The Inspiration Simplex

Using derivative-free optimization in concept space
to create new derivative-free optimizers

Peter Cotton

Working paper, July 16, 2026

Abstract

Place K established algorithms at the vertices of a simplex. A point is a vector of mixing weights, and a language model turns the point into a working program: the heaviest vertex supplies the architecture and the others graft their ideas into it, in the stated proportions (Figure 2 shows the literal prompt). Scoring the program on a benchmark makes the simplex a continuous space of algorithm designs that ordinary derivative-free optimizers can search. The construction's first product is Alloy, generated at the equal-weights point: on twenty-nine problems never used in any selection step it has the best mean rank at every budget from 60 to 480 evaluations and beats six competitors including CMA-ES pairwise ($p < 10^{-10}$); it ships in the humpday package. Letting weights go negative extends the recipes beyond the simplex; of six semantics for a minus sign, tested under a paired-ablation protocol, one survives out of sample, with a rule attached: negative weights repair weak hosts and spoil strong ones. Failures are reported throughout: the continuous search never beat the obvious point, single generations vary severalfold, and in-sample scores mislead reliably. Two further worked examples test the construction's reach: cache-eviction policies, where static blends fail and adaptive shares rediscover the design of CAR, and distributional forecasting, whose champion dominates synthetic suites, loses on real data, and ships anyway, recast as a transform inside the forecaster it could not beat.

1 Introduction

Producing new methods by combining old ones is established stochastic practice. Genetic programming assembles programs by mutation and crossover [2], and evolutionary methods generate images and designs the same way [3].

Language models now serve as the variation operator in the same loop: FunSearch [4], AlphaEvolve [5], and related work [6, 7]. All of these search a discrete space of individual artifacts by sampling variations of them.

A separate tradition makes algorithm design continuous by tuning the constants of a fixed template: algorithm selection [8], hyper-heuristics [9], automated configuration. There the space is continuous but the architecture is frozen.

The inspiration simplex is a space that is continuous *and* architecturally open: barycentric coordinates over K named methods, with a language model mapping each coordinate to a working program. Performance on a benchmark then defines an objective that standard derivative-free machinery can search.

2 The construction

Fix K established methods, the vertices. A candidate is a point $w = (w_1, \ldots, w_K)$ with $w_k \geq 0$ and $\sum_k w_k = 1$: a recipe saying how much of each ancestor a new program should contain (Figure 1).

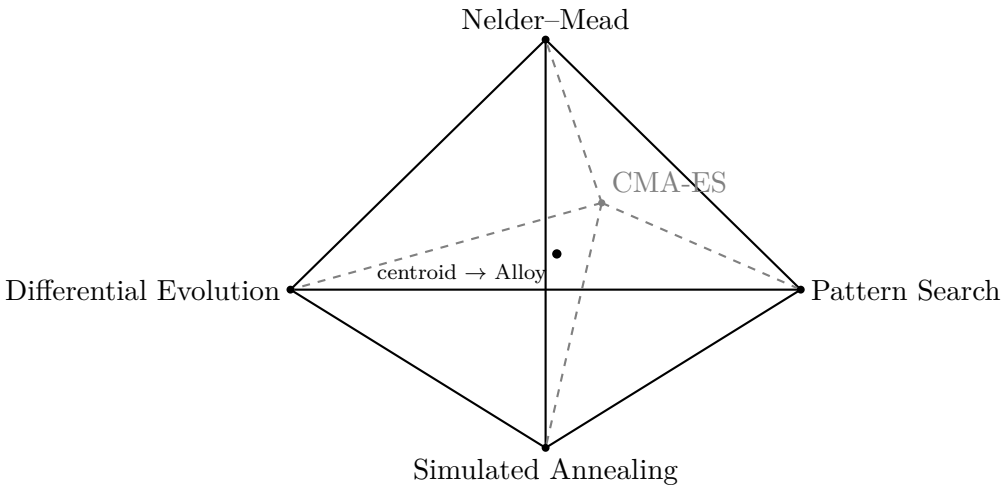

Figure 1: The five-vertex simplex drawn in perspective: a 4-simplex has ten edges, and the dashed ones recede to the rear vertex (CMA-ES). The marked interior point is the equal-weights centroid, the recipe whose prompt (Figure 2) produced Alloy.

