RELAXATION-FREE ANNEALING FOR QUADRATIC UNCONSTRAINED BINARY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

A large part of combinatorial optimization can be written as quadratic unconstrained binary optimization (QUBO), the maximization of a quadratic function over binary vectors, which is also the form in which Ising machines and quantum annealers accept a problem. At the sizes met in practice, QUBO is solved approximately, and neural models trained to predict good assignments increasingly take the place of classical search rules. We reassess the advantage reported for these models by granting a classical method the same arithmetic, and we do so with a solver that involves no learning at all. Shifting the diagonal of the coefficient matrix below its smallest eigenvalue renders the quadratic part convex on the binary cube, so its tangent plane is an exact separable minorant, maximized in closed form by thresholding. We obtain an iteration that costs a single matrix-vector product, stays inside the discrete feasible set, needs no relaxation (in the sense of losing tightness), and never decreases the objective. We then lower the shift towards the smallest eigenvalue along a schedule and embed the iteration in batched perturb-anneal-harden cycles over an archive of the best solutions found so far, and we call this solver iPAMM. We characterize its fixed points, expose a trade-off between the curvature shift and the stationarity it guarantees, prove that the resulting path decreases a Lyapunov energy, and show the spin parameterization is canonical for this scheme. On dense benchmarks, iPAMM outperforms every learned, hybrid, relaxation-based and sampling-based solver we evaluate, and it also runs faster than the samplers among them.

1 INTRODUCTION

Given a symmetric matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$, quadratic unconstrained binary optimization (QUBO) asks for the binary vector maximizing $\mathbf{x}^\top \mathbf{Q}\mathbf{x}$. A broad class of combinatorial problems can be expressed in this form, among them max-cut [1], maximum independent set [2], set packing and partitioning [3, 4], graph colouring [5], linear ordering [6], quadratic assignment [7], portfolio selection [8] and transaction settlement [9], and a large family of constrained quadratic programs, all reached by elementary penalty transformations [10–15]. It is also the form in which quantum annealers and Ising machines accept a problem [16, 17]. Exact branch-and-bound and branch-and-cut methods [2, 18–21] reach only a few hundred variables on dense instances, more than an order of magnitude below the sizes studied here, whose best-known values remain unproven. Practice therefore rests on approximate solvers.

Approximate solvers for QUBO have long been organized around local search, which examines and accepts one variable flip at a time. Progress under such a rule is inherently serial, since each accepted flip determines which flips are examined next, so that the running time is governed by the depth of the search, whatever the width of the machine on which it runs. Modern accelerators reward the opposite arrangement, in which one wide arithmetic operation advances many candidate solutions at once, and it is this mismatch that motivated the search for alternatives. The most active of them replaces the hand-designed search rule by a learned one [22, 23]. Most such methods cast solution construction as a Markov decision process (MDP) and train a policy to select the next flip, either a pointer network [24–26] or a graph neural network [27, 28], following a broader programme of neural combinatorial optimization pursued for routing and assignment as well [29, 30]. The cost of learning is then paid only once, since a network trained on one family of instances is applied to every new instance without further optimization. The serial structure of the search, however, is retained.

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

Since the objective change must be re-evaluated after each accepted flip, the cost of a trajectory grows with the number of variables, and these models become expensive precisely at the sizes for which they were proposed.

Two more recent approaches depart from this sequential formulation. The first trains a network, per instance and without labels, on a continuous relaxation of the objective, as in PI-GNN [31] and its successors [32, 33]. What is optimized there is a relaxed proxy for the objective, and the binary solution comes from a rounding step that the training loss never sees. The second keeps the variables discrete and predicts the entire assignment in one shot, without a trajectory. The value classification model (VCM) [34] represents this route. A depth value network extracts a value feature per variable with a QUBO-tailored graph convolution, a classification network thresholds these into a solution, and a greedy-guided self trainer labels by improving the model's own output with batched greedy flips. What such a model can express is fixed once training ends. Nothing in the forward pass reads the objective at test time, so the assignment returned cannot be improved on the instance at hand, and the labels that supervise the network are themselves manufactured by the batched greedy flips against which its output is later measured.

Both approaches are evaluated against sequential heuristics, yet a learned solver of either kind ultimately rests on two elementary operations, batched dense matrix products and closed-form evaluations of the objective change caused by a single flip. Neither is a product of training, and both are equally available to a classical algorithm. The reported margins therefore confound what the network learned with the arithmetic it runs on, and separating the two requires a learning-free solver built from exactly those operations.

