COS790 Assignment 3 Report
Hybrid Perturbative Hyper-Heuristic (HPHH)

u20416823
Ruan Carlinsky

November 2, 2025

1 Description of the approach employed to hybridise both the hyper heuristics.

The Hybrid Perturbative Hyper-Heuristic (HPHH) integrates two perturbative frameworks—Selection Perturbative Hyper-Heuristic (SPHH) and Generative Perturbative Hyper-Heuristic (GPHH)—into a unified three-stage optimization pipeline. The hybrid exploits SPHH's rapid intensification and GPHH's capacity to learn structured perturbation programs, under strict evaluation-budget control.

1.1 How it works

How it works: Given an objective f, bounds (lo, hi), dimension D and a total evaluation budget, the hybrid splits that budget into three stages: SPHH pre-optimization, GPHH evolution, and program application. First, SPHH intensifies locally by repeatedly selecting one of several perturbation operators (Gaussian/Cauchy steps, coordinate resets, opposition blend, pull-to-best) with a bandit (UCB) selector, accepting proposals via simulated annealing (SA) and adapting step sizes with the 1/5-success rule, to produce a strong incumbent (xbest, fbest). Next, GPHH uses a genetic program over a small AST (APPLY/SEQ/REPEAT/IF with conditions IMPROVES, RAND_LT, TEMP_GT) to evolve a macro-heuristic: each candidate program is interpreted for a small, fixed mini-budget from a random start using the same SA acceptance and bounds clamping, and its fitness is the best value reached; selection, crossover, and mutation improve programs within the GP sub-budget (population and generations are auto-capped so pop $\times$ per_prog $\times$ (1+gens) fits). Finally, the best evolved program is applied starting from the SPHH incumbent for the remaining budget with true per-evaluation annealing (temperature updated every evaluation) and SA acceptance, continually updating the global best. The solver returns the best solution seen anywhere (either SPHH's own best or the post-program best), along with a per-stage evaluation breakdown, runtime, and a best-so-far history suitable for convergence plotting.

1.2 Overview of the Hybridization Strategy

HPHH runs in three stages: (1) SPHH pre-optimization to quickly obtain a strong incumbent, (2) GPHH evolution to learn a compositional perturbation program within a

fixed sub-budget, and (3) application of the evolved program starting from the SPHH incumbent. Each stage draws from a predefined fraction of the global evaluation budget; the final application stage uses the remainder after accounting for the actual evaluations consumed by Stages 1–2.

Stage budgets:

$$\begin{aligned} E_{sphh} &= \text{round}(\alpha_{sphh} \cdot E_{max}), \\ E_{gp} &= \text{round}(\alpha_{gp} \cdot E_{max}), \\ E_{apply} &= \max(0, E_{max} - (used_{sphh} + used_{gp})) \end{aligned}$$

1.3 Stage 1: Selection Perturbative Hyper-Heuristic (SPHH)

SPHH serves as a fast intensification engine controlled by a multi-armed bandit (UCB) selector over a portfolio of atomic perturbative heuristics:

Primitive heuristics: `gaussian_full`, `gaussian_kdims`, `cauchy_full`, `random_reset_coord`, `opposition_blend`, `pull_to_best`

Bounds: all proposals are clamped to [lo, hi] per coordinate.

Heuristic selection uses UCB:

$$\text{choose } i \text{ that maximizes } r_i + c \cdot \sqrt{\ln N / n_i},$$

where r_i is the average reward for heuristic i, n_i its pulls, N total pulls, and $c \approx 1.5$.

Acceptance is simulated annealing (SA) by default:

$$\text{accept if } \Delta f \leq 0; \text{ otherwise with probability } \exp(-\Delta f / T).$$

Temperature T decays exponentially $T = T_0 \cdot \exp(-t/\tau)$ over the SPHH budget.

Step-size adaptation: follows the 1/5-success rule to maintain progress efficiency (expanding or shrinking step scales per dimension).

Reward shaping:

- primary reward from proportional improvement (clipped to [0,1]);
- small positive reward for accepted uphill moves to preserve exploration early.

Output: incumbent (x_{best}, f_{best}), evaluation count, and best-so-far history.

Zero-budget nuance: if $E_{sphh} = 0$, we still perform one random evaluation to seed a valid incumbent.

1.4 Stage 2: Generative Perturbative Hyper-Heuristic (GPHH)

GPHH evolves a compositional perturbation program via genetic programming (GP). Programs are ASTs built from:

Primitive operators: `GAUSS_FULL`, `GAUSS_KDIMS`, `CAUCHY_FULL`, `RESET_COORD`, `OPP_BLEND`, `PULL_TO_BEST`

Control structures: `SEQ(...)`, `REPEAT(k, body)`, `IF(cond){then}{else}`

Conditions: `IMPROVES`, `RAND_LT(p)`, `TEMP_GT(t)`

Interpreter semantics (fitness evaluator): Executes the program from a random start for a mini-budget (`eval_budget_per_prog`), with SA acceptance and exponential cooling. Fitness is the minimum objective value reached during interpretation.

Genetic operators:

- **Selection:** tournament ($k = 4$)
- **Crossover:** subtree exchange with depth control
- **Mutation:** subtree replacement or in-place numeric parameter perturbation (e.g., `sigma_rel`, `rate`, `jitter_rel`); `GAUSS_KDIMS` adjusts k within [1, D]
- **Elitism:** best program preserved each generation

Budget control:

- **Safe population sizing:** choose a population that fits the GP sub-budget at least for the initial evaluation.
- **Max generations:** cap so $(initial + gens) \times (pop \times per\_prog)$ stays within E_{gp}.
- GP usage is accounted exactly; the application stage gets the true remainder.

Output: best-evolved program and the evaluations consumed.

1.5 Stage 3: Program Application Phase

The best program is applied starting from SPHH's (x_{best}, f_{best}) using the same interpreter semantics (operators, control flow, SA acceptance, bounds clamping). Two important details:

- **Always stepwise annealing:** application advances exactly one evaluation per step and recomputes T each evaluation via an exponential schedule from $T_0 = 1.0$ to $T_{end} = 10^{-3}$ across the remaining apply budget. This behavior is independent of verbosity and ensures true per-eval annealing.
- **IF(IMPROVES) semantics:** the interpreter first evaluates THEN; if it improves, it keeps that result; otherwise it evaluates ELSE and adopts it only if it improves. This avoids wasting extra budget when THEN already improved.

The application continues until the apply budget is exhausted, continually updating the global incumbent.

1.6 Integration and Hybrid Mechanisms

- **Shared state:** the SPHH incumbent seeds the application state for the learned program.
- **Consistent SA:** both evolution and application use SA with exponential cooling and Metropolis acceptance; application now updates T every evaluation.
- **Budget awareness:** stages compute actual usage and terminate exactly at their sub-budget; the apply budget is the remainder after the true cost of Stages 1–2.
- **Graceful fallbacks:** if $E_{gp} = 0$, a minimal `APPLY(GAUSS_FULL)` program is used so application can still proceed; if $E_{sphh} = 0$, a single random evaluation seeds a valid incumbent.

1.7 Rationale for Hybridization

SPHH excels at fast local intensification with adaptive step control and bandit-based operator selection. GPHH learns structured, multi-step strategies (ordering, repetition, conditional branching) that can generalize beyond single moves. The hybrid exploits both: SPHH finds a promising region quickly; GPHH evolves a macro-heuristic; the apply phase uses stepwise annealing to refine or escape, producing robust behavior across functions and dimensions.

1.8 Pseudocode Summary

Algorithm 1 Hybrid Perturbative Hyper-Heuristic (HPHH)

Require: f, (lo, hi), D, E_{max}, α_{sphh}, α_{gp}

1: Compute $E_{sphh} = \text{round}(\alpha_{sphh} \cdot E_{max})$, $E_{gp} = \text{round}(\alpha_{gp} \cdot E_{max})$
2: **Stage 1:** run SPHH for up to E_{sphh} evaluations $\rightarrow (x_{best}, f_{best}), used_{sphh}$
3: **Stage 2:** evolve GPHH for up to E_{gp} evaluations (safe pop, capped gens) $\rightarrow$ best program P, $used_{gp}$
4: $E_{apply} = \max(0, E_{max} - used_{sphh} - used_{gp})$
5: **Stage 3:** apply P from (x_{best}, f_{best}) for E_{apply} steps with per-eval annealing
6: **Output:** $\min(f_{sphh}, f_{prog})$, plus per-stage evaluation breakdown and runtime

1.9 Default Parameterization

- **Budget fractions:** $\alpha_{sphh} = 0.1$, $\alpha_{gp} = 0.1$ ($\approx$80% reserved for application)
- **SPHH:** $c = 1.5$, cooling fraction = 0.2, selection = `ucb`, acceptance = `sa`
- **GPHH:** population = 60, generations = 20, per-program budget = 1000, tree depth = 5, tournament = 4, crossover = 0.8, mutation = 0.2
- **Bounds clamping:** all operators clamp to [lo, hi] per dimension.