The example uses five classical methods as vertices: Nelder–Mead [10], Differential Evolution [11], CMA-ES [12], pattern search [13], and simulated annealing [14]. The generator is a frontier language model (`claude-opus-4-8`).

Every iterative optimizer has the same five parts: how it starts (initialization), how it proposes the next candidate (move generation), how it decides to keep or reject it (acceptance), how it adjusts step sizes (adaptation), and when it restarts (restart).

Each vertex algorithm is famous for some parts and silent on others. Differential evolution is a way of initialising and proposing moves; simulated annealing is a way of accepting moves and restarting.

The heart of the map from recipe to program is an asymmetric instruction. The heaviest vertex is the host and supplies the architecture; every other vertex is a graft, and "70% A, 30% B" means "A, but borrow an idea from B", not a chimera. What B's idea looks like inside A's architecture is the model's decision.

The weights also set quantitative anchors. Each part is divided among the vertices that have a mechanism for it, in proportion to their weights, and the shares appear in the prompt as percentages: a vertex holding 25% of move generation means its move must be chosen a quarter of the time.

The coordinate therefore acts on two scales. Coarse changes act semantically: they change who hosts, which ideas appear, and how prominently. Fine changes, 20% versus 15% of a graft, move only the percentages; at that scale the map is parametric.

Figure 2 shows the actual prompt at the equal-weights center. The prompt is a deterministic function of the coordinate; the model decides how to implement each mechanism and how to wire the parts together.

```
Build a black-box numerical optimizer by BLENDING base algorithms - but
blending is ASYMMETRIC, not a 50/50 average. The highest-weight method is
the HOST architecture; the others donate GRAFTED ideas in proportion to
their weight (a 30% method contributes a prominent borrowed mechanism; a 5%
method only a light inflection). '70% A, 30% B' means 'A, but borrow an
idea from B', not a chimera.

HOST architecture (20%): NelderMead - build the skeleton from this.
GRAFT into it: 20% DifferentialEvolution, 20% CMAEvolutionStrategy,
20% PatternSearch, 20% SimulatedAnnealing

Inspiration weights: NelderMead 20%, DifferentialEvolution 20%,
CMAEvolutionStrategy 20%, PatternSearch 20%, SimulatedAnnealing 20%

Per-slot guidance (host owns a slot unless a graft's weight dominates it):
  - initialization: DifferentialEvolution (100%): population +
    difference-vector mutation (rand/1, current-to-best/1) and crossover
  - move_generation: NelderMead (25%): downhill simplex: reflect/expand/
    contract/shrink a set of n+1 points; DifferentialEvolution (25%):
    population + difference-vector mutation and crossover;
    CMAEvolutionStrategy (25%): sample from a Gaussian whose covariance
    adapts to successful steps; PatternSearch (25%): Hooke-Jeeves
    coordinate moves with a step size that shrinks on failure
  - acceptance: SimulatedAnnealing (100%): accept uphill moves with
    probability exp(-dF/T); cool T over time
  - adaptation: NelderMead (33%); CMAEvolutionStrategy (33%);
    PatternSearch (33%)
  - restart: SimulatedAnnealing (100%)

Write connective glue as needed so the slots cooperate (e.g. the
acceptance rule gates the move-generation output; adaptation updates the
step/covariance used by move-generation). Favour a real, working blend
over a faithful copy of any single algorithm.

Write a single Python function with EXACTLY this signature and contract:

    def optimize(objective, n_trials, n_dim):
        """Minimise `objective` (a callable taking a list of n_dim floats
        in [0,1] and returning a float). Use AT MOST n_trials calls to
        objective. Return (best_value, best_point)."""

Hard rules:
  - Pure Python. You may `import math` and `import random` only.
  - Every candidate must be clipped into [0,1] before evaluation.
  - Count objective calls; never exceed n_trials.
  - No global state, no prints. Just the function.

Return ONLY a ```python code block containing the optimize function.
```

Figure 2: The prompt at the equal-weights point $w = (0.2, 0.2, 0.2, 0.2, 0.2)$, lightly line-wrapped for the page. Moving w changes the host, the graft proportions, and the per-slot shares, and nothing else.

