	Makes stochastic decisions dependent on the magnitude of local improvement	Motivates using local gains to determine the relative probabilities of candidate moves

alternative values under the current assignments of its neighbors and greedily proposes the value with the largest local gain. For a minimization problem, the local gain of changing the current value x_i to an alternative v is

$$\Delta_i(v) = c_i(x_i, x_{-i}) - c_i(v, x_{-i}). \tag{6}$$

Standard MGM selects

$$v_i^\star = \arg\max_{v \neq x_i} \Delta_i(v), \tag{7}$$

communicates the corresponding gain to neighboring agents, and allows an update only when the proposal wins the neighborhood gain competition.

The main limitation relevant to this example is that the local proposal in MGM is deterministic: once the local gains are computed, the agent always proposes the single highest-gain move. This restricts the set of local search trajectories that can be explored.

Sources contributing to the evolved algorithm. The evolved algorithm can be understood as combining mechanisms from three sources: (i) the current evolutionary parent derived from MGM, (ii) DMW-Game (Hatano & Yoshida, 2015) retrieved from the foundational wiki, and (iii) DSA-SDP (Zivan et al., 2014) retrieved from the foundational wiki. Each source contributes a different component relative to the original MGM baseline (Tab. 6).

Source 1: the evolutionary parent. The immediate parent is itself derived from MGM. It preserves MGM's original communication and neighborhood competition, but already contains an annealing-based acceptance mechanism for non-improving proposals. Therefore, before the current evolutionary step, its decision process can be summarized as

$$\underbrace{\arg\max_v \Delta_i(v)}_{\text{greedy proposal}} \rightarrow \underbrace{\text{annealing acceptance}}_{\text{inherited from parent}} \rightarrow \underbrace{\text{MGM competition}}_{\text{original MGM}}. \tag{8}$$

The current evolutionary step preserves the latter two components and modifies only how the local proposal is generated.

Source 2: probabilistic choice from DMW-Game. The retrieved DMW-Game source represents an agent's alternatives using a probability distribution rather than committing immediately to a single deterministic choice. The evolved algorithm does not reproduce DMW-Game itself: it does not introduce multiplicative-weight updates or communicate probability vectors between agents. Instead, WIKIEVOLVE transfers the more general mechanism of representing multiple candidate actions probabilistically and adapts it to MGM's local value-selection step.

Relative to the original MGM baseline, this changes the proposal representation from

$$\text{one selected move} \quad \Longrightarrow \quad \text{a distribution over alternative moves.} \tag{9}$$

Source 3: gain-dependent stochasticity from DSA-SDP. The retrieved DSA-SDP source provides a complementary mechanism: stochastic decisions are related to the magnitude of local improvement. This suggests that candidate moves should not be sampled uniformly; instead, moves with larger local gains should receive higher probability.

Relative to the original MGM baseline, the local gains $\{\Delta_i(v)\}$ are therefore used not only to identify the maximum-gain move, but also to control the stochastic preference among alternative moves.

Combining the three sources. WIKIEVOLVE combines the probabilistic-choice mechanism inspired by DMW-Game with the gain-dependent stochasticity inspired by DSA-SDP, while retaining the acceptance and coordination mechanisms of the current parent. Concretely, instead of selecting the single highest-gain move as in Eq. (7), the evolved algorithm constructs

$$p_i(v) = \frac{\exp(\Delta_i(v))}{\sum_{v' \neq x_i} \exp(\Delta_i(v'))}, \qquad \tilde{v}_i \sim p_i. \tag{10}$$

The resulting algorithm can therefore be decomposed as

$$\underbrace{\tilde{v}_i \sim \mathrm{Softmax}(\Delta_i(v))}_{\substack{\text{probabilistic choice from DMW-Game} \\ + \text{ gain-dependent stochasticity from DSA-SDP}}} \rightarrow \underbrace{\text{annealing acceptance}}_{\text{current parent}} \rightarrow \underbrace{\text{MGM competition}}_{\text{MGM backbone}}. \tag{11}$$

When one candidate move has a substantially larger gain than the others, the distribution in Eq. (10) concentrates on that move and the proposal behaves similarly to greedy MGM. When several moves have comparable gains, multiple proposals receive non-negligible probability, allowing the algorithm to explore search trajectories that deterministic MGM would not take. A sampled non-improving proposal is not necessarily executed: it is subsequently filtered by the annealing-based acceptance mechanism inherited from the parent and by the original MGM neighborhood competition.

