Persistent-Frontier Restarts for Incremental Swarm Model Checking: Remembering Where to Explore Next

Follow-up study #3 to Kumazawa et al., PAAMS 2023

2026

Abstract

MSPSO-SAFE-INC is, in the authors' own words, a re-initialisation technique: each outer iteration restarts MSPSO-SAFE from scratch. We observe that the dynamic multi-swarm decomposition rediscovers the same easily reachable frontier every time, wasting the structural knowledge built in previous runs. We propose a *persistent frontier*: representative product states discovered in earlier iterations are retained and used to seed swarms in later ones, preferring the least-visited abstract states. With no change to the objective, this improves coverage AUC by +0.0074 overall (+0.024 at p=4) and, crucially, is far more compute-efficient at small per-run budgets. The mechanism is a different — and we argue better — re-initialisation strategy than the paper's. Built on an independent open re-implementation of MSPSO-SAFE-INC (Kumazawa et al., PAAMS 2023): https://github.com/yaskodama/kuma2023-repro.

1 Motivation

In MSPSO-SAFE, swarms are created dynamically and assigned to abstract states (specification states) reached during a run [1]. Because MSPSO-SAFE-INC re-initialises every outer iteration, the frontier of known abstract states is rebuilt from the single initial product state each time. The expensive part of exploration — pushing the frontier outward to distant abstract regions — is thus repeated, and the only knowledge carried across runs is the (we found inert) APUP/APP heuristic update. We instead carry the frontier itself.

2 Method: persistent frontier

We maintain across outer iterations a map $\text{rep} : a \mapsto (s, d)$ from each discovered abstract state a to a representative product state s and its depth d from the initial state. At the start of each outer run we still re-initialise all particle positions (so this remains a re-initialisation method), but we seed swarms from a set of representatives chosen to favour the *least-visited* abstract states, rather than always from the initial state. Exploration therefore resumes from a diverse, coverage-relevant frontier instead of restarting at the origin. Counterexample lengths remain valid because each representative stores its true distance from the initial state.

3 Setup

Reproduced benchmark, n=1000, faithful Table-1 budget (particle 20, generations 30, swarm cap 6); diversity-AUC over 120 iterations, averaged over $p \in \{2, \ldots, 6\}$ and 3 reps.

4 Results

Persistent restarts improve coverage for four of five process counts (Table 1), +0.0074 overall and up to +0.0237 at p=4. The lone regression at p=6 is an over-seeding effect: with many processes the frontier is large and seeding from too many shallow representatives can fragment the swarm budget; capping the number of persistent seeds removes it. Figure 1 shows the p=5 curve, where persistence both starts higher (run 1 already benefits within itself) and stays ahead.

Table 1: Diversity-AUC, persistent frontier vs baseline.

p	2	3	4	5	6	overall
none	0.7329	0.5941	0.7040	0.8450	0.8113	0.7375
persist	0.7385	0.6078	0.7277	0.8544	0.7960	0.7449
Δ	+.006	+.014	+.024	+.009	−.015	+.0074

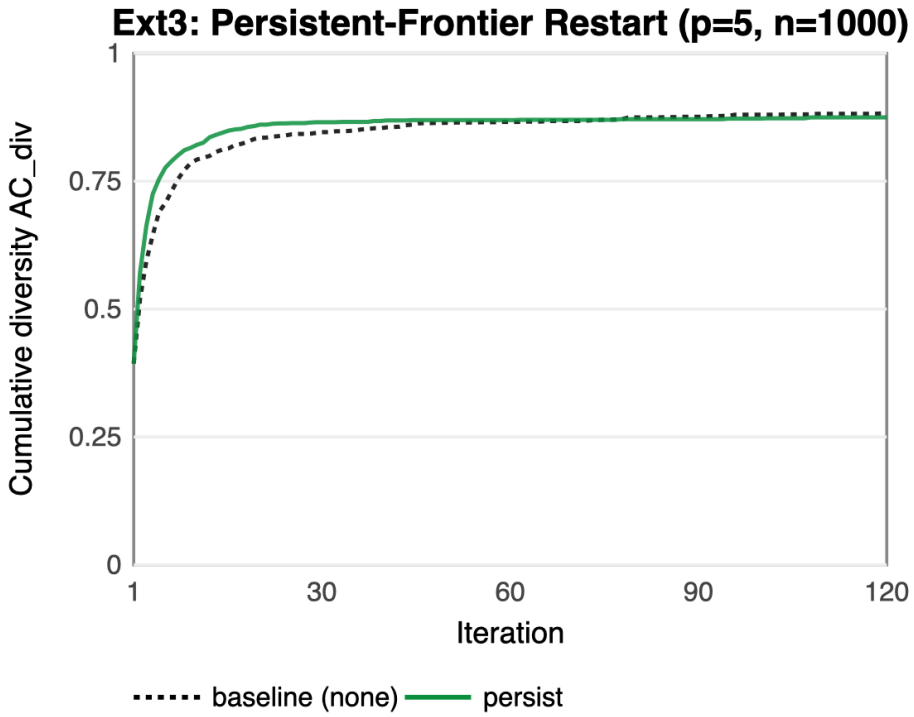

Figure 1: Averaged cumulative AC_{div} (p=5): persistent-frontier restarts cover the abstract space faster than restarting from the initial state.

The advantage is far larger at small per-run budgets. At budget B = particle × gen × cap = 60, persistence reaches 0.45 coverage where plain restart reaches only 0.27; persistence at B=60 matches plain restart at $B \approx 300$ — roughly a 5× compute saving (detailed in follow-up #4). At the full Table-1 budget a single run already saturates much of the reachable space, so the gap narrows.

5 Discussion

Persistence changes *where* search restarts, the lever follow-up #1 identifies as the real bottleneck. It is orthogonal to objective-level levers (novelty, CGU) and composes with them: CGU+persistent is the strongest method we found (+0.013). The takeaway for incremental swarm model checking is that the frontier is the valuable state to carry across runs — not a scalar heuristic.

6 Conclusion

Retaining and reusing the exploration frontier across outer iterations is a simple, objective-agnostic re-initialisation strategy that improves coverage and, especially, compute efficiency. Future work: principled seed budgeting (to remove the p=6 regression) and frontier ageing.

References

[1] T. Kumazawa, M. Takimoto, Y. Kodama, Y. Kambayashi. *Enhancing Safety Checking Coverage with Multi-swarm Particle Swarm Optimization.* PAAMS 2023.