In this paper, we propose such a solver to address this issue, which we call iPAMM. At its centre is a surrogate that we construct for QUBO itself, one that lower-bounds the objective over the binary cube and touches it at the current iterate, so that maximizing the surrogate in place of the objective can never decrease it; a surrogate with these two properties is a minorizer in the sense of minorization-maximization (MM) [35, 36]. Our construction starts from the fact that for QUBO the quadratic diagonal reduces to a linear term on binary vectors, so that shifting the diagonal of $\mathbf{Q}$ below its smallest eigenvalue makes the quadratic part convex and its tangent plane a valid minorant. The minorant is affine, hence separable, and its binary maximizer follows in closed form by thresholding, at the cost of a single matrix-vector product. Above all the iteration is relaxation-free, in that the variables stay binary at every step, so no rounding is applied and no tightness is lost. This distinguishes our route from the classical convex one, which relaxes the problem, in the sense of losing tightness, by lifting it to a semidefinite program and then recovers a solution by hyperplane rounding [37, 38], so that it cannot guarantee optimality [39, 40]. The shift that validates the tangent inequality also appears in the update as an inertia term that damps coordinate changes, and we characterize this dependence exactly. We therefore treat the shift as a homotopy parameter and lower it along a schedule. At every value of the shift the iteration decreases a bivariate energy whose diagonal is minus the objective, so the schedule is a principled path that runs from exploratory threshold dynamics to guaranteed ascent. Embedding this schedule in a batched pass over an ensemble of parallel restarts gives PAMM, and repeating that pass in perturb-anneal-harden cycles over an archive of the best solutions found so far completes our solver iPAMM.

Our contributions. The main contributions of this work are summarized as follows.

- **A relaxation-free minorant on the binary cube.** A scalar shift of the diagonal of the coefficient matrix convexifies the objective on the binary cube without altering it, and the tangent plane of the convexified quadratic is a valid minorant whose binary maximizer follows in closed form by thresholding. The iteration stays inside the discrete feasible set, so no relaxation (in the sense of losing tightness) and no rounding enter at any stage.
- **Fixed points and the curvature–stationarity trade-off.** We prove monotone ascent and finite termination, and characterize the fixed points exactly through an identity relating the update field to the single-flip increments. The identity shows that a more conservative curvature shift guarantees a weaker form of stationarity, a trade-off that determines the design of the solver.
- **Annealing as a path of energy-minimizing dynamics.** At every value of the shift the iteration is a synchronous threshold network admitting a bivariate Lyapunov energy whose diagonal is minus the objective, its trajectories terminate at fixed points or two-cycles, and once the shift

is admissible only fixed points remain. The schedule is therefore a continuous path through energy-minimizing dynamics, exploratory at one end and guaranteed to ascend at the other.

- **The canonical spin parameterization.** Recasting the construction in spin variables is strictly preferable, because the admissible window of shifts then always has positive length, which it provably loses in binary variables on positive semidefinite instances such as max-cut.
- **A batched solver built on a single primitive.** We assemble these ingredients into PAMM and iPAMM, which advance an ensemble of candidate solutions with one dense matrix product per iteration and repeat perturb-anneal-harden cycles over an archive of the best solutions found so far, with no training, no loss of tightness and no instance-specific tuning.
- **A calibration study of learned QUBO solvers.** We evaluate iPAMM against state-of-the-art learned, hybrid, relaxation-based and sampling-based solvers on the standard dense benchmarks of this literature, at each competitor's own configuration and, for the strongest sampler, again at a matched wall-clock budget, with significance tests, seed-dispersion studies and ablations. The evidence places annealed dynamics as the reference point for learned QUBO solvers.

Notation. For the notation used throughout this paper, please refer to Appendix A.

2 Problem Formulation

Given a symmetric matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$, the QUBO problem seeks the binary assignment

$$\max_{\mathbf{x}\in\{0,1\}^n} \; f(\mathbf{x}) \;=\; \mathbf{x}^\top \mathbf{Q}\mathbf{x} \;=\; \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j, \tag{1}$$

and we refer to $f(\mathbf{x})$ as the objective function value (OFV) of the assignment $\mathbf{x}$. Problem (1) is NP-hard [41], and its difficulty has two independent sources. Its feasible set is the binary cube, which contains 2^n points and admits no useful ordering, and its objective is in general a non-concave quadratic form, since $\mathbf{Q}$ is typically indefinite. Practical methods therefore rely on additional structure. The one used here is that, on the binary cube, the change in the objective caused by flipping any single coordinate has a closed form.