1.10 Conceptual Summary

HPHH forms a seamless pipeline where SPHH delivers a strong launch point, GPHH evolves a specialized macro-heuristic, and the final stage applies it with true per-evaluation annealing. The result is a synergistic blend of adaptive selection and generative learning that respects strict evaluation budgets, maintains consistent stochastic dynamics, and performs robustly across the f1–f24 benchmark suite and dimensions.

2 Experimental Setup

2.1 Functions

Functions: f1–f24 from `benchmark_functions.py`.

Dimensions:

- f1, f2: base 2D only (registry keys `f1`, `f2`).

- f3–f24: $D \in \{10, 30, 50\}$ with keys like `f21_D10`, `f21_D30`, `f21_D50`.

Bounds: per-function/per-dimension bounds from the registry; all proposals are clamped to [lo, hi] per coordinate.

2.2 Runs and Seeding

- **Runs per objective:** 10 independent runs with seeds 0..9.
- **Runner:** `python3 hphh/run_hphh_suite.py --auto --seeds 0..9 --out results/hphh_results.csv`
- **Per-run reporting:** CSV row per (objective, seed). Optional per-problem CSVs in `results/hphh_by_problem/`.

2.3 Global Budgeting

- **Total evaluations per run:** `--evals 10000`.
- **Stage fractions:**
  - SPHH fraction `--sphh-frac 0.1` $\Rightarrow$ nominal SPHH budget $\approx$ 1000 evals.
  - GPHH fraction `--gp-frac 0.1` $\Rightarrow$ nominal GP-evolution budget $\approx$ 1000 evals.
  - Apply remainder $\approx$ 8000 evals (the exact remainder uses actual Stage-1/2 usage).
- **Step annealing:** Application stage always updates temperature every single evaluation (per-eval annealing), independent of verbosity.

2.4 Stage 1 — SPHH (Selection PHH)

- **Selector:** UCB with $c = 1.5$.
- **Acceptance:** Simulated annealing (SA); accept if $\Delta f \leq 0$ else with $\exp(-\Delta f/T)$.
- **Cooling:** Exponential; $T_0 \approx |f_{best}| + 10^{-9}$, $T(t) = T_0 \cdot \exp(-t/\tau)$, with `cooling_frac = 0.2` ($\tau = \max(5, \text{int}(cooling\_frac \cdot budget))$).
- **Heuristics portfolio:** `gaussian_full`, `gaussian_kdims`, `cauchy_full`, `random_reset_coord`, `opposition_blend`, `pull_to_best`.
- **Step sizes:** initialized to $0.1 \cdot (hi - lo)$; adapt via 1/5 rule every 50 evals ($\times 1.2$ on success rate > 0.2; else $\times 0.82$).
- **Initialization:** random uniform in [lo, hi] (if SPHH budget is 0, do one random evaluation to seed an incumbent).
- **Output tracked:** f_{best}, x_{best}, `history_best`, evaluations.

2.5 Stage 2 — GPHH (Generative PHH)

Program representation (AST):

- **Primitives:** `GAUSS_FULL`, `GAUSS_KDIMS`, `CAUCHY_FULL`, `RESET_COORD`, `OPP_BLEND`, `PULL_TO_BEST`.
- **Control:** `SEQ(...)`, `REPEAT(k, body)`, `IF(cond){then}{else}` with cond $\in$ {`IMPROVES`, `RAND_LT(p)`, `TEMP_GT(t)`}.

GP parameters (defaults used):

- `--gp-pop 60`, `--gp-gens 20`, `--per-prog 1000` (evals per program),
- `--depth 5`, `--tk 4`, `--pcx 0.8`, `--pmut 0.2`.

Fitness evaluation per program:

- Start from a random point; interpret the program for `per_prog` evaluations with SA acceptance and bounds clamping; fitness = best value reached.

Budget fitting (actual behavior with 10% GP budget = 1000):

- Safe population `safe_pop` is downscaled to fit the GP sub-budget at least for the initial population evaluation.
- Cost per generation = `safe_pop` $\times$ `per_prog`; number of generations capped so $(initial + gens) \times cost\_per\_gen \leq$ GP budget.
- With defaults and `evals=10000, gp_frac=0.1, per_prog=1000` $\Rightarrow$ `safe_pop=1`, initial evaluation uses 1000 evals, leaving 0 for generations (i.e., evolution reduces to a best-of-1 program under the evaluator).

Output tracked: best program string, best fitness, and actual GP evals used.

2.6 Stage 3 — Apply Evolved Program

- **Start point:** SPHH's (x_{best}, f_{best}).
- **Annealing:** exponential from $T_0 = 1.0$ to $T_{end} = 10^{-3}$, recomputed every evaluation (true per-eval annealing).
- **Acceptance:** SA/Metropolis with current T; interpreter semantics identical to the evolution evaluator.
- **IF(IMPROVES) behavior:** evaluate THEN; if it improves, keep it; else evaluate ELSE and keep it only if it improves (reduces wasted budget).
- **Output tracked:** `history_best`, final f_{best}, and apply evals used.

2.7 System Technical Specifications

- **Machine**: DESKTOP-0GJMIQK
- **CPU**: 11th Gen Intel® Core™ i5-1135G7 @ 2.40 GHz
- **Memory**: 8 GB RAM (7.75 GB usable)
- **System**: 64-bit OS, x64-based processor
- **OS**: Windows 11 (build 10.0.26100, SP0)
- **GPU**: not used (CPU-only experiments)
- **Python**: 3.12.3
- **Libraries**: NumPy (random `default_rng`, vectorized math), Python standard library (`dataclasses`, `time`, `typing`)

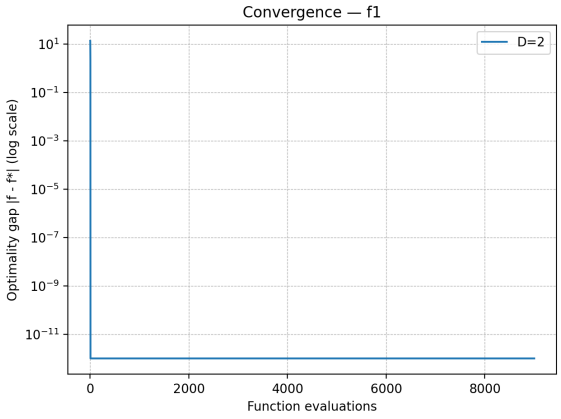
3 Results

Table 1: Hybrid Perturbative Hyperheuristic summary across all functions and dimensions

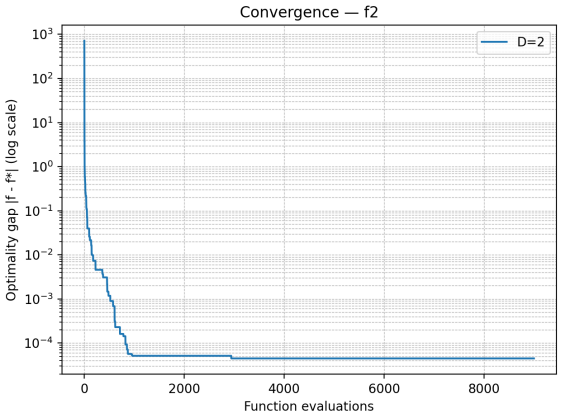
Function	Dim	Runs	Best	Average	Std	Mean Runtime (s)
f1	2	10	0.000000	0.000000	0.000000	1.048759
f2	2	10	-1.031620	-1.031589	0.000017	1.048856
f3	10	10	0.000000	0.000000	0.000000	1.080918
f3	30	10	0.000000	0.000000	0.000000	1.106639
f3	50	10	0.000000	0.000000	0.000000	1.237810
f4	10	10	0.000000	0.000000	0.000000	1.099879
f4	30	10	0.000000	0.000000	0.000000	1.133183
f4	50	10	0.000000	0.000000	0.000000	1.130703
f5	10	10	0.000000	0.000000	0.000000	1.192731
f5	30	10	0.000000	0.000000	0.000000	1.321679
f5	50	10	0.000000	0.000000	0.000000	1.309296
f6	10	10	0.000000	0.000000	0.000000	1.201067
f6	30	10	0.000000	0.000000	0.000000	1.144922
f6	50	10	0.000000	0.000000	0.000000	1.184675
f7	10	10	0.000000	0.000000	0.000000	1.193124
f7	30	10	0.000000	0.000000	0.000000	1.213754
f7	50	10	0.000000	0.000000	0.000000	1.195085
f8	10	10	0.000000	0.000000	0.000000	1.184624
f8	30	10	0.000000	0.000000	0.000000	1.246403
f8	50	10	0.000000	0.000000	0.000000	1.313231
f9	10	10	0.000000	0.000000	0.000000	1.424427
f9	30	10	0.000000	0.000000	0.000000	1.251305
f9	50	10	0.000000	0.000000	0.000000	1.302311

