HOPSO: Hybrid Particle-Swarm Optimizer

Variational Quantum Algorithms · Module 6 — Optimization Dynamics

Node 6.8

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.8

This is the **closing node** of Module 6 and the most thesis-specific.

HOPSO — **Hybrid Optimizer Particle Swarm Optimization** — is the optimizer the thesis uses as its primary case study for the regularization-as-dynamical-modification framework. It is a hybrid of:

1. **Population-based exploration** (from PSO, Section 6.7), giving it the ability to escape local minima and barren plateau regions through population diversity.
2. **Per-particle gradient descent** (from parameter-shift, Section 6.1), giving each particle the ability to refine its position via gradient information rather than relying purely on swarm dynamics.

The result is an optimizer that combines the global-exploration benefit of populations with the local-precision benefit of gradients. It is the thesis's preferred answer to "what optimizer should you use for a difficult VQA," with the understanding that "difficult" means "barren-plateau-prone, multi-modal, or both."

This node walks through HOPSO's design, why each component is justified, and how it embodies the regularization-as-dynamics-modification thesis.

The HOPSO Update Rule

Each particle in the swarm has a position $\boldsymbol{\theta}_i^k$ and a velocity $\mathbf{v}_i^k$. At each iteration:

Step 1: Local gradient. Each particle computes its local cost gradient $\widehat{\nabla C}(\boldsymbol{\theta}_i^k)$ via parameter-shift, with whatever shot budget is allocated.

Step 2: Velocity update. The velocity is updated by combining gradient descent with PSO swarm terms:

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_g(-\widehat{\nabla C}(\boldsymbol{\theta}_i^k)) + c_p r_p(\mathbf{p}_i - \boldsymbol{\theta}_i^k) + c_s r_s(\mathbf{g} - \boldsymbol{\theta}_i^k),$$

where: - w is the inertia (typically 0.5-0.9). - c_g is the gradient coefficient (the "gradient pull"). - c_p is the personal-best coefficient. - c_s is the swarm-best (global) coefficient. - r_p, r_s are uniform random numbers in $[0, 1]$. - $\mathbf{p}_i$ is particle i's personal-best position so far. - $\mathbf{g}$ is the swarm's global-best position so far.

Step 3: Position update. $\boldsymbol{\theta}_i^{k+1} = \boldsymbol{\theta}_i^k + \mathbf{v}_i^{k+1}$.

Step 4: Personal/global best update. If the new position has a better cost than $\mathbf{p}_i$, update $\mathbf{p}_i$. If better than $\mathbf{g}$, update $\mathbf{g}$.

Step 5: Diversity check. If the swarm has collapsed (all particles within some radius), inject mutation or restart some particles.

What Each Term Does

The four velocity terms each play a distinct role.

1. Inertia term $w\mathbf{v}_i^k$. Maintains the previous velocity, smoothing the trajectory. Like momentum in classical SGD.

2. Gradient term $c_g(-\widehat{\nabla C})$. Pulls each particle in its local descent direction. This is the "particle does GD" component. In a region where the gradient is reliable (above noise floor, no plateau), this makes the particle converge quickly.

3. Personal-best term $c_p r_p(\mathbf{p}_i - \boldsymbol{\theta}_i^k)$. Pulls the particle back toward its own best historical position. This term is *self-correcting*: if a particle wanders into a bad region, it gets pulled back to its best known location.

4. Global-best term $c_s r_s(\mathbf{g} - \boldsymbol{\theta}_i^k)$. Pulls the particle toward the swarm's global best. This is the *social* component, which lets information flow between particles. The swarm collectively concentrates on the best region found so far.

The mix of gradient (local information) and PSO terms (population information) is what makes HOPSO a *hybrid*.

Why HOPSO Works In The Plateau Regime

The key claim of the thesis is that HOPSO can find solutions in regions where vanilla gradient descent fails because of barren plateaus.

Mechanism 1: Population diversity beats single-point starting. A vanilla GD optimizer with random initialization has a probability of landing in a non-plateau region that decays exponentially with qubit count. A swarm of N particles has a probability $\sim N \times 2^{-n}$ of having at least one particle land in a non-plateau region — still exponentially small, but multiplied by N. For $N = 100$, this is a 100x improvement, which can be the difference between "no signal" and "some signal."

Mechanism 2: Particles in the plateau follow the leader. Once one particle finds a non-plateau region (via the global-best pull, or by random PSO motion), the rest of the swarm is pulled toward it. The plateau-trapped particles don't need to find the signal themselves — they just need to follow the leader.

Mechanism 3: Gradient terms become useful again outside the plateau. Once a particle is outside the plateau, its local gradient is meaningful and the gradient term in HOPSO accelerates convergence. The PSO terms get the particle out of the plateau; the gradient term takes it from "in a non-plateau region" to "at a minimum."

Mechanism 4: Diversity preservation. HOPSO's diversity-check step prevents the swarm from collapsing entirely, which would destroy the population benefit. By keeping some spread, the optimizer continues to explore even after concentrating most of its mass.