Single-flip increments. Let $\mathbf{x} \in \{0,1\}^n$ and let k be a coordinate index. Flipping x_k produces the neighbour $\mathbf{x}' = \mathbf{x} + (1-2x_k)\,\mathbf{e}_k$, which is again a binary vector. Expanding the quadratic form at $\mathbf{x}'$ and using the symmetry of $\mathbf{Q}$,

$$\begin{aligned} f(\mathbf{x}') \;&=\; \mathbf{x}^\top\mathbf{Q}\mathbf{x} \;+\; (1-2x_k)\big(\mathbf{e}_k^\top\mathbf{Q}\mathbf{x} + \mathbf{x}^\top\mathbf{Q}\mathbf{e}_k\big) \;+\; (1-2x_k)^2\,\mathbf{e}_k^\top\mathbf{Q}\mathbf{e}_k \\ &=\; f(\mathbf{x}) \;+\; 2(1-2x_k)(\mathbf{Q}\mathbf{x})_k \;+\; q_{kk}, \end{aligned} \tag{2}$$

where the last equality uses $(1-2x_k)^2 = 1$, which holds precisely because x_k is binary. The quantity by which the OFV changes is called the objective function increment (OFI). Writing $\mathrm{OFI}_k(\mathbf{x}) = f(\mathbf{x}') - f(\mathbf{x})$ and collecting the n coordinates of (2) into a single vector gives

$$\mathrm{OFI}(\mathbf{x}) \;=\; \mathrm{diag}(\mathbf{Q}) \;+\; 2\,(\mathbf{1}-2\mathbf{x}) \odot \mathbf{Q}\mathbf{x}. \tag{3}$$

Equation (3) has two consequences for the design of the solver. First, the increments of *all* n possible flips are obtained from a single matrix-vector product $\mathbf{Q}\mathbf{x}$, and, when several candidate solutions are stacked as the columns of a matrix $\mathbf{X}$, from a single matrix-matrix product $\mathbf{Q}\mathbf{X}$. An entire ensemble of candidates can thus be advanced by one dense product, which is exactly the operation that modern accelerators execute most efficiently. Second, the increments are exact and not estimates, so any decision based on (3) is taken on the true objective.

Greedy flip search. The simplest procedure built on (3) is the batch greedy flip (BGF) heuristic. Starting from $\mathbf{x} = \mathbf{1}$, it repeatedly applies the single flip with the largest positive increment and updates $\mathbf{Q}\mathbf{x}$ incrementally in $O(n)$ operations, since flipping coordinate k modifies $\mathbf{Q}\mathbf{x}$ by one column of $\mathbf{Q}$. The procedure halts when $\max_i \mathrm{OFI}_i(\mathbf{x}) \le 0$, and its output is by construction a 1-*opt local maximum*, that is, a point that no single flip can improve. We use BGF both as a baseline and as a hardening step applied to the output of our own iteration. Its limitation is that its progress is serial, since each accepted flip depends on the previous one, so the number of sequential steps grows with n even though each step is cheap.

Both primitives above, the batched product $\mathbf{QX}$ and the closed-form increments (3), are exactly what the learned classifiers of Section 1 are built on, and neither of them is a consequence of training. Section 3 constructs an iteration that uses only these two operations.

3 SOLVING THE QUBO PROBLEM

The solver we propose for problem (1) is built on the MM principle (for an overview, see Appendix C), and its design is guided by two requirements. The minorant must be maximizable in closed form over the binary cube, so that each iteration costs no more than the matrix-vector product of Section 2, and its maximization has to take place on the cube itself, so that the problem is never relaxed, none of its tightness is lost, and no rounding is required at the end.