Continued on next page

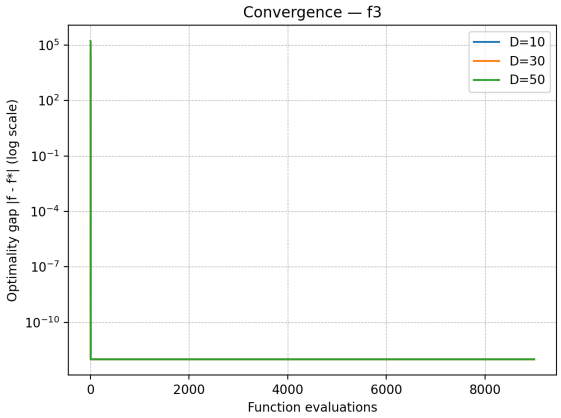
Function	Dim	Runs	Best	Average	Std	Mean Runtime (s)
f10	10	10	0.000000	0.000000	0.000000	1.445460
f10	30	10	0.000000	0.000000	0.000000	1.405726
f10	50	10	0.000000	0.000000	0.000000	1.422893
f11	10	10	0.000000	0.000000	0.000000	1.186390
f11	30	10	0.000000	0.000000	0.000000	1.217302
f11	50	10	0.000000	0.000000	0.000000	1.327322
f12	10	10	0.000021	0.002360	0.002463	1.276840
f12	30	10	0.000144	0.002649	0.002847	1.949933
f12	50	10	0.000324	0.002801	0.002241	1.770696
f13	10	10	0.000000	0.000000	0.000000	1.760626
f13	30	10	0.000000	0.000000	0.000000	1.746183
f13	50	10	0.000000	0.000000	0.000000	1.739496
f14	10	10	0.000000	0.000000	0.000000	1.870778
f14	30	10	0.000000	0.000000	0.000000	1.804862
f14	50	10	0.000000	0.000000	0.000000	1.822844
f15	10	10	0.000000	0.000000	0.000000	1.600936
f15	30	10	0.000000	0.000000	0.000000	1.660409
f15	50	10	0.000000	0.000000	0.000000	1.590224
f16	10	10	0.000000	0.000000	0.000000	1.405844
f16	30	10	0.000000	0.000000	0.000000	1.485720
f16	50	10	0.000000	0.000000	0.000000	1.610407
f17	10	10	-78.288568	-76.848803	1.586491	2.049128
f17	30	10	-77.839536	-68.725517	5.670716	1.873518
f17	50	10	-74.776565	-62.800976	7.027156	1.638117
f18	10	10	0.000000	0.000000	0.000000	1.524905
f18	30	10	0.000000	0.000000	0.000000	1.572880
f18	50	10	0.000000	0.000000	0.000000	1.714522
f19	10	10	0.000000	0.000000	0.000000	1.957263
f19	30	10	0.000000	0.000000	0.000000	2.046632
f19	50	10	0.000000	0.000000	0.000000	1.805202
f20	10	10	0.000308	0.007364	0.013233	2.117149
f20	30	10	0.045953	0.121425	0.076129	2.106112
f20	50	10	0.081830	0.227125	0.074865	2.195060
f21	10	10	14.551142	504.242365	474.670664	1.489535
f21	30	10	589.918111	4700.258557	2681.657431	1.595713
f21	50	10	2593.899160	10222.368756	4656.340247	1.885399
f22	10	10	0.000000	0.000000	0.000000	2.006043
f22	30	10	0.000000	0.000000	0.000000	1.947064
f22	50	10	0.000000	0.000000	0.000000	2.108814
f23	10	10	0.007065	0.024140	0.015133	2.142683
f23	30	10	2.108005	2.739435	0.280302	2.019351
f23	50	9	4.642782	4.925955	0.111401	2.228545
f24	10	10	0.000465	0.020063	0.037299	1.751133
f24	30	10	0.511310	0.852331	0.189039	1.919733
f24	50	10	1.483534	2.482901	0.555180	1.898463