The generated program is compiled, validated, and scored on a benchmark, and that score is the objective value at w. Generation is stochastic, so the objective is noisy, as simulation-based objectives usually are.

Because the simplex maps to the unit cube by a standard bijection, any off-the-shelf derivative-free optimizer can search it; the outer loop sees only a box-constrained noisy objective.

The geometry protects the search. Every point is a committee of mechanisms that individually work, so the worst case of a step is a mediocre blend of good algorithms, not an arbitrary program; and nearby points receive nearby specifications, which gives a derivative-free optimizer something to exploit.

3 First example: Alloy

Candidates are scored on disguised versions of real-world objectives (enzyme kinetics, wind-farm layout, battery dispatch, and others). A seeded smooth bijection of the unit cube relocates each optimum, so a program cannot succeed by memorisation.

The score is regret normalised against a panel of standard optimizers, averaged over problems and seeds: 0 matches the best panel member everywhere, 1 the worst. Selection used sixteen problems, three seeds, 120 evaluations per run.

3.1 Selection

Table 1 scores the five pure recipes and the equal-weights centroid. The centroid's program, named Alloy, wins at 0.087; the best pure vertex, Nelder–Mead, scores 0.160.

recipe	regret
centroid (equal fifths) → Alloy	0.087
pure Nelder–Mead	0.160
pure Simulated Annealing	0.225
pure Differential Evolution	0.358
pure Pattern Search	0.477
pure CMA-ES	1.000

Table 1: Panel-normalised regret (lower is better) of the program generated at each recipe: sixteen disguised objectives, three seeds, 120 evaluations per run.

Structure mattered at generation time. Eight free-form attempts by the same model, prompted simply for the strongest optimizer it could write, gave a best of 0.141 and a median of 0.202 on the identical suite, against Alloy's 0.087.

3.2 Validation on problems the search never saw

A selection winner can be a fluke, so the test that matters is out of sample. Twenty-nine objectives, spanning 2 to 90 dimensions, were verified against the full history of the benchmark library to have been used by no selection step; each ran at budgets of 60, 120, 240 and 480 evaluations with five seeds, 580 instances in all.

Alloy has the best mean rank and the most outright wins at every budget against six competitors: Nelder–Mead, Differential Evolution, CMA-ES, Nevergrad's CMA, a tuned 14-gene

template, and the best free-form generation. Pairwise it beats each of them on 64% to 77% of instances, with sign-test p-values from 4×10^{-11} down to 1×10^{-40}.

Nor is it a lucky singleton. Adding the other strong selection-stage blends gives Table 2: the blends hold the top four places at the two smaller budgets, above every classical implementation, and the best of them is statistically even with Alloy (48% pairwise, $p = 0.29$).

	mean rank of 11 (lower is better)			
optimizer	60	120	240	480
Alloy	4.6	4.3	4.1	4.4
warm4 blend (Patt 53, CMA 23)	4.0	4.2	4.8	5.7
rand14 blend (Patt 64, CMA 21)	4.0	4.6	5.6	6.4
14-gene template (evolved)	5.8	5.6	5.3	5.4
warm7 blend (Patt 92)	5.0	5.1	5.9	7.4
generated pure Nelder–Mead	5.5	6.2	6.7	6.1
Nevergrad CMA	6.7	6.6	5.9	5.7
Differential Evolution	7.6	7.0	6.4	5.2
CMA-ES	7.7	7.1	6.3	5.7
free-form best	6.8	7.3	6.9	6.4
Nelder–Mead	7.8	7.8	7.7	7.1

Table 2: Race on the 29 held-out problems, 580 instances per budget column, rows ordered by mean rank across budgets. Blends dominate at small budgets; Alloy is the most consistent.

Alloy initialises with a DE population and builds a Nelder–Mead simplex from the best members. At each step it picks one of four move generators with equal probability (simplex reflect-expand-contract, DE mutation with crossover, a Gaussian perturbation with success-adapted covariance, a coordinate probe with shrinking step); every candidate passes a simulated-annealing acceptance gate, and stagnation triggers a reheat around the incumbent. It ships in the humpday package, in Python and JavaScript.