Importantly, the two retrieved source algorithms are not directly copied. DMW-Game contributes the idea of probabilistic choice, DSA-SDP contributes gain-dependent stochasticity, and the current parent supplies the existing acceptance and coordination structure. The final algorithm is therefore a mechanism-level synthesis of these three sources within the MGM backbone.

We present snapshots for both original MGM and WIKIEVOLVE MGM in one case in Fig. 8.

G BO EXAMPLE

We use an evolved algorithm for DM as a concrete example to illustrate how WIKIEVOLVE combines an evolving parent algorithm with mechanisms retrieved from the foundational wiki. The algorithm was returned with Qwen3.8-27B (seed 1101). Its regret reduction over GP-EI is 77.10% with 60 evaluations, the budget used during self-evolution, and 52.84% with 200 evaluations.

Throughout this section, b counts the evaluations made by a BO algorithm, and x_b is its b-th query, as in App. E.1; we avoid t, which denotes self-evolution iterations elsewhere.

Task and original algorithm. DM searches six continuous mixture weights, which form two 3-simplices (one per training stage), to maximize the downstream performance of the trained model. The initial algorithm is GP-EI: at each step, it fits a Gaussian process with posterior mean μ_b and standard deviation σ_b to the observations and queries the point that maximizes expected improvement over the best observed value $f_b^\star$ over the whole domain $\mathcal{X}$,

$$x_{b+1} = \arg\max_{x \in \mathcal{X}} \mathbb{E}\left[\max\left(f(x) - f_b^\star, 0\right)\right]. \tag{12}$$

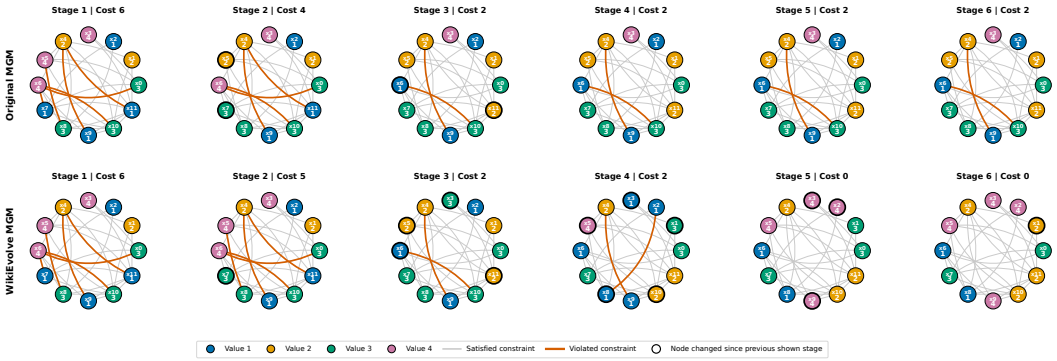


Figure 8: Six snapshots of the original MGM (top) and WikiEvolve's evolved MGM of App. F (bottom) on one Max-DisCSP instance with 12 variables, which are controlled by 12 agents separately, and four possible values per variable, and 30 binary constraints. Each circle is a variable, colored by its current value; orange edges are violated constraints, and and a thick black outline marks variables whose value changed since the previous snapshot. Both algorithms start from the same assignment with cost 6 and reach similar assignments with cost 2 by stage 3, where the original MGM stops changing. The evolved MGM instead changes five variables at stage 4 without lowering the cost, and reaches cost 0 at stage 5, consistent with its sampling among moves with comparable gains (Eq. (10)).

Table 7: Three sources of the evolved DM algorithm and their contributions relative to the initial GP-EI algorithm.