3.1 A LINEAR MINORANT ON THE BINARY CUBE

The construction rests on two elementary facts. The first is that the quadratic diagonal reduces to a linear term on binary vectors. Indeed, $x_i \in \{0,1\}$ implies $x_i^2 = x_i$ for every i, and summing over i gives

$$\mathbf{x}^\top \mathbf{x} \;=\; \sum_{i=1}^{n} x_i^2 \;=\; \sum_{i=1}^{n} x_i \;=\; \mathbf{1}^\top \mathbf{x}, \qquad \forall\, \mathbf{x} \in \{0,1\}^n. \tag{4}$$

The second is that (4) allows the curvature of the objective to be changed by an arbitrary amount while the objective itself is unchanged. For an arbitrary $\mu \in \mathbb{R}$ we may subtract $\mu\, \mathbf{x}^\top \mathbf{x}$ and add back $\mu\, \mathbf{1}^\top \mathbf{x}$, which by (4) are the same quantity, and obtain the exact splitting

$$f(\mathbf{x}) \;=\; \underbrace{\mathbf{x}^\top (\mathbf{Q} - \mu \mathbf{I})\, \mathbf{x}}_{\psi_\mu(\mathbf{x})} + \mu\, \mathbf{1}^\top \mathbf{x}, \qquad \forall\, \mathbf{x} \in \{0,1\}^n. \tag{5}$$

What the shift does change is the curvature of the quadratic part. Whenever

$$\mu \;\le\; \lambda_{\min}(\mathbf{Q}), \tag{6}$$

the matrix $\mathbf{Q} - \mu\mathbf{I}$ is positive semidefinite, since its eigenvalues are $\lambda_i(\mathbf{Q}) - \mu \ge 0$, and consequently ψ_μ is a convex function on the whole of $\mathbb{R}^n$. A convex quadratic dominates its own tangent plane, and in the present case the gap is available explicitly. Writing $\nabla \psi_\mu(\mathbf{x}_t) = 2(\mathbf{Q} - \mu\mathbf{I})\mathbf{x}_t$ and expanding,

$$\psi_\mu(\mathbf{x}) - \psi_\mu(\mathbf{x}_t) - 2\, \mathbf{x}_t^\top (\mathbf{Q} - \mu\mathbf{I})(\mathbf{x} - \mathbf{x}_t) \;=\; (\mathbf{x} - \mathbf{x}_t)^\top (\mathbf{Q} - \mu\mathbf{I})(\mathbf{x} - \mathbf{x}_t) \;\ge\; 0, \tag{7}$$

the inequality being a direct consequence of (6). Substituting the resulting lower bound on ψ_μ into the splitting (5) and regrouping the linear terms around $\mathbf{x}_t$ gives, for every $\mathbf{x} \in \{0,1\}^n$,

$$f(\mathbf{x}) \;\ge\; g(\mathbf{x} \mid \mathbf{x}_t) \;\triangleq\; f(\mathbf{x}_t) + \mathbf{g}_t^\top (\mathbf{x} - \mathbf{x}_t), \qquad \mathbf{g}_t \;\triangleq\; 2(\mathbf{Q} - \mu\mathbf{I})\, \mathbf{x}_t + \mu \mathbf{1}. \tag{8}$$

The function $g(\cdot \mid \mathbf{x}_t)$ is a valid minorizer, since it satisfies both conditions that the MM principle places on a surrogate, which Appendix C states. The lower-bound condition (S2) is (7), and the tangency condition (S3) holds because the linear term is zero at $\mathbf{x} = \mathbf{x}_t$. Note that the shift enters $\mathbf{g}_t$ only through the combination

$$\mathbf{g}_t \;=\; 2\mathbf{Q}\mathbf{x}_t - 2\mu\, \mathbf{x}_t + \mu \mathbf{1} \;=\; 2\mathbf{Q}\mathbf{x}_t + \mu(\mathbf{1} - 2\mathbf{x}_t), \tag{9}$$

so that a single product $\mathbf{Q}\mathbf{x}_t$ suffices to form it.

It remains to solve the maximization step (S4). Because $g(\cdot \mid \mathbf{x}_t)$ is affine, it is separable across coordinates, and the maximization over the cube decouples completely,

$$\max_{\mathbf{x} \in \{0,1\}^n} g(\mathbf{x} \mid \mathbf{x}_t) \;=\; f(\mathbf{x}_t) - \mathbf{g}_t^\top \mathbf{x}_t \;+\; \sum_{i=1}^{n} \max_{x_i \in \{0,1\}} g_{t,i}\, x_i. \tag{10}$$