A distilled twin performs identically. Rewriting Alloy's architecture as one ordinary program with the mixing proportions as parameters, then tuning those with twenty cheap evaluations, gives an optimizer statistically even with Alloy on the held-out race (45% pairwise, $p = 0.05$) with no language model anywhere at run time.

4 What did not work

Generation is a lottery. Five draws at the centroid recipe scored 0.087, 0.249, 0.301, 0.320 and 1.000, and roughly one generation in seven failed to compile. Part of that failure rate was self-inflicted: an 8,000-token generation cap truncated the longest programs mid-line, and at 16,000 tokens the compile failures largely vanish. Alloy is the validated draw, not the expected one; the procedure that works samples a few draws and keeps what validates.

The map has blind spots. The pure CMA-ES recipe never produced a working program: correct covariance adaptation is too much to write in budgeted pure Python, and its generations were degenerate.

The continuous search never earned its keep on this vertex set. The winner was the centroid, the one point anyone would evaluate first; random sweeps and a guided derivative-free outer

loop (best-of-twenty 0.270 against 0.308 for random sampling) found nothing better. A second round with unequal vertices confirmed the diagnosis: there the equal blend was mid-pack, so the centroid's win reflects a symmetric vertex set, not a generally rich interior.

In-sample scores mislead, reliably. A 14-gene parametric template tuned by evolutionary search posted the best selection score of anything run (0.244) and then collapsed to mean rank 7 of 13 on the held-out problems. Free-form one-shots sit at the bottom of every held-out table despite respectable selection scores. Only races on untouched problems told the truth.

Cheap parametric search matches expensive semantic search per evaluation. On identical coordinates, a hand-parameterized version of Alloy's architecture scored 0.272 in-sample where fresh language-model generations scored 0.308. The model's value was the architecture, found once; after distillation it is no longer needed.

5 Seen from global optimization

The two-level structure is classical, split across two spaces. The semantic move is the basin hop and the parametric tuning is the local descent [15]; done inside the loop it is memetic search [16]; and the simplex is a continuous relaxation of architecture choice, as in differentiable architecture search [17], with a language model realizing the mixture instead of a weighted sum.

The closest workflow is SHGO [20]: sample the domain, identify the distinct basins, run one local minimization per basin. The basins live in the space of generated programs, not in w, since one coordinate yields different architectures across draws.

The recipe that survives: generate a small number of structured blends, validate the winner on problems the search never touched, distill it to a parametric family, and tune cheaply from then on. The language model is used a handful of times, early, and never again.

6 Outside the simplex: negative weights

Drop the non-negativity constraint and keep the sum: recipes now range over an affine slice of which the simplex is the all-positive face. A negative weight needs a semantics, and six were tested. Avoidance: shadow the negated algorithm's proposals at zero cost and steer away. Density: thin the host's proposals toward the signed mixture of proposal densities. Caricature: exaggerate the host along the axes where it most differs from the negated reference. Pitfalls: design the negated algorithm's failure modes out of the host. Baseline: treat its shadow as the bar to beat. Residual: graft in, amplified, only what the negated algorithm has that the host lacks.

Attribution is by paired ablation. Every generated program gates its negative-weight machinery through one module constant, so its twin, the same program with the constant zeroed, recovers the plain host and is scored on identical instances. Single-draw leaderboards reward whatever a label elicits from the generator; only the paired delta counts.

The protocol exists because avoidance sets the trap. Its best recipe won a selection round outright (0.179 against 0.326 for the best unsigned control), and disabling the repulsion in that same program improved it to 0.130: the win belonged to the unusually good host the prompt elicited, and the signed term was a handicap.

Under the protocol, caricature is the only semantics of the six that helps on average at selection (Table 3), and the mathematically prettiest, rejection-thinned proposal densities, goes zero for five against its twins. Sweeping all twenty ordered pairs, eight of nineteen benefit

from caricature, the map is uncorrelated with the ancestors' rank correlations (+0.05), and the magnitude dial optimises at restraint: mild exaggeration helps, twofold and beyond tips into parody.

semantics	mean paired delta	arms improving
caricature	−0.032	4/8
repulsion	−0.009	3/7
residual	+0.014	2/8
pitfalls	+0.018	2/7
density	+0.040	0/5
baseline	+0.086	0/4

Table 3: Six semantics for a negative weight under the paired protocol, selection suite spanning 2 to 100 dimensions; negative delta means the signed machinery helps.