Source	Mechanism provided	Role in the evolved algorithm
Current parent	GP-UCB acquisition with a noise-adaptive confidence schedule	Provides the surrogate model and the acquisition function
GP-UCB	Explicit confidence multiplier β_b balancing exploration and exploitation	Provides the acquisition rule that is optimized; the optimization is changed
TuRBO	Trust region centered at the best observed point	Restricts where the acquisition function is optimized

Sources contributing to the best-found algorithm. The best-found algorithm, *GP-UCB with Trust-Region Local Search*, was generated in iteration 8 and remained the parent for the rest of the run. It can be understood as combining mechanisms from three sources (Tab. 7): (i) the current evolutionary parent derived from GP-EI, (ii) GP-UCB (Srinivas et al., 2010), and (iii) TuRBO (Eriksson et al., 2019), the latter two retrieved from the foundational wiki and cited by the candidate.

Source 1: the evolutionary parent. The lineage replaced EI with GP-UCB before the online iterations, and in iteration 6 accepted a noise-adaptive confidence schedule. The parent thus queries

$$x_{b+1} = \arg\max_{x \in \mathcal{X}} \; \mu_b(x) + \sqrt{\beta_b}\, \sigma_b(x), \qquad \beta_b = \frac{2}{5} \log \frac{(b+1)\pi^2}{6\delta} \cdot \mathrm{clip}\left(1 + \rho_b,\, 0.5,\, 3\right), \tag{13}$$

with $\delta = 0.1$, where b is the number of evaluations made so far and ρ_b is the noise variance of the Gaussian process fitted to the standardized observations divided by the sample variance of the raw observations. Because the two variances are on different scales, ρ_b is a heuristic rather than a true noise-to-signal ratio. The factor $1/5$, present since the warm-up, matches an empirical finding recorded in the GP-UCB source card: scaling down the theoretical β_b improves GP-UCB in practice.

Diagnosis. In iteration 6, the agent accepted the candidate with the noise-adaptive confidence schedule (the parent above), whereas another candidate of the same iteration, which changed the kernel to reflect the simplex geometry, regressed. The research planning of iterations 7 and 8 asks about this contrast; the agent's "iteration 1" and "iteration 3" refer to the first and third candidates of iteration 6:

Research question on the target failure (iterations 7 and 8)

Why did the GP-UCB with Noise-Adaptive Confidence Schedule (iteration 1) show a positive benefit against the baseline and parent, yet the subsequent attempts to modify the kernel geometry (Simplex-Aware Kernel, iteration 3) resulted in a significant regression, suggesting that the noise-adaptive beta schedule is effective only when paired with a kernel that accurately models the simplex geometry, and that the current RBF kernel's isotropic assumption is a limiting factor?

The candidate that became the best-found algorithm, the third candidate of iteration 8, states the hypothesis:

Hypothesis of the returned candidate (iteration 8)

The global acquisition optimization is unreliable in high-dimensional spaces, leading to suboptimal candidate selection. Using a trust-region local search, where the acquisition function is optimized within a local region around the current best point, will improve the search direction and performance.

Sources 2 and 3: GP-UCB and TuRBO. The candidate keeps the GP-UCB acquisition function of Eq. (13) unchanged and cites the GP-UCB. From the TuRBO card, it takes the trust region: TuRBO optimizes the acquisition function within a hyperrectangle around the best observed point, and resizes it according to consecutive successes and failures. The candidate notes that TuRBO uses Thompson sampling whereas the plan uses UCB, and that the trust region is the transferable component.