This is the core thesis claim: **plateau escape via population diversity, plateau exploitation via local gradients**.

HOPSO And The Regularization-As-Dynamics Thesis

HOPSO's connection to the thesis runs deeper than "it works on plateaus."

The thesis distinguishes: - **Objective-side regularization:** modify the cost function (add penalty terms). - **Dynamics-side regularization:** modify the optimizer dynamics (don't touch the cost).

HOPSO is **purely dynamics-side**. The cost function is unchanged. The regularization happens through the optimizer's own structure: the population maintains diversity, the swarm dynamics encourage exploration, the gradient terms encourage exploitation. There is no added term in $C(\boldsymbol{\theta})$.

This is operationally meaningful because:

1. **You don't need to know the right penalty term.** Adding a penalty to C requires designing the penalty, choosing its weight, tuning its hyperparameters. HOPSO has its own hyperparameters (w, c_g, c_p, c_s) but they parameterize the optimizer's behavior, not the problem.
2. **You don't change the global minimum.** A penalty in C shifts the minimum (intentionally — that's its purpose). HOPSO leaves the minimum where it is. If the optimizer converges, it converges to the *original* minimum, not a regularized one.
3. **The regularization is adaptive.** The swarm's exploration vs exploitation balance shifts over time as the population concentrates. This is a natural form of annealing without requiring an explicit schedule.
4. **The framework is composable.** You can add objective-side regularization on top of HOPSO if you want both. The two are not mutually exclusive — they are different layers of the optimization stack.

The thesis argues that **dynamics-side regularization deserves equal billing with objective-side**, and HOPSO is a clean example of why: it solves real problems (barren plateau escape) without requiring any modification to the cost function.

Hyperparameters And Tuning

HOPSO has more hyperparameters than vanilla GD or even Adam. The main ones:

- N: population size. Typical $20 - 100$.
- w: inertia. Typical $0.5 - 0.9$.
- c_g: gradient coefficient. Typical $0.1 - 1.0$ (depends on landscape gradient scale).
- c_p: personal best coefficient. Typical $0.5 - 2.0$.
- c_s: global best coefficient. Typical $0.5 - 2.0$.
- Restart threshold: when the swarm has collapsed below this radius, inject mutations.
- Shot budget per particle per iteration.

This is more knobs than most VQA optimizers, which is a real cost. But the defaults work reasonably well across a wide range of problems, and tuning is rarely needed for casual use.

A practical default: $N = 50$, $w = 0.7$, $c_g = 0.3$, $c_p = 1.0$, $c_s = 1.0$, restart threshold 0.1, shot budget per particle = same as vanilla GD baseline.

When HOPSO Is And Isn't The Right Choice

HOPSO is a good choice for:

- **Difficult VQAs with barren plateaus** that vanilla GD can't escape.
- **Multi-modal landscapes** where the optimizer needs to explore multiple basins.
- **Noisy hardware** where the population averaging helps stabilize the trajectory.
- **Problems with adequate compute** to support a large population per iteration.

HOPSO is **not** a good choice for:

- **Simple smooth VQAs** where vanilla GD or Adam works fine. The population overhead is wasted.
- **Very high-dimensional problems** ($P > 200$) where the population can't densely sample parameter space anyway.
- **Time-critical experiments** where the per-iteration cost is prohibitive.
- **Problems with very expensive cost evaluations** (e.g., real hardware with limited time slots) where shooting your shot budget across 50 particles is too costly.

For "small-easy" problems, use Adam. For "big-hard" problems, use HOPSO. For everything in between, it depends on your shot budget and your patience.

Empirical Performance Of HOPSO

The thesis reports empirical comparisons of HOPSO against vanilla GD, Adam, natural gradient, and other baselines on a suite of VQA problems. The summary findings (paraphrased):

1. **On smooth, plateau-free problems** ($n \leq 8$ qubits, shallow ansatz), HOPSO is comparable to or slightly slower than Adam in total cost. The population overhead is wasted on easy problems.
2. **On medium-difficulty problems** ($n = 10 - 15$, moderate plateau risk), HOPSO is competitive with Adam and significantly more reliable — it succeeds on a higher fraction of random initializations.
3. **On hard problems** ($n = 18 - 24$, strong plateau risk, multi-modal), HOPSO finds significantly better minima than Adam or natural gradient. The gap widens with problem difficulty.
4. **On very hard problems** ($n > 24$ with deep ansatz), all optimizers struggle, but HOPSO degrades more gracefully than gradient methods.

The thesis interprets this as evidence that **population-based exploration becomes more important as problem difficulty grows**, which is exactly what the regularization-as-dynamics framing predicts.

Closing Module 6

Module 6 has now traversed the full optimizer landscape:

- **Vanilla GD (6.1):** the baseline.
- **Natural Gradient (6.2):** geometric correctness at high cost.
- **Adam (6.3):** practical adaptive methods, the workhorse default.
- **Stochastic Optimization (6.4):** the foundational theory.
- **Convergence Theory (6.5):** what we can prove.
- **Optimal Transport (6.6):** the distributional view.
- **Population Methods (6.7):** the practical population framework.
- **HOPSO (6.8, this node):** the thesis's hybrid recommendation.