Searching the extended slice (weights in roughly $[-1, 2]$, each negative weight a caricature reference with its magnitude as exaggeration length), the guided outer loop reached 0.078 with two thirds of its evaluations improving on their twins, against 0.047, 0.198 and 0.374 for three random shards: one lucky draw aside, guidance pays. That lucky 0.047, paired delta −0.134 at selection, sets the second trap: on the held-out problems its twin beats it on 65% of paired instances ($p = 9 \times 10^{-12}$). The paired ablation solves attribution, not transfer; a delta earned on the selection distribution is a claim about that distribution only, so the protocol acquires a second stage, and only deltas that survive out of sample count.

Two do. The CMA-ES-away-from-DE artifact beats its twin on 65% of held-out paired instances ($p = 2 \times 10^{-12}$), and annealing-away-from-pattern-search on 59% ($p = 10^{-5}$). The pattern across the raced winners is consistent: where the twin is already excellent the signed term vanishes or reverses; where the twin is weak the signed term repairs it. Negative weights, under the one semantics that works, are a treatment for weak hosts, not an enhancement for strong ones, and no signed artifact yet leads the held-out field absolutely.

A byproduct, twice replicated: the twins of the strongest signed recipes, budget-aware pattern-search hosts with Nelder–Mead grafts, rank first of fourteen at the largest budget on the held-out problems and edge Alloy pairwise there (54%, $p = 0.04$). That margin was discounted twice, by post-hoc selection among the raced twins and by the held-out set's reuse, and a succession match on fresh instances (disguise seeds no experiment had used) confirms the discount: Alloy beats each twin head to head, 57% and 56% ($p \leq 0.007$), and reversal of the reused-suite edge is itself a measurement of how much that reuse had come to be worth. The twins keep the best mean ranks at the two smallest budgets and remain the strongest small-budget artifacts found. Exploring outside the simplex keeps producing strong inhabitants of the inside.

7 Second example: cache eviction

The construction transfers to a new domain with two changed nouns. Vertices: LRU, LFU, FIFO, CLOCK and random eviction. Parts: on-hit bookkeeping, victim selection, on-insert bookkeeping, aging, and adaptation. Traces come from six seeded workload families whose parameters are randomized and whose keys are permuted per seed, the disguise analog, and the score is miss-rate regret normalised against the five-policy panel.

This domain offers what the optimizer example could not: proof that the interior is rich. ARC [18], the celebrated hand-designed blend of recency and frequency, scores 0.19 where the panel spans 0.47 to 0.65.

Static blending cannot reach it. With fixed shares, every generated blend (centroid 0.63, random interior 0.51 to 0.67) scores worse than every generated pure vertex (0.43 to 0.52), the inverse of the optimizer example. The diagnosis: mixing victim rules by proportion corrupts each policy's invariants, and ARC's value is not a fixed mixture at all but the online reallocation between recency and frequency.

One added instruction repairs it. Rendering the shares as a starting allocation the policy must adapt online, using feedback structures such as ghost lists, the generated pure-CLOCK point scores 0.24 at selection and 0.25 on held-out instances no experiment had used, second only to ARC and ahead of every panel policy and every other artifact. Its code maintains a ghost list of evicted keys tagged frequent-or-not and shifts a weight between frequency and recency whenever a ghost is re-requested: the design of CAR [19], ARC's successor by its own authors, rediscovered from a one-line semantic change. The adaptive centroid stays poor (0.57); in this domain adaptivity, not mixing, is the treasure.

Two lessons transfer. Whether the interior is reachable depends on where the domain keeps its value: optimizers compose per-step mechanism choices, so fixed proportions work; eviction policies live on cross-step invariants and workload tracking, so the coordinate must become state. And the elicitation effect replicates: the model's generated pure FIFO (0.43) beats the textbook FIFO reference (0.65) by budget-aware implementation alone.

8 Third example: distributional forecasting

The vertices become forecasting ideas: exponential smoothing, autoregression, GARCH, Student tails, and a regime watcher. The parts: conditional mean, conditional scale, distribution shape, adaptation, and regime handling. A candidate must report the log-density of the next observation before seeing it, then update.