Combining the three sources. The evolved algorithm optimizes the parent's acquisition function only within a trust region around the best observed point $x_b^\star$:

$$x_{b+1} = \arg\max_{x \in \mathcal{T}_b} \mu_b(x) + \sqrt{\beta_b}\,\sigma_b(x), \qquad \mathcal{T}_b = [x_b^\star - L_b(u-l),\ x_b^\star + L_b(u-l)] \cap \mathcal{X}, \quad (14)$$

where l and u are the bounds of the domain and $L_b = \mathrm{clip}\,(0.5/(1+0.1b),\ 0.05,\ 0.5)$. If the trust region becomes narrower than 1% of the domain in any dimension, the algorithm optimizes over the whole domain instead. Compared with TuRBO, the trust region shrinks with a fixed schedule rather than in response to successes and failures. The main newly synthesized mechanism relative to the parent is therefore

$$\boxed{\arg\max_{x \in \mathcal{X}} \mu_b(x) + \sqrt{\beta_b}\,\sigma_b(x) \quad \Longrightarrow \quad \arg\max_{x \in \mathcal{T}_b} \mu_b(x) + \sqrt{\beta_b}\,\sigma_b(x)}. \quad (15)$$

The parent contributes the acquisition function and its noise-adaptive schedule, GP-UCB the acquisition rule, and TuRBO the restriction of its optimization to a local region.

Implementation. We show the code that changes between the initial GP-EI algorithm and the best-found algorithm, all inside the function `select_next_point`. Numbers on the left are line numbers in the source files, and `# [...]` marks omitted lines. Both versions standardize the observations and fit the same Gaussian process; outside this function, the two files differ only in the imported acquisition function.

GP-EI fits a Gaussian process and maximizes expected improvement over the whole domain:

Initial algorithm (GP-EI): `select_next_point`, lines 60–73

```
60      gp, fitted_state = fit_single_task_gp(
61          train_x, standardized_y, warm_start_state=_warm_start_state
62      )
63
64      best_f = standardized_y.max()
65      acq_func = ExpectedImprovement(model=gp, best_f=best_f)
66
67      candidate, _ = optimize_acqf(
68          acq_function=acq_func,
69          bounds=bounds,
70          q=q,
71          num_restarts=num_restarts,
72          raw_samples=raw_samples,
73      )
```

The best-found algorithm replaces expected improvement with GP-UCB and sets its confidence multiplier β_b as in Eq. (13). Here `_iteration` is the number of evaluations made so far and

`noise_to_signal` is the ratio ρ_b. The code comments attribute the schedule and its scaling by $1/5$ to Srinivas et al. (2010), as recorded in the GP-UCB source card:

Best-found algorithm: noise-adaptive GP-UCB confidence multiplier, lines 93–109

```
93      # GP-UCB: beta_t = 2 * log(t * pi^2 / (6 * delta))
94      # Using delta = 0.1 for 90% confidence, and a practical scaling factor
95      # to avoid overly conservative exploration (per Srinivas et al. 2010 findings)
96      t = max(_iteration + 1, 1)
97      delta = 0.1
98      beta_t = 2.0 * math.log(t * math.pi**2 / (6.0 * delta))
99      # Scale down by factor of 5 as recommended in practice (Srinivas et al. 2010)
100     beta_t = beta_t / 5.0
        # [...]
106     noise_adjustment = 1.0 + noise_to_signal
107     # Clamp the adjustment to a reasonable range [0.5, 3.0] to maintain stability
108     noise_adjustment = max(0.5, min(3.0, noise_adjustment))
109     beta_t = beta_t * noise_adjustment
```

It then optimizes the acquisition function only inside the trust region of Eq. (14). The trust region is a box of half-width L times the domain width around the best observed point, clipped to the domain. When the box becomes narrower than 1% of the domain, the algorithm falls back to the whole domain:

Best-found algorithm: trust region around the best observed point, lines 122–167

```
122     best_idx = train_y.argmax().item()
123     best_point = train_x[best_idx].clone()
        # [...]
135     L = 0.5 / (1.0 + 0.1 * _iteration)
136     L = max(0.05, min(0.5, L))  # Clamp to [0.05, 0.5]
        # [...]
142     tr_lb = best_point - L * (ub - lb)
143     tr_ub = best_point + L * (ub - lb)
144     # Clip to the global domain
145     tr_lb = torch.clamp(tr_lb, min=lb, max=ub)
146     tr_ub = torch.clamp(tr_ub, min=lb, max=ub)
        # [...]
153     tr_width = (tr_ub - tr_lb).min().item()
154     domain_width = (ub - lb).min().item()
155     if tr_width < 0.01 * domain_width:
156         # Trust region is too small, use global search
157         tr_lb = lb.clone()
158         tr_ub = ub.clone()
        # [...]
161     acq_func = UpperConfidenceBound(model=gp, beta=beta_t)
        # [...]
165     candidate, _ = optimize_acqf(
166         acq_function=acq_func,
167         bounds=torch.stack([tr_lb, tr_ub], dim=0),
            # [...]
```