The arc of the module is from "simplest possible optimizer" to "most sophisticated thesis-aligned hybrid." Each step adds a layer that addresses a failure mode of the previous one.

The most important take-away: **there is no single best optimizer for VQA**. The right choice depends on the problem's difficulty, the noise level, the parameter count, and the available compute. The thesis's specific recommendation — HOPSO for difficult problems, Adam for easy ones — is one reasonable rule, but the broader framework (point vs population, gradient vs zero-order, objective-side vs dynamics-side regularization) is what matters.

Module 7 will look at all of this through a control-theoretic lens. Module 8 will use it to build the expressivity-trainability tradeoff. Modules 9 and 10 will tie it back to hardware and deployment.

Structural Role

This node is the **operational closure** of Module 6. It is also the most direct embodiment of the thesis's central optimization-side argument: that population-based, dynamics-side regularization is a viable and underexploited alternative to objective-side fixes.

Relations to Other Concepts

- **PSO (Eberhart-Kennedy 1995)** — the parent algorithm.
- **CMA-ES** — alternative population method with adaptive geometry.
- **Hybrid optimization** — the broader category of gradient + population mixes.
- **Module 5 regularization** — the framework HOPSO embodies.
- **Wasserstein gradient flow (6.6)** — the continuous limit of HOPSO's swarm dynamics.

Worked Example — HOPSO On A Plateau-Prone Toy

Consider a 6-qubit HEA with 3 layers and a global cost $C = \langle Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 \rangle$.

Vanilla GD with random init. Initial gradient is ~ 0.05, shot noise floor is ~ 0.03. The optimizer barely moves, and after 1000 iterations, the cost has decreased by < 0.01. **Failure.**

Adam with random init. Same regime. Adam's adaptive step size doesn't help — the gradient is too noisy. Same failure.

HOPSO with $N = 30$ particles. - Iteration 1: 30 particles initialized randomly. Global best particle has cost ~ 0.1 (better than typical). - Iteration 10: the swarm has concentrated around the global best. A few particles have wandered into less-plateau regions and discovered cost ~ -0.2. - Iteration 50: the global best is now -0.6. The gradient at this point is meaningful (~ 0.3), and the gradient terms in HOPSO start contributing significantly. - Iteration 200: the swarm has converged on the global minimum at cost -1.

Total cost comparison: - HOPSO: 200 iterations $\times 30$ particles $\times (2P + 1)$ cost evaluations per iteration $= \sim 200 \times 30 \times 37 \approx 220{,}000$ cost evaluations. - Vanilla GD that fails: 1000 iterations $\times (2P + 1) \approx 37{,}000$ cost evaluations, but doesn't reach the answer.

HOPSO uses 6x more cost evaluations but actually finds the minimum. Vanilla GD uses fewer but fails.

This is the canonical HOPSO benefit: **more total cost, but finds answers that gradient methods cannot.**

Visualization

Pending. A 2D plot showing two trajectories on a multi-modal cost landscape with a plateau region: vanilla GD (gets stuck in the plateau, never reaches the minimum) and HOPSO (population spreads, one particle finds the minimum, rest of the swarm converges). Caption: "HOPSO escapes plateaus that GD cannot."

Exercises

1. For a 4-parameter ansatz with cost $C = \theta_1^2 + \theta_2^2 + \theta_3^2 + \theta_4^2$, run HOPSO with 5 particles for 10 iterations from random initialization. How fast does the global best converge to zero?
2. For HOPSO on a 1D bimodal cost $C(\theta) = \cos(2\theta)$, starting with a uniform initial swarm, sketch how the population evolves and which minimum it converges to.
3. (VQA) For a 20-qubit barren-plateau-prone HEA, estimate the minimum population size N such that HOPSO has $> 50\%$ probability of having at least one particle outside the plateau region at initialization. (Use any reasonable assumption about the plateau coverage fraction.)

Research Extensions

- **Adaptive HOPSO hyperparameters** — auto-tuning w, c_g, c_p, c_s based on swarm behavior.

- **HOPSO with quantum-aware mutations** — using parameter-shift periodicity to design mutation operators.
- **Multi-population HOPSO** — running multiple independent swarms in parallel, with occasional cross-population exchange.
- **HOPSO + natural gradient** — combining the swarm dynamics with QFIM-rescaled gradient terms.

Cross-links to Other Courses

- **Evolutionary Computation** — the parent field of population methods.
- **Module 5 (Regularization)** — the theoretical framework HOPSO embodies.
- **Module 6 Section 6.7 (Population Methods)** — the broader class.
- **Module 7 (Control-Theoretic View)** — alternative framing of the same dynamics.

Variational Quantum Algorithms · Module 6 — Optimization Dynamics · Node 6.8
Concepts: stochastic-optimization, gradient-descent, feedback, optimization-landscape
Prerequisites: 6.1, 6.2, 6.7

hbar.university — own.your.cognition