Series come from six seeded generator families behind an affine disguise; the panel holds five simple references, and the champion reference is `laplace` from the skaters package, a likelihood-weighted ensemble refined over months, which towers over the panel at regret 0.05.

The two-example lesson makes a prediction, registered in the harness before any generation ran: forecasting keeps its value in cross-step invariants, so static shares should fail as they did for caches, and adaptive shares should carry. Confirmed: the static round's best is 0.30, the adaptive round's best 0.045, and the centroid is poor in both.

The winning artifact is a flat composite with no conjugation and no population. Two mean models and two variance models are each mixed by softmax over decayed log-scores, the champion's own ensemble principle reinvented at width two; a Student-t density tracks its degrees of freedom from realized kurtosis; a break monitor inflates variance fivefold and resets state when standardized errors run hot.

On synthetic ground it dominates: 0.036 against laplace's 0.049 on held-out seeds, nineteen of twenty on families that appeared in no selection round, and roughly eighty percent of paired instances across three fresh batches.

On five hundred real FRED series, under the skaters package's own harness, it loses the log-likelihood race on 71% of series. Three domains, three selection champions, and the same law each time: dominance on the selection distribution says little beyond it.

The signal survives the defeat. A crude score-weighted mixture with laplace beats laplace on CRPS on 73% of the same series; a cautious online regression nudging laplace's pre-projection mean wins 59% of log-likelihood contests with worst series -0.11 nats; and the final form, the signal recast as a conjugation transform joining laplace's own candidate population, wins 61% with worst series -0.012, every test of the host package passing. That form is merged: `markov_drift` ships in skaters, the first artifact of this construction to enter a production package by improving a champion it could not beat.

Both decisive integration choices came from human review, not from the construction: the caution of the coupling, and the placement of the regression before the lattice projection rather than after. The construction generates organs; the surgery remains a craft.

9 Generalization and further examples

Nothing here is specific to optimizers. The construction needs named concepts the model knows, a decomposition of the target artifact into parts, and a measurable score. Sorting algorithms, data structures, trading rules, prompt styles, and agent architectures all qualify, and further worked examples are planned for later versions of this paper.

The construction also refines. Like SHGO one can subdivide, searching the sub-simplex spanned by the best recipes at finer resolution; unlike SHGO one can promote discovered programs to vertices, so each generation's validated winners become the next generation's ancestors while the geometry stays K-dimensional.

The boundaries of the claims: budgets are small, the regime where blending plausibly helps most, and one generator model was used. The suite, while disguised, derives from public examples, so relatives of these objectives are likely in the generator's training data even though the optima's locations are not.

Availability. Alloy ships in the humpday package [1], in Python and JavaScript, and the forecasting artifact ships in the skaters package as the `markov_drift` transform. The search harnesses, the run logs behind every number in this paper, and every generated program are in the humpday repository.

References

[1] P. Cotton. HumpDay: recommendations and comparisons of derivative-free optimizers. `https://humpday.microprediction.org`, 2026.

[2] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[3] K. Sims. Artificial evolution for computer graphics. *Computer Graphics (SIGGRAPH)*, 25(4):319–328, 1991.

[4] B. Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[5] A. Novikov et al. AlphaEvolve: a coding agent for scientific and algorithmic discovery. Google DeepMind technical report, 2025.

[6] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley. Evolution through large models. arXiv:2206.08896, 2022.

[7] E. Meyerson, M. J. Nelson, H. Bradley, A. Gaier, A. Moradi, A. K. Hoover, and J. Lehman. Language model crossover: variation through few-shot prompting. arXiv:2302.12170, 2023.

[8] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[9] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[10] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[11] R. Storn and K. Price. Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[12] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[13] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, 1961.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[15] D. J. Wales and J. P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters. *Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[16] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech C3P Report 826, 1989.

[17] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *International Conference on Learning Representations*, 2019.

[18] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. *USENIX FAST*, 2003.

[19] S. Bansal and D. S. Modha. CAR: CLOCK with adaptive replacement. *USENIX FAST*, 2004.

[20] S. C. Endres, C. Sandrock, and W. W. Focke. A simplicial homology algorithm for Lipschitz optimisation. *Journal of Global Optimization*, 72:181–217, 2018.