Comparison with experience-based evolution. At the larger budget $B = 200$, the best-found algorithms of Experience Only are worse than GP-EI on DM (Tab. 1). To illustrate why, we compare the best-found algorithm above with the best-found algorithm of Experience Only on the same configuration, i.e., same task, LLM, and seed in Tab. 8 and their code for selecting the next query below.

Both best-found algorithms replace expected improvement with UCB, but they set its confidence multiplier and the region over which it is optimized differently. In the code below, comments are omitted, the function-body indentation is removed, two long lines are wrapped, and `# [...]` marks omitted lines; numbers on the left are line numbers in the source files.

Table 8: Best-found algorithms of Experience Only and WIKIEVOLVE on DM (Qwen3.8-27B, seed 1101), and their improvement (%) over the initial algorithm GP-EI at the tuning budget $B = 60$ and the larger budget $B = 200$.

	Experience Only	WIKIEVOLVE
Best-found algorithm	Simplex-Adaptive UCB (iteration 10)	GP-UCB with Trust-Region Local Search (iteration 8)
Lineage from GP-EI	Rescales the noise variance of the Gaussian process (iteration 9), then replaces EI with UCB (iteration 10)	Replaces EI with GP-UCB (warm-up), adapts its confidence multiplier to noise (iteration 6), then adds a trust region (iteration 8)
Cited source cards	None	GP-UCB and TuRBO (Tab. 7)
Hypothesis of the best-found algorithm	"The fixed EI acquisition function does not adapt to the simplex constraint structure"	"The global acquisition optimization is unreliable"
Confidence multiplier	constant, $\beta = 2\,(1 + 0.5\,n_{\text{groups}})(1 + 0.1\,d_{\text{simplex}}) = 6.4$	Grows with the number of evaluations (Eq. (13))
Acquisition optimized over	Whole domain	Trust region around the best observed point that shrinks as evaluations accumulate (Eq. (14))
Improvement, $B = 60$	63.99%	**77.10%**
Improvement, $B = 200$	-113.98%	**52.84%**

Experience Only: Simplex-Adaptive UCB

```
100  total_simplex_dims = sum(end - start
         for start, end in simplex_groups)
101  num_simplex_groups = len(simplex_groups)
105  simplex_factor = 1.0 + 0.5 * num_simplex_groups
107  dim_factor = 1.0 + 0.1 * total_simplex_dims
109  beta = 2.0 * simplex_factor * dim_factor
     # [...]
115  acq_func = UpperConfidenceBound(model=gp,
                                     beta=beta)
117  candidate, _ = optimize_acqf(
118      acq_function=acq_func,
119      bounds=bounds,
```

WIKIEVOLVE: GP-UCB with Trust-Region Local Search

```
 96  t = max(_iteration + 1, 1)
 98  beta_t = 2.0 * math.log(t * math.pi**2
                             / (6.0 * delta))
100  beta_t = beta_t / 5.0
     # [...]
135  L = 0.5 / (1.0 + 0.1 * _iteration)
136  L = max(0.05, min(0.5, L))
142  tr_lb = best_point - L * (ub - lb)
143  tr_ub = best_point + L * (ub - lb)
     # [...]
161  acq_func = UpperConfidenceBound(model=gp,
                                     beta=beta_t)
165  candidate, _ = optimize_acqf(
166      acq_function=acq_func,
167      bounds=torch.stack([tr_lb, tr_ub], dim=0),
```