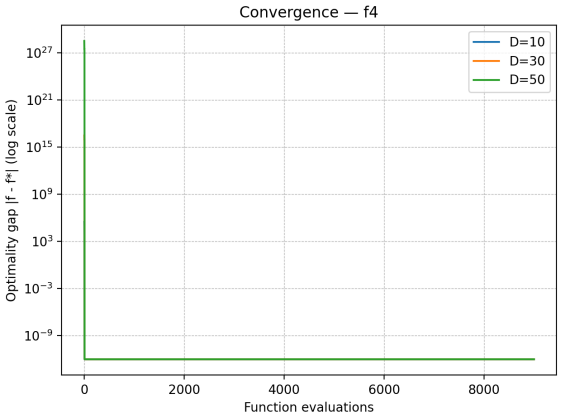
(a) Convergence on f_1



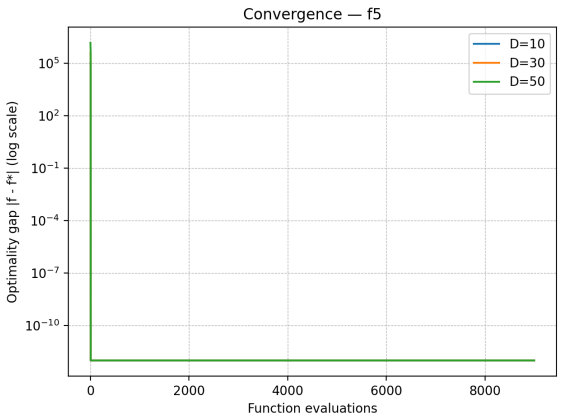
(b) Convergence on f_2



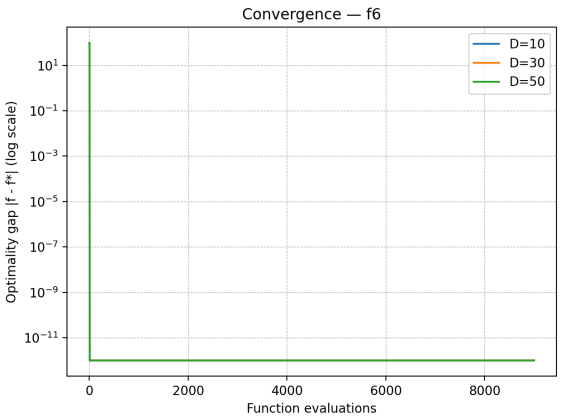
(a) Convergence on f_3



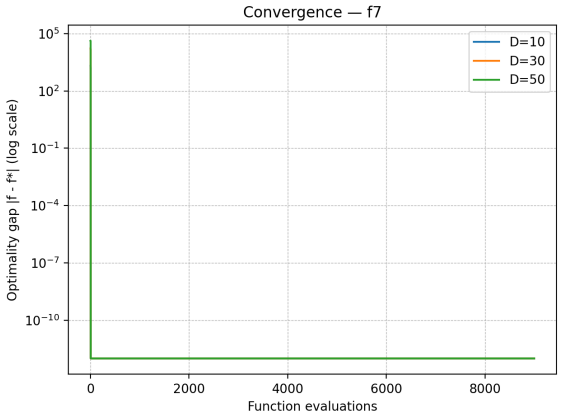
(b) Convergence on f_4



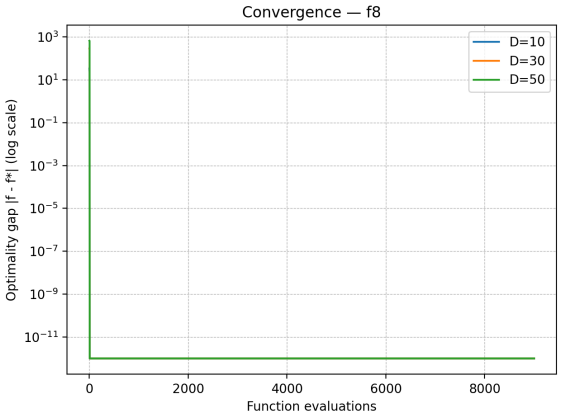
(a) Convergence on f_5



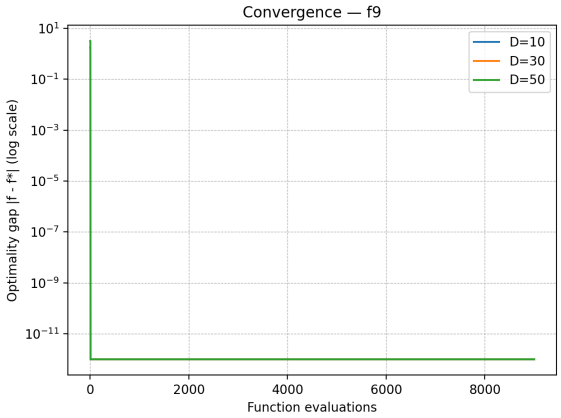
(b) Convergence on f_6



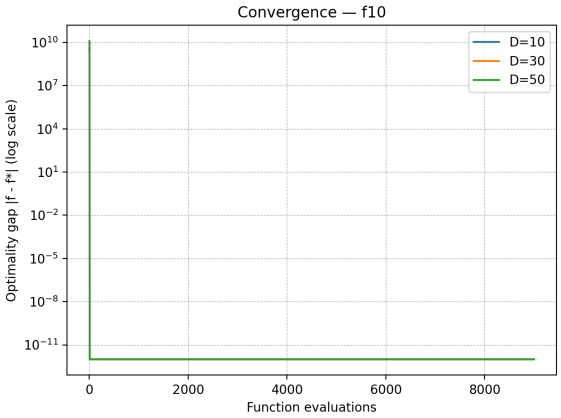
(a) Convergence on f_7



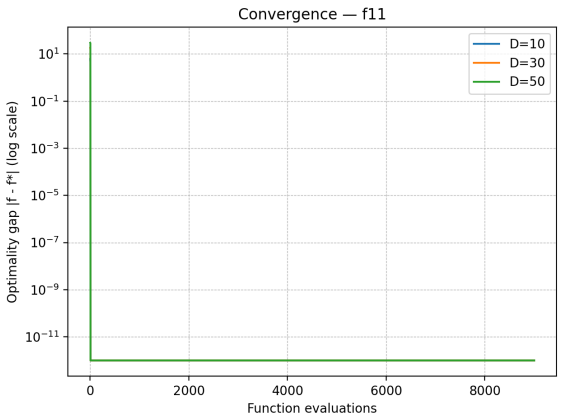
(b) Convergence on f_8



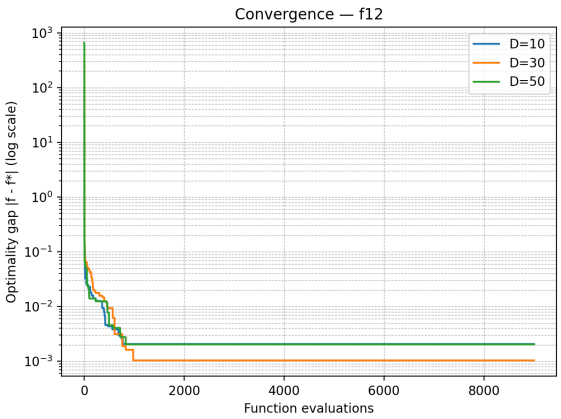
(a) Convergence on f_9



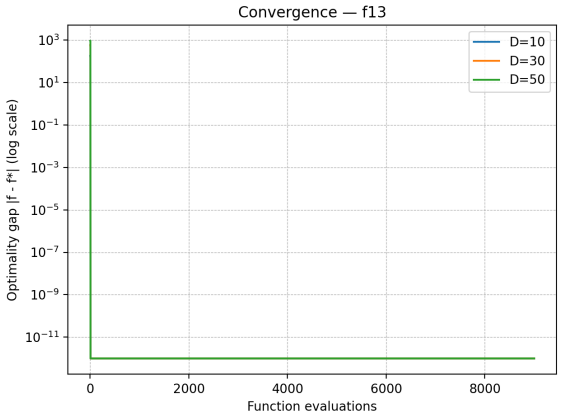
(b) Convergence on f_{10}



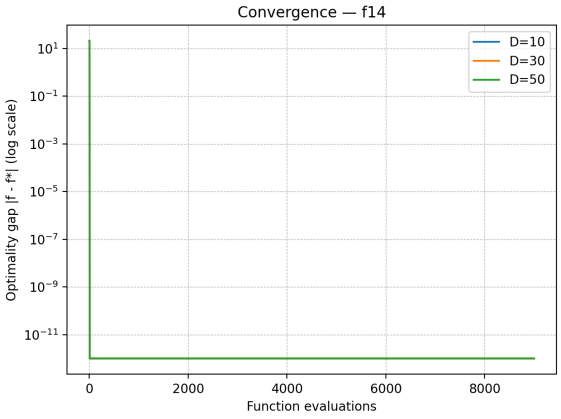
(a) Convergence on f_{11}



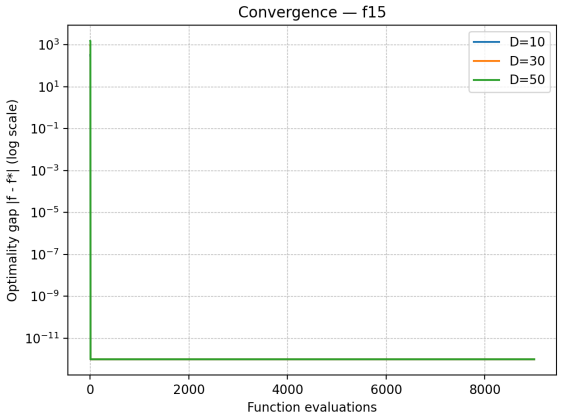
(b) Convergence on f_{12}



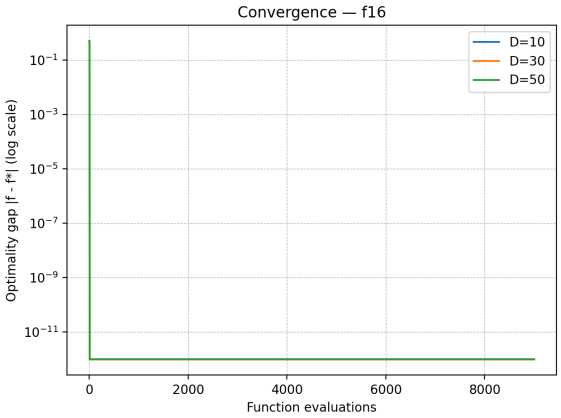
(a) Convergence on f_{13}



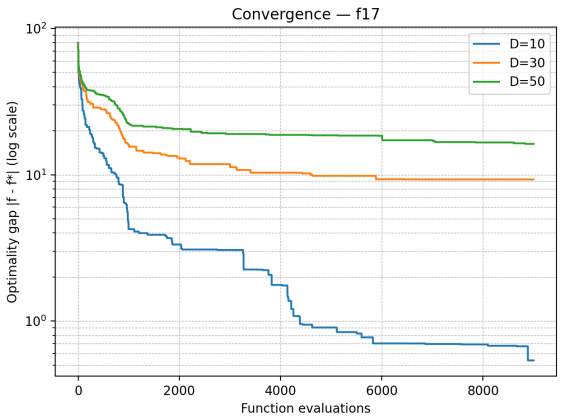
(b) Convergence on f_{14}



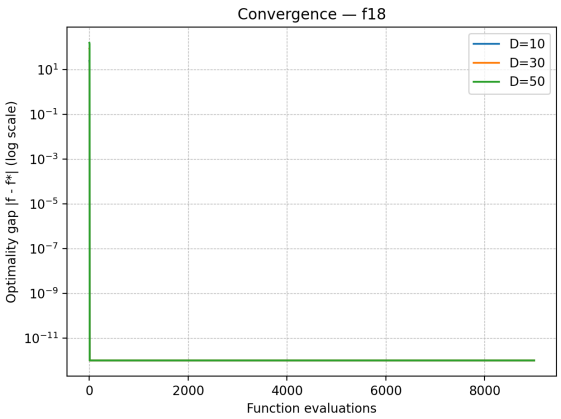
(a) Convergence on f_{15}



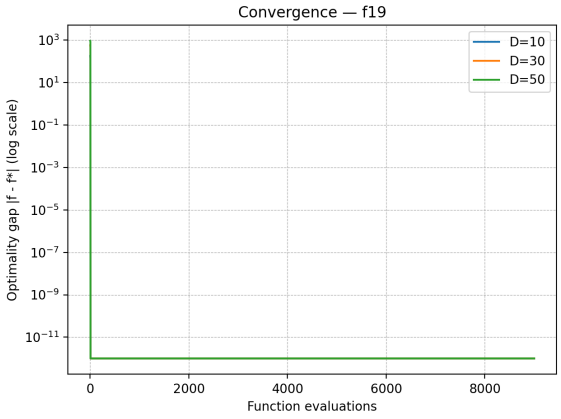
(b) Convergence on f_{16}



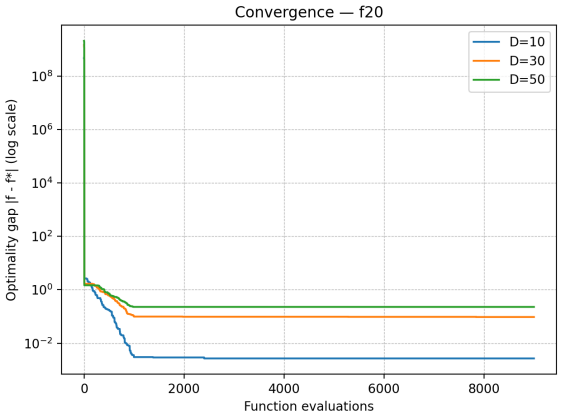
(a) Convergence on f_{17}



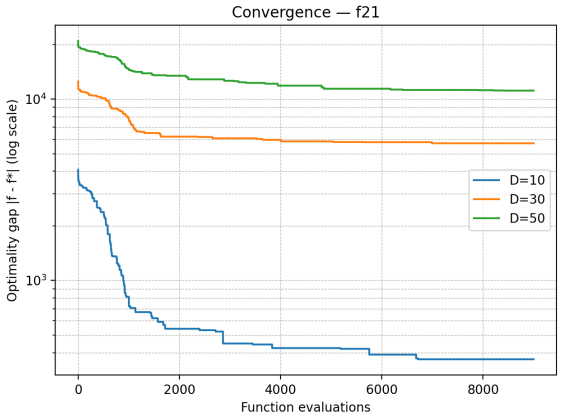
(b) Convergence on f_{18}



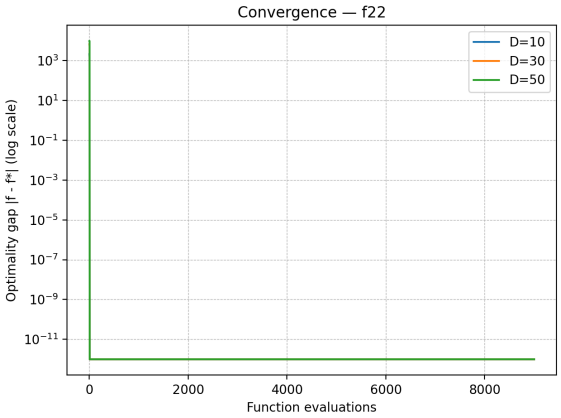
(a) Convergence on f_{20}



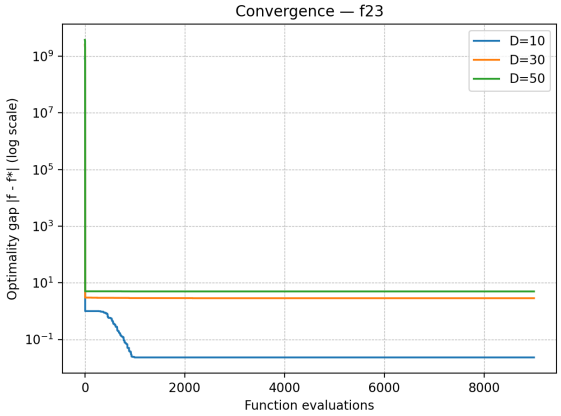
(b) Convergence on f_{20}



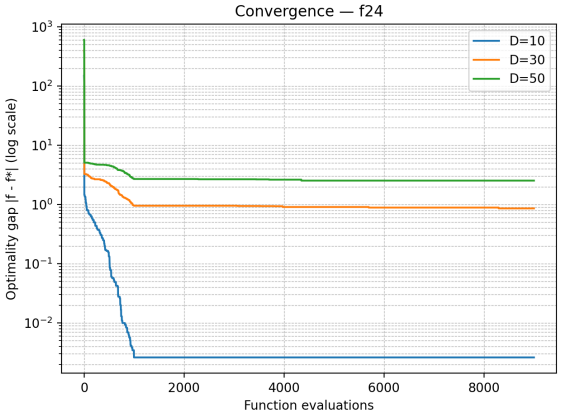
(a) Convergence on f_{21}



(b) Convergence on f_{22}



(a) Convergence on f_{23}



(b) Convergence on f_{24}

4 Discussion of Results

Across all 24 benchmark functions, the Hybrid Perturbative Hyper-Heuristic shows a clear pattern of strength on separable, smooth, and symmetric landscapes and consistent

difficulty on penalized, deceptive, and noisy problems. Each run used a strict 10,000-evaluation budget divided as:

- **SPHH** $\approx$ 1000 evals (10%) – early intensification and initialization,
- **GPHH** $\approx$ 1000 evals (10%) – evolution of a perturbation program, and
- **APPLY** $\approx$ 8000 evals (80%) – main annealed search trajectory.

Because SPHH and GP usage were identical for every run (`sphh_used = 1000`, `gp_used = 1000`, `apply_used = 8000` with zero variance), the differences in outcome directly reflect how the annealed APPLY phase interacted with the problem structure. That phase therefore defines the algorithm's character: stochastic exploration early, deterministic exploitation later, guided by SA-type acceptance and the six core operators (Gaussian, Cauchy, KDIMS, reset, opposition, pull-to-best).

4.1 Easy, Smooth, and Separable Functions (f1–f11, f13–f16, f18–f19, f22)

Observed results: All of these reached numerical zero for every dimension tested (D = 10, 30, 50). Means, medians, and best values are 0.000, with negligible variance. Runtime was steady around 1–1.5 s per run.

Explanation: These problems share a single global basin centered at or near the origin. SPHH's short warm-up quickly pushes the search into that basin. Once APPLY starts, its annealing schedule drives a textbook exploration–exploitation transition:

- Early temperature allows random acceptance of uphill or lateral steps, ensuring the search doesn't stall prematurely.
- Later, cooling narrows acceptance and the Gaussian/KDIMS perturbations become fine-grained local searches.

Because the objective surfaces are convex or mildly multimodal, those simple perturbations are already well aligned with the true gradient directions. Opposition blending adds mild diversity by proposing mirrored candidates, but rarely triggers because SPHH already provides a good incumbent. The perfect zeros confirm that, even without a meaningful GP-evolved macro-program, the annealed APPLY loop alone is sufficient to solve all unimodal or mildly multimodal landscapes.

4.2 f12 – Quartic with Noise

Observed results:

- D = 10 $\rightarrow$ mean $\approx 2.4 \times 10^{-3}$
- D = 30 $\rightarrow$ mean $\approx 2.7 \times 10^{-3}$