Each scalar problem in (10) is solved by inspection, as the product $g_{t,i} x_i$ is maximized by $x_i = 1$ when $g_{t,i} > 0$, by $x_i = 0$ when $g_{t,i} < 0$, and by either value when $g_{t,i} = 0$. Breaking ties in favour of the current value $x_{t,i}$, a rule we call keep-on-tie, gives the *thresholding update*

$$x_{t+1,i} \;=\; \begin{cases} 1 & \text{if } g_{t,i} > 0, \\ 0 & \text{if } g_{t,i} < 0, \\ x_{t,i} & \text{if } g_{t,i} = 0, \end{cases} \qquad \mathbf{g}_t = 2\mathbf{Q}\mathbf{x}_t + \mu(\mathbf{1} - 2\mathbf{x}_t). \tag{11}$$

The update (11) maximizes the minorant *exactly*, so each step secures the largest improvement that the minorant can guarantee. It costs a single symmetric matrix-vector product and batches over instances and restarts when $\mathbf{x}_t$ is replaced by a matrix of columns, and Appendix E gives the operation count and compares it with neighbouring solver families. Its output is binary by construction, so the iteration remains in the feasible set of (1) throughout and requires no rounding.

Because the feasible set is finite, f is bounded on it and attains only finitely many values, and together with the properties above this gives the following convergence result.

Proposition 3.1 (Monotone ascent, convergence and finite termination)**.** *Let $\mu \le \lambda_{\min}(\mathbf{Q})$ and let $(\mathbf{x}_t)$ follow* (11)*. Then $f(\mathbf{x}_{t+1}) \ge f(\mathbf{x}_t)$ for all t, with strict inequality whenever $\mathbf{x}_{t+1} \ne \mathbf{x}_t$. Consequently the objective sequence converges, and the iterates reach a fixed point after finitely many steps (for the proof, see Appendix J).*

The ascent itself is the generic chain (S5) instantiated at the minorant (8); what the proof adds is the strictness, which follows from the separability of g, explicit upper and lower bounds on f over the cube, and the step from the convergence of the objective to that of the iterates, which are constant once no strict increase remains. The next subsection characterizes the fixed points exactly, and this characterization is what the schedule of Section 3.3 is built around.

3.2 The Curvature–Stationarity Trade-off

The quality of the fixed points reached by (11) can be assessed exactly, by comparing the update field (9) with the flip increments (3). Fix $\mathbf{x} \in \{0,1\}^n$ and a coordinate i, and multiply the i-th entry of (9) by $(1-2x_i)$. Using $(1-2x_i)^2 = 1$ once more, and then adding and subtracting q_{ii} so as to recognize the closed form (3), we obtain

$$(1-2x_i)\, g_i \;=\; 2(1-2x_i)(\mathbf{Q}\mathbf{x})_i + \mu \;=\; \big[\, q_{ii} + 2(1-2x_i)(\mathbf{Q}\mathbf{x})_i \,\big] - q_{ii} + \mu \;=\; \mathrm{OFI}_i(\mathbf{x}) - q_{ii} + \mu. \tag{12}$$

The interest of (12) is that its left-hand side is positive exactly when coordinate i changes under (11), since a change occurs either when $x_i = 0$ and $g_i > 0$ or when $x_i = 1$ and $g_i < 0$. A point $\mathbf{x}$ is therefore a fixed point of (11) if and only if

$$\mathrm{OFI}_i(\mathbf{x}) \;\le\; q_{ii} - \mu \qquad \text{for all } i. \tag{13}$$

This should be compared with the definition of a 1-opt local maximum, the strictly stronger requirement $\mathrm{OFI}_i(\mathbf{x}) \le 0$ for all i. The two conditions differ by the slack $q_{ii} - \mu$, and this slack is never negative, since evaluating the Rayleigh quotient of $\mathbf{Q}$ at the coordinate vector $\mathbf{e}_i$ gives $\lambda_{\min}(\mathbf{Q}) \le q_{ii}$, whence $q_{ii} - \mu \ge q_{ii} - \lambda_{\min}(\mathbf{Q}) \ge 0$ under (6). The slack also scales with the problem, since for a dense random matrix with entry scale σ one has $-\lambda_{\min}(\mathbf{Q}) = \Theta(\sigma\sqrt{n})$.

We refer to this phenomenon as the *curvature–stationarity trade-off*, and it is intrinsic to the construction. Condition (6) is what makes the tangent inequality (7) valid, and therefore what guarantees ascent, and the same shift appears in the update field (9) as the inertia term $\mu(\mathbf{1} - 2\mathbf{x}_t)$, which damps every coordinate change and thereby widens the set of points at which the iteration stops. Among all admissible shifts, $\mu = \lambda_{\min}(\mathbf{Q})$ is the largest and hence yields the tightest condition (13), while any safer choice weakens it and so can only enlarge the fixed-point set; the trade-off is thus unavoidable for scalar diagonal shifts, as Appendix J makes precise. Turning this dependence on μ from a constraint into a degree of freedom is what the schedule developed below does.