Both best-found algorithms improve on GP-EI at the tuning budget, but at the larger budget the best-found algorithm of Experience Only more than doubles the regret of GP-EI, whereas that of WIKIEVOLVE keeps a 52.84% improvement. This difference is consistent with how the two algorithms optimize the acquisition function: Experience Only optimizes it over the whole domain with a constant confidence multiplier, whereas WIKIEVOLVE optimizes it within a trust region that shrinks as evaluations accumulate, so its queries concentrate around the best observed point at larger budgets. The best-found algorithms of Experience Only are below GP-EI in all three DM runs with Qwen3.8-27B (-113.98%, -68.76%, and -31.26%). The contrast also holds for the candidate algorithms: over these three runs, 90% of the candidate algorithms of WIKIEVOLVE name an established BO method, such as TuRBO, SAASBO, knowledge gradient, or Thompson sampling, compared with 9% for Experience Only, whose candidate algorithms mostly adjust the simplex handling and confidence multiplier of their parent algorithm.

H RESEARCH AGENDA AND QUESTION TEMPLATES

As described in Sec. 3.2, the research planning stage first constructs a high-level research agenda G_t and then formulates research questions Q_t to guide subsequent wiki retrieval. We provide the templates and one generated example below.

Template.

Research Agenda Template

`hypothesis`: A falsifiable explanation for the observed limitation, stated as a hypothesis rather than an established cause.

`target_failure`: The specific measured limitation or unresolved performance gap to investigate.

`plan`: What knowledge to retrieve and what minimal comparison to conduct, without prescribing the algorithm before retrieval.

`success_criteria`:

- Improve the task metric against the explicitly named comparison method.
- Preserve performance on previously successful cases within the evaluator's tolerance.
- Complete the required evaluations with valid outputs and preserved execution constraints.

Research Question Template

`target_failure`: What mechanisms could explain [measured limitation] under [task conditions], and what evidence would distinguish competing explanations?

`orthogonal_mechanism`: What alternative mechanisms address [missing capability] beyond the current method's existing approach, using information available in this setting?

`limitation_or_conflict`: Under what assumptions could these mechanisms fail or conflict with [existing method or execution constraints], and what observations would reveal that conflict?

Example. We show an example generated during the first outer iteration when evolving MGM for Max-DisCSP. The labels S1–S6 index the six instance groups of Tab. 4 in order; S2, S4, and S6 are the tight groups.

Generated Research Agenda

`hypothesis`: The fixed 200-cycle horizon in MGM is insufficient for resolving long-range conflicts in high-tightness and dense Max-DisCSP instances, leading to premature convergence and suboptimal violation costs.

`target_failure`: Suboptimal violation costs in high-tightness and dense scenarios due to premature convergence within the fixed cycle limit.

`plan`: Retrieve evidence on adaptive cycle termination, dynamic threshold adjustment, and conflict-aware local search in DCOP algorithms. Design a minimal experiment to test whether extending the effective search horizon or adjusting the acceptance threshold based on local conflict signals improves violation costs in high-tightness scenarios without regressing sparse, low-tightness instances.

`success_criteria`:

- All requested development blocks complete with valid outputs.
- The candidate produces an observable behavior change in high-tightness scenarios.
- The primary metric (total violation cost) improves against baseline and parent in high-tightness scenarios.
- No scenario regresses beyond the task evaluator's lineage threshold.

Generated Research Questions

`target_failure`: Why does the baseline MGM exhibit high variance and suboptimal costs in high-tightness scenarios (S2, S4, S6) despite having sufficient message volume, and is this due to premature convergence or insufficient conflict resolution depth?

`orthogonal_mechanism`: How can the local utility estimation in MGM be augmented with a lightweight, non-intrusive signal (e.g., conflict frequency or neighbor disagreement) to dynamically adjust the acceptance threshold or cycle allocation without changing the message format?

`limitation_or_conflict`: What are the risks of introducing adaptive cycle termination or threshold adjustment in MGM, and could these changes destabilize the algorithm on sparse, low-tightness instances (S1) where the baseline already achieves near-optimal results?