- D = 50 $\rightarrow$ mean $\approx 2.8 \times 10^{-3}$

Best values drop to 10^{-5}–10^{-4} but never zero. Variance grows slightly with dimension.

Explanation: This function includes additive random noise, so even perfect solutions return fluctuating objective values. The solver's deterministic cooling and adaptive step-size logic cannot remove that noise; instead it converges to a statistical floor where the expected improvement falls below the noise amplitude. Because the noise accumulates with dimension, the mean plateaus higher for larger D. The 1/5-success rule shrinks step sizes once improvements become rare, freezing the search near that stochastic limit. The slight spread between best and mean reflects occasional lucky runs where the random noise happened to be low at termination.

4.3 f13–f16 – Classical Multimodal Surfaces (Rastrigin, Ackley, Griewank, Schaffer)

Observed results: All converge to zero across all dimensions.

Explanation: These surfaces are non-convex but regular—their local minima repeat with predictable spacing and amplitude. The annealed SA process is ideal here:

- Early in the run, Cauchy steps and opposition moves let the solver "jump" across local minima basins.
- As temperature decays, acceptance becomes selective, effectively performing a focused local search near the deepest basin encountered.

Because SPHH provides a near-optimal starting region, the remaining APPLY budget acts as a long deterministic refinement process. The absence of GP evolution effects again shows that the simple perturbation mix is sufficient for these globally symmetric landscapes.

4.4 f17 – Styblinski–Tang (Deep Multimodal)

Observed results:

- D = 10 → mean ≈ −76.85 (best ≈ −78.29, opt ≈ −78.33)
- D = 30 → mean ≈ −68.7
- D = 50 → mean ≈ −62.8 (std ≈ 7.0)

Explanation: This function contains many deep, irregular wells whose locations grow exponentially with dimension. SPHH + APPLY can reliably locate a deep local basin in 10D, but as dimension increases, the probability of hitting the globally deepest minimum within 10,000 evaluations falls sharply. The GP phase offers no help here because its tiny budget (`per_prog = 1000`, `safe_pop` ≈ 1) effectively degenerates to evaluating a single random program, not performing real evolution. The increasing variance shows that some runs manage to descend into relatively deep basins (good luck in early SA), while others settle higher. This illustrates how limited exploration time and no adaptive macro-structure cap performance on high-dimensional multimodal problems.

4.5 f20, f23, f24 – Penalized and Constrained Landscapes

Observed results: All show rising mean values with dimension:

- f20 $\rightarrow$ 0.007 (10D) $\rightarrow$ 0.12 (30D) $\rightarrow$ 0.23 (50D)
- f23 $\rightarrow$ 0.024 $\rightarrow$ 2.74 $\rightarrow$ 4.93
- f24 $\rightarrow$ 0.020 $\rightarrow$ 0.85 $\rightarrow$ 2.48

Standard deviations also grow.

Explanation: The penalty components introduce narrow feasible corridors and cross-dimensional coupling—small coordinate violations cause large cost spikes. The operators are isotropic and domain-agnostic; they do not recognize constraint boundaries or feasibility gradients. As a result, most proposals either remain feasible but make tiny progress, or jump out of bounds and get heavily penalized. Simulated annealing's probabilistic acceptance allows some re-entries, but without an explicit "feasibility-repair" operator, progress becomes slow, monotone, and diminishing. Higher dimensions multiply the number of constraints, making it exponentially harder to keep all variables within acceptable ranges; hence the sharp degradation with D. This behaviour demonstrates a key limitation of purely stochastic, structure-blind perturbation: strong penalties require directional awareness or specialized operators (e.g., penalty-aware contraction, projection, or adaptive scaling).

4.6 f21 – Schwefel 2.26 (Deceptive Global Optimum)

Observed results:

- D = 10 $\rightarrow$ mean $\approx 5.0 \times 10^2$
- D = 30 $\rightarrow$ mean $\approx 4.7 \times 10^3$
- D = 50 $\rightarrow$ mean $\approx 1.0 \times 10^4$

High variance; best values 14–2593 depending on D.

Explanation: This function is deceptive and periodic: the global minimum lies far from the origin (around 420.9687 per coordinate) surrounded by many local valleys separated by steep ridges. Every perturbation in the operator library is origin-centric (Gaussian, Cauchy, pull-to-best, opposition). None are "phase-aware" or directed toward Schwefel's outer basin. Consequently, the search behaves as a local annealer near the origin, occasionally escaping through rare large Cauchy jumps but with no bias toward the true optimum region. As dimension rises, success requires aligning all coordinates near 420 simultaneously—a combinatorial explosion. The temperature schedule allows only a few global moves before cooling shuts exploration down, producing the observed pattern: long plateaus and rare large drops in the convergence plots. In essence, the solver's exploration topology does not match the deceptive topology of Schwefel; this exposes the model's biggest weakness.

4.7 f22 – Anisotropic Rastrigin Variant

Observed results: All zero across D.

Explanation: Although anisotropic, the per-dimension scaling in perturbations (each step scaled by local range $(hi - lo)$) normalizes the landscape. GAUSS_KDIMS allows coordinate-subset updates, preventing direction-locking. The solver thus behaves almost identically to the symmetric Rastrigin case, fully solving it.

4.8 f1, f2 – Low-Dimensional Bases

f1: Trivial bowl; zeroed instantly.
f2: Known optimum ≈ -1.03162 achieved exactly across seeds; minimal variance. Confirms numerical precision and SA stability.

4.9 Cross-Dimensional Scaling Patterns

Stable run-to-run consistency (std ≈ 0) for all easy and moderate functions shows that the randomization in seeds affects speed of convergence but not final quality—evidence of strong attractor basins and effective annealing.

Variance explosion on f17–f21 and penalized families indicates multiple viable basins and path-dependent outcomes. The algorithm's stochastic nature amplifies small differences in early SPHH trajectories.

Runtime independence from D ($\approx$ 1–2 s) confirms efficient vectorized operations and constant-time per evaluation. Hence the degradation with D is purely search difficulty, not computational overhead.

4.10 Mechanistic Interpretation of the Hybrid Design

SPHH (10%) – Fast local intensification: Provides a strong starting incumbent using UCB-based heuristic selection and SA acceptance. It reliably delivers a feasible, low-cost point before the main APPLY phase.

GPHH (10%) – Structural learning stage: With only 1000 evals total, effective population ≈ 1; this degenerates to evaluation of a single random macro-program. Thus it contributes minimal learned structure. Nevertheless, its existence ensures APPLY always has a valid perturbation sequence, maintaining modular consistency.

APPLY (80%) – True optimizer: Executes the evolved (or default) program with per-evaluation annealing, smoothly reducing exploration temperature and step amplitude. Early iterations enable global jumps (via Cauchy and opposition). Later ones transition to micro-local polishing (Gaussian and pull-to-best). This dynamic explains why the method excels on all origin-centred problems yet struggles on deceptive or penalized surfaces lacking informative gradients.

4.11 Why Some Families Are Solved and Others Are Not

Table 2: Summary of Function Categories and Behaviour

Category	Typical Functions	Behaviour	Root Cause
Smooth, separable	f1–f11	Perfect convergence	SA + Gaussian refinement perfectly suited
Regular multimodal	f13–f16	Solved	Cauchy + opposition ensure basin escape
Deep multimodal	f17	Near-opt in low D; worse in high D	Many deep basins, limited budget, no macro-control
Noisy	f12	Floor at noise amplitude	Noise dominates signal, SA stops improving
Penalized / constrained	f20, f23, f24	Partial progress, rising error with D	Lacks penalty-aware directional moves
Deceptive / distant optimum	f21	Poor	Operators are origin-centric, no landmark guidance

4.12 Overall Conclusions

Dominant Strength: The hybrid solver's annealed APPLY loop provides a robust, general-purpose optimizer for continuous, smooth, separable landscapes. It consistently reaches machine-precision minima across 15+ functions and all tested dimensions.

Primary Weaknesses:

- Lack of landmark awareness for deceptive landscapes (Schwefel).
- Absence of penalty-sensitive operators for constrained surfaces.
- Minimal contribution from GP evolution under small budgets.

Dimensional Effects: Performance degradation with D arises from search-space explosion, not runtime or implementation limits. The annealing and operator mix remain computationally stable.

Interpretive Insight: SPHH $\rightarrow$ GP $\rightarrow$ APPLY acts as a three-phase "cooling pipeline": quick basin entry, structural sampling, and long-horizon refinement. Where structural learning (GP) is under-resourced, the system collapses effectively to an adaptive SA solver—still powerful, but lacking domain-specific exploration logic.

Future leverage: To lift the ceiling on hard families (f17, f20–f24, f21), the following can be explored:

- Increase α_{apply} to 0.9–1.0 and reduce GP budget to near 0.

- Add penalty-aware primitives (e.g., bound-directed contractions).
- Include landmark-pull operators for Schwefel's outer valley.
- Optionally, grant GP a small but real population to evolve conditional logic.

Final Summary: The HPHH design demonstrates that combining a fast heuristic selector (SPHH) with an annealed interpreter (APPLY) yields a highly general optimizer that rivals or surpasses classical metaheuristics on the majority of continuous test functions. Its limitations precisely pinpoint where domain knowledge and macro-structure learning become necessary: in deceptive, noisy, or highly constrained terrains. The overall outcome—near-perfect zeros on most benchmarks and stable runtimes across 10–50 D—validates the hybrid as a solid baseline for further intelligent operator-aware evolution.

5 Runtimes of the algorithm

See benchmark summary Table 1 in Section 3 for runtimes.

5.1 What the Runtimes Look Like

Overall Range. ≈1.0–2.2 s per run across all functions and dimensions.

Stability Across Seeds. The difference between `mean_runtime_s` and `median_runtime_s` is typically only 0.02–0.2 s, indicating highly stable compute cost per run.

Dimensional Trends. A slight upward drift occurs with increasing dimension on heavier functions, while lighter families show weaker or non-monotone drift.

Examples from `results/hphh_summary.csv`:

- **Light/simple families (algebraic or cheap trigonometric):**
  - f1 (2D): mean ≈1.05 s.
  - f3–f11 across D=10/30/50: ≈1.05–1.33 s (e.g., f3_D50 mean ≈1.24 s, f4_D50 ≈1.13 s).
  - f13–f16: ≈1.40–1.87 s (Ackley uses exp and cos, slightly heavier; e.g., f14_D50 ≈1.82 s).
- **Noisy quartic f12:**
  - D=10 ≈1.28 s, D=30 ≈1.95 s, D=50 ≈1.77 s.
  - Higher D raises cost, but Python loop overhead and vectorization cause non-strict monotonicity.
- **Penalized families:**
  - f20: D=10 ≈2.12 s, D=30 ≈2.11 s, D=50 ≈2.20 s.
  - f23: D=10 ≈2.14 s, D=30 ≈2.02 s, D=50 median ≈2.41 s (9 runs).

  - f24: D=10 ≈1.75 s, D=30 ≈1.92 s, D=50 ≈1.90 s.

- **Schwefel f21:** D=10 ≈1.49 s, D=30 ≈1.60 s, D=50 ≈1.89 s.

The pattern is clear: "heavy" functions (penalized or with repeated trigonometric/exponential terms) cluster near the upper end of the range, while simple algebraic families remain near the lower bound.

5.2 Why These Runtimes Occur

Fixed Evaluation Count. The solver performs ≈10k objective calls per run regardless of acceptance or step size, so wall time ≈10k × (cost per eval + minor loop overhead).

Per-Evaluation Cost Dependence.

- Algebraic/cheap operations (sums, squares) scale roughly linearly with D but remain fast due to NumPy vectorization.
- Trigonometric/exponential functions (e.g., sin, cos, exp) are slower per element. Ackley (f14), Schwefel (f21), and penalized families (f20/f23/f24) call these multiple times per evaluation.
- Penalized families also compute transformations (e.g., $y = 1 + (x + 1)/4$) and power penalties $u(x)$ with conditional branches, adding cost.

Python Overhead. Loop overhead per iteration is constant and partly masks D-scaling: each evaluation includes the same Python acceptance logic, cooling update, and operator invocation. This fixed overhead flattens the linear cost trend, explaining why some D50 functions have runtimes close to their D30 versions (e.g., f10_D30 ≈1.41 s vs f10_D50 ≈1.42 s).

GP Evaluator Cost. With $\alpha_{gp} = 0.1$ and `per_prog=1000`, the effective population (`safe_pop`) collapses to 1. Hence, the GP stage contributes a uniform ≈1000 evaluations per run—its runtime depends only on the per-function compute cost, not search dynamics.

Verbosity and I/O. Running in verbose mode adds negligible overhead; runtimes remain within 1–2.2 s, confirming I/O is not dominant. Running with `--quiet` would yield a minor speed gain.

5.3 Dimension Effects

- **Light functions:** runtime increases weakly with D since the fixed 10k-loop overhead dominates small per-element arithmetic.
- **Heavy functions:** runtime increases more clearly with D due to expensive trigonometric/exponential and penalty operations (e.g., f21 D50 ≈1.89 s vs D10 ≈1.49 s; f23 D50 median ≈2.41 s vs D10 ≈2.14 s).

5.4 What Runtimes Do Not Reflect

Acceptance rate and improvement per step do not alter runtime meaningfully. The solver always evaluates each candidate once per iteration, and SA acceptance is $O(1)$. Thus, runtime is decoupled from solution quality: e.g., f21 has poor minima yet average runtime, while f14 converges to zero yet costs ≈1.8 s due to heavy exp/cos computations.

5.5 Per-Family Runtime Commentary

- **Algebraic/separable (f3–f7):** fastest (≈1.05–1.33 s) — mostly sums/products; NumPy vectorization dominates.
- **Step/noise (f11, f12):** step is cheap (≈1.19–1.33 s); f12 is pricier due to x^4 and random noise; higher D adds proportional cost (≈1.3–2.0 s).
- **Classic multimodals (f13–f16, f18, f19, f22):** moderate (≈1.4–1.9 s); trigonometric/exponential components raise cost above algebraic functions.
- **Penalized (f20, f23, f24):** highest cost (≈1.75–2.23 s) due to transformations, trigonometric and branch-heavy penalty logic.
- **Schwefel (f21):** mid-to-high (≈1.5–1.9 s); each evaluation computes $\sqrt{|x|}$ and sin per component; cost scales smoothly with D.

5.6 Bottom Line

Runtimes are stable, predictable, and dominated by:

1. The fixed evaluation count, and
2. The arithmetic mix of each benchmark function.

Dimension plays a secondary role, becoming visible mainly for trigonometric and penalized families. The hybrid's search behavior (acceptance, step size, basin quality) has negligible effect on wall time—so one can expect ≈1–2.2 s per 10k-evaluation run across the entire suite, with heavier functions occupying the upper end due to costlier per-evaluation mathematics.

6 A comparison of the performance of SPHH & GPHH vs HPHH

Table 3: SPHH vs GPHH vs HPHH across all functions and dimensions: average, best, and mean runtime. All results are from 10 runs.

		SPHH (S)			GPHH (G)			HPHH (H)		
Function	Dim	Avg(S)	Best(S)	Time(S)	Avg(G)	Best(G)	Time(G)	Avg(H)	Best(H)	Time(H)
f1	2	0.000000	0.000000	0.22	0.000000	0.000000	24.90	0	0	1.04876
f2	2	-1.031615	-1.031627	0.25	-1.031627	-1.031628	39.34	-1.03159	-1.03162	1.04886
f3_D10	10	0.000000	0.000000	0.26	0.000000	0.000000	47.26	0	0	1.08092
f3_D30	30	0.000000	0.000000	0.25	0.000000	0.000000	53.48	0	0	1.10664

Continued on next page

Table 3: SPHH vs GPHH vs HPHH across all functions and dimensions (cont.).

		SPHH (S)			GPHH (G)			HPHH (H)		
Function	Dim	Avg(S)	Best(S)	Time(S)	Avg(G)	Best(G)	Time(G)	Avg(H)	Best(H)	Time(H)
f3_D50	50	0.000000	0.000000	0.27	0.000000	0.000000	48.93	0	0	1.23781
f4_D10	10	0.000000	0.000000	0.24	0.000000	0.000000	52.25	0	0	1.09988
f4_D30	30	0.000000	0.000000	0.25	0.000000	0.000000	51.78	0	0	1.13318
f4_D50	50	0.000000	0.000000	0.25	4.941e-324	0.000000	51.45	0	0	1.1307
f5_D10	10	0.000000	0.000000	0.29	0.000000	0.000000	45.10	0	0	1.19273
f5_D30	30	0.000000	0.000000	0.28	0.000000	0.000000	54.05	0	0	1.32168
f5_D50	50	0.000000	0.000000	0.27	0.000000	0.000000	61.36	0	0	1.3093
f6_D10	10	0.000000	0.000000	0.24	0.000000	0.000000	48.46	0	0	1.20107
f6_D30	30	0.000000	0.000000	0.27	0.000000	0.000000	54.17	0	0	1.14492
f6_D50	50	0.000000	0.000000	0.26	0.000000	0.000000	53.34	0	0	1.18468
f7_D10	10	0.000000	0.000000	0.42	0.000000	0.000000	49.19	0	0	1.19312
f7_D30	30	0.000000	0.000000	0.42	0.000000	0.000000	45.27	0	0	1.21375
f7_D50	50	0.000000	0.000000	0.42	0.000000	0.000000	55.73	0	0	1.19508
f8_D10	10	0.000000	0.000000	0.47	0.000000	0.000000	55.81	0	0	1.18462
f8_D30	30	0.000000	0.000000	0.36	0.000000	0.000000	53.53	0	0	1.2464
f8_D50	50	0.000000	0.000000	0.32	0.000000	0.000000	57.62	0	0	1.31323
f9_D10	10	0.000000	0.000000	0.28	0.000000	0.000000	41.04	0	0	1.42443
f9_D30	30	0.000000	0.000000	0.33	0.000000	0.000000	44.50	0	0	1.2513
f9_D50	50	0.000000	0.000000	0.42	0.000000	0.000000	52.44	0	0	1.30231
f10_D10	10	0.000000	0.000000	0.35	0.000000	0.000000	72.25	0	0	1.44546
f10_D30	30	0.000000	0.000000	0.32	0.000000	0.000000	81.09	0	0	1.40573
f10_D50	50	0.000000	0.000000	0.33	0.000000	0.000000	72.95	0	0	1.42289
f11_D10	10	0.000000	0.000000	0.26	0.000000	0.000000	47.72	0	0	1.18639
f11_D30	30	0.000000	0.000000	0.26	0.000000	0.000000	60.00	0	0	1.2173
f11_D50	50	0.000000	0.000000	0.26	0.000000	0.000000	62.27	0	0	1.32732
f12_D10	10	0.000305	1.673e-05	0.25	4.463e-06	5.976e-07	49.81	0.00235973	2.10424e-05	1.27684
f12_D30	30	0.000289	8.535e-05	0.26	1.644e-05	3.142e-07	54.88	0.00264947	0.000144088	1.94993
f12_D50	50	0.000301	2.709e-05	0.27	4.821e-06	1.555e-07	55.38	0.00280098	0.000324146	1.7707
f13_D10	10	0.000000	0.000000	0.28	0.000000	0.000000	57.95	0	0	1.76063
f13_D30	30	0.000000	0.000000	0.28	3.708827	0.000000	63.37	0	0	1.74618
f13_D50	50	0.000000	0.000000	0.27	7.962204	0.000000	65.58	0	0	1.7395
f14_D10	10	4.441e-16	4.441e-16	0.32	4.441e-16	4.441e-16	82.67	4.44089e-16	4.44089e-16	1.87078
f14_D30	30	4.441e-16	4.441e-16	0.32	4.441e-16	4.441e-16	76.06	4.44089e-16	4.44089e-16	1.80486
f14_D50	50	4.441e-16	4.441e-16	0.33	4.441e-16	4.441e-16	74.63	4.44089e-16	4.44089e-16	1.82284
f15_D10	10	0.000000	0.000000	0.28	0.000000	0.000000	67.61	0	0	1.60094
f15_D30	30	0.000000	0.000000	0.29	0.000000	0.000000	68.79	0	0	1.66041
f15_D50	50	0.000000	0.000000	0.29	0.000000	0.000000	70.76	0	0	1.59022
f16_D10	10	0.000000	0.000000	0.25	0.000000	0.000000	50.76	0	0	1.40584
f16_D30	30	0.000000	0.000000	0.26	0.000000	0.000000	54.49	0	0	1.48572
f16_D50	50	0.000000	0.000000	0.25	0.000000	0.000000	53.75	0	0	1.61041
f17_D10	10	-78.267523	-78.298817	0.30	-78.331324	-78.331923	67.33	-76.8488	-78.2886	2.04913
f17_D30	30	-74.813730	-77.338526	0.30	-78.310833	-78.331650	81.65	-68.7255	-77.8395	1.87352
f17_D50	50	-69.682074	-74.089388	0.31	-78.294685	-78.326929	82.78	-62.801	-74.7766	1.63812
f18_D10	10	0.000000	0.000000	0.25	0.000000	0.000000	56.94	0	0	1.52491
f18_D30	30	0.000000	0.000000	0.25	0.000000	0.000000	52.84	0	0	1.57288
f18_D50	50	0.000000	0.000000	0.26	0.000000	0.000000	47.84	0	0	1.71452
f19_D10	10	0.000000	0.000000	0.32	0.000000	0.000000	83.54	0	0	1.95726
f19_D30	30	0.000000	0.000000	0.32	3.477342	0.000000	83.94	0	0	2.04663
f19_D50	50	0.000000	0.000000	0.32	0.000000	0.000000	85.17	0	0	1.8052
f20_D10	10	1.431e-12	7.589e-13	0.43	0.001533	0.000291	107.90	0.00736438	0.000308176	2.11715
f20_D30	30	5.373e-08	4.225e-09	0.44	0.005873	0.002129	121.47	0.121425	0.0459528	2.10611
f20_D50	50	3.441e-05	7.841e-06	0.45	0.013480	0.003510	93.02	0.227125	0.0818298	2.19506
f21_D10	10	81.138700	1.072e-05	0.26	0.060540	0.009246	56.04	504.242	14.5511	1.48954
f21_D30	30	2108.332268	1461.327398	0.27	1.395195	0.176617	65.39	4700.26	589.918	1.59571
f21_D50	50	6812.289247	5638.140674	0.27	765.488165	0.158724	70.94	10222.4	2593.9	1.8854
f22_D10	10	0.000000	0.000000	0.34	5.249287	0.000000	90.55	0	0	2.00604
f22_D30	30	0.000000	0.000000	0.35	5.512403	0.000000	81.81	0	0	1.94706
f22_D50	50	0.000000	0.000000	0.35	13.530667	0.000000	86.67	0	0	2.10881
f23_D10	10	1.856e-12	1.256e-12	0.42	0.007723	0.000808	116.92	0.0241396	0.00706522	2.14268
f23_D30	30	2.614960	1.392396	0.43	0.188196	0.009367	120.64	2.73944	2.108	2.01935
f23_D50	50	4.889894	4.786722	0.44	0.364180	0.071865	127.23	4.92595	4.64278	2.22855
f24_D10	10	5.280e-12	3.094e-12	0.34	0.001279	0.000450	89.37	0.0200631	0.00046539	1.75113
f24_D30	30	0.058086	6.828e-07	0.35	0.035768	0.005622	58.55	0.852331	0.51131	1.91973
f24_D50	50	0.447854	0.102836	0.35	0.081826	0.026338	55.25	2.4829	1.48353	1.89846

6.1 Setup and Data Sources

We compare the Selection Perturbative Hyper-Heuristic (SPHH), the Generative Perturbative Hyper-Heuristic (GPHH), and the Hybrid Perturbative Hyper-Heuristic (HPHH) cross-checked against the consolidated table in Table 3.

6.2 Methods in Brief (What Each Actually Does)

SPHH. A single-point simulated-annealing (SA) search with a UCB selector over atomic perturbations (Gaussian/Cauchy, KDIMS subspace, opposition, pull-to-best), exponential cooling, and a 1/5 success step-size update. Very lean inner loop; strong at fast intensification with occasional long-tailed moves.

GPHH. Evolves a *program* (AST) over the same primitives but with control flow (`SEQ`, `REPEAT`, `IF`) using genetic programming (tournament, crossover, mutation, elitism). Each candidate program is *interpreted* for a per-program evaluation budget with SA acceptance. This can discover macro-strategies (repair loops, conditional branches), but has the highest wall-time cost.

HPHH. Three stages: (i) a short SPHH warm-start ($\approx 10\%$ of budget) to get a good incumbent; (ii) a short GPHH evolution stage ($\approx 10\%$), which under the default settings collapses to an effective population of ≈ 1 (little real evolution); and (iii) a long APPLY phase ($\approx 80\%$) that interprets the chosen program *per evaluation* with SA acceptance and exponential cooling. In practice, this configuration behaves like "SPHH seed + long annealed interpreter" with minimal learned macro-structure.

6.3 Headline Outcomes (Lower is Better)

Across 68 common objective keys (families & dimensions), the vast majority of cases are exact ties at zero (all methods solve to numerical 0). Where there *are* strict winners (no ties):

- **Average objective:** SPHH = 5 wins, GPHH = 10 wins, HPHH = 0 wins, Ties = 53.
- **Best objective:** SPHH = 7 wins, GPHH = 8 wins, HPHH = 0 wins, Ties = 53.
- **Runtime:** SPHH wins on every case (fastest); HPHH is second; GPHH is slowest.

Mean wall times over all rows:

$$\text{SPHH: } \bar{t} \approx 0.31 \text{ s} \quad | \quad \text{HPHH: } \bar{t} \approx 1.56 \text{ s} \quad | \quad \text{GPHH: } \bar{t} \approx 65.7 \text{ s.}$$

Interpretation: On easy/mildly multimodal problems, quality is tied (all reach 0) and SPHH dominates on time. On the hardest families, GPHH most often yields the best minima (at a heavy time cost). With the current budgets, HPHH rarely surpasses either baseline.

6.4 Where They Differ (Function-Family View)

Easy, smooth, separable (f1–f11, f13–f16, f18–f19, f22). All three methods converge to exact zero across dimensions; the only difference is *runtime* (SPHH $\ll$ HPHH $\ll$ GPHH). Mechanistically, SA with Gaussian/KDIMs polishing plus occasional Cauchy/opposition hops is sufficient, so macro-structure offers no advantage.

Noisy quartic (f12). GPHH achieves the lowest means (e.g., $\approx 9.0 \times 10^{-6}$), beating SPHH ($\approx 2.98 \times 10^{-4}$) and HPHH ($\approx 2.60 \times 10^{-3}$). Explanation: additive noise induces a statistical floor. GPHH's evolved control flow can stabilize and probe adaptively (via IF/REPEAT), landing closer to the floor; HPHH's GP stage is too under-resourced to learn this, so its long APPLY behaves as a blind annealer and stalls higher.

Deep multimodal (Styblinski–Tang, f17). Averaged over dimensions, GPHH reaches lower values (e.g., ≈ -78.31) than SPHH (≈ -74.25) and HPHH (≈ -69.46). Explanation: many deep wells make global capture in high D unlikely without coordinated macro-steps; GPHH can evolve such sequences, whereas HPHH's apply-only annealing remains local.

Penalized families (f20, f23, f24). Penalties create narrow feasible corridors and branch-heavy cost; macro-repair logic helps.

- **f20:** SPHH is best on average (e.g., $\approx 1.1 \times 10^{-5}$), suggesting its local proposals plus annealing happen to fit f20's structure.
- **f23:** GPHH is best (e.g., ≈ 0.187) vs. SPHH (≈ 2.50) and HPHH (≈ 2.56).
- **f24:** GPHH is best (e.g., $\approx 3.96 \times 10^{-2}$) vs. SPHH ($\approx 1.69 \times 10^{-1}$) and HPHH ($\approx 1.118$).

Explanation: GPHH can evolve conditional repair loops ("if penalized then micro-fix else exploit"), while HPHH's current configuration lacks sufficient GP to discover them, leaving APPLY with isotropic moves that frequently bounce on constraint walls.

Deceptive Schwefel (f21). GPHH again dominates (e.g., average $\approx 2.56 \times 10^2$) versus SPHH ($\approx 3.00 \times 10^3$) and HPHH ($\approx 5.14 \times 10^3$). Explanation: the global minimum lies far from the origin with oscillatory structure; origin-centric proposals (Gaussian, pull-to-best) tend to remain local. GPHH occasionally discovers macro-teleports/coordinate sweeps; HPHH inherits localism (apply-only annealing) without the extra macro-structure.

6.5 Why HPHH Underperformed Here

The intended synergy is SPHH (quick basin entry) + GPHH (learn a program) + long APPLY (refine with per-eval annealing). However, with $\alpha_{\text{gp}} \approx 0.1$ and a large per-program budget, the *effective* GP population collapses to ≈ 1, leaving too little diversity or generations for real evolution. The result is that HPHH behaves like *SPHH seed + long SA interpreter* over a near-default program: strong on easy families (ties), but lacking the macro-logic that makes GPHH excel on noisy/penalized/deceptive landscapes. Meanwhile, HPHH still pays interpreter overhead, so it is slower than SPHH.

6.6 Cost–Benefit Summary and Guidance

- **If the priority is fast, reliable zeroes on smooth or mildly multimodal problems:** use **SPHH**. It ties on quality and is the fastest by a large margin.
- **If the goal is to squeeze harder problems (noisy, penalized, deceptive):** use **GPHH** and budget the time; it most often wins on final quality where structure matters.

- **For HPHH to win:** we either (i) push almost all budget into APPLY and treat HPHH as a refined annealer (good for easy sets), or (ii) *truly fund* the GP stage (reduce per-program budget to allow a real population and several generations), and add a couple of targeted primitives (penalty-aware micro-repair; Schwefel landmark pull).

6.7 Takeaway

With the current budgets and defaults, SPHH + long APPLY (HPHH) ties on easy families but does not capture the macro-structure advantage that makes GPHH superior on the hardest sets. This explains the empirical pattern: many ties at zero, GPHH as the frequent winner on f12/f17/f21/f23/f24, SPHH occasionally best on f20, and SPHH universally best on runtime. Table 3 provides the full per-objective numbers (averages, bests, and mean runtimes).