Collecting the two properties established so far, and adding the greedy hardening step of Section 2, gives the following description of the points at which the iteration comes to rest.

Corollary 3.2 (Convergence to stationary points)**.** *Let $\mu \le \lambda_{\min}(\mathbf{Q})$ and let $(\mathbf{x}_t)$ follow* (11) *from any $\mathbf{x}_0 \in \{0,1\}^n$. Then* (i) *after finitely many iterations the sequence is constant, $\mathbf{x}_t = \mathbf{x}^\star$, and the limit satisfies the stationarity condition* (13)*, that is $\mathrm{OFI}_i(\mathbf{x}^\star) \le q_{ii} - \mu$ for all i; and* (ii) *if $\mathbf{x}^\star$ is subsequently hardened by greedy flips, the hardening terminates after finitely many flips at a point $\mathbf{x}^{\star\star}$ with $f(\mathbf{x}^{\star\star}) \ge f(\mathbf{x}^\star)$ satisfying* 1-opt stationarity*, $\mathrm{OFI}_i(\mathbf{x}^{\star\star}) \le 0$ for all i, so that $\mathbf{x}^{\star\star}$ is a local maximum of f over the unit-Hamming neighbourhood (for the proof, see Appendix J).*

Both notions of stationarity are taken with respect to the unit-Hamming neighbourhood system, which is the discrete counterpart of a first-order optimality condition.

3.3 The Dynamics Along the Annealing Path

Section 3.2 shows that the shift which makes the minorant valid is also the quantity that sets the inertia of the update. This suggests treating μ not as a constant but as a homotopy parameter, so that the iteration is run first with shifts above $\lambda_{\min}(\mathbf{Q})$, where the inertia term is weak and the coordinates move freely, and the shift is lowered to an admissible value only in the closing iterations, where ascent is guaranteed. For $\mu > \lambda_{\min}(\mathbf{Q})$ the matrix $\mathbf{Q} - \mu\mathbf{I}$ is no longer positive semidefinite and the minorant argument of Proposition 3.1 does not apply, so the early iterations call for a different analysis. For any fixed μ, admissible or not, the update (11) is a synchronous threshold network,

$$\mathbf{x}_{t+1} = \mathbf{1}[\mathbf{W}\mathbf{x}_t + \mathbf{b} > 0], \qquad \mathbf{W} = 2(\mathbf{Q} - \mu\mathbf{I}), \qquad \mathbf{b} = \mu\mathbf{1}, \tag{14}$$

with a symmetric weight matrix. Networks of this kind are governed by a classical convergence theory [42, 43], and adapting it to the present setting supplies a Lyapunov function at every stage of the schedule.

Proposition 3.3 (Lyapunov energy and the disappearance of two-cycles)**.** *Fix any $\mu \in \mathbb{R}$ and let $(\mathbf{x}_t)$ follow* (11) *with keep-on-tie. Define the bivariate energy*

$$E_\mu(\mathbf{u}, \mathbf{v}) \;=\; -\,\mathbf{u}^\top \big(2\mathbf{Q} - 2\mu\mathbf{I}\big)\,\mathbf{v} \;-\; \mu\mathbf{1}^\top(\mathbf{u} + \mathbf{v}), \qquad \mathbf{u}, \mathbf{v} \in \{0,1\}^n. \tag{15}$$

Then $E_\mu(\mathbf{x}_t, \mathbf{x}_{t+1})$ is non-increasing in t, strictly decreasing whenever $\mathbf{x}_{t+2} \neq \mathbf{x}_t$, and the dynamics converge in finitely many steps to a fixed point or to a cycle of period two. Moreover $E_\mu(\mathbf{x}, \mathbf{x}) = -2f(\mathbf{x})$ for every binary $\mathbf{x}$, and if $\mu \leq \lambda_{\min}(\mathbf{Q})$ every period-two cycle is in fact a fixed point (for the proof, see Appendix J).

The proof telescopes the difference $E_\mu(\mathbf{x}_t, \mathbf{x}_{t+1}) - E_\mu(\mathbf{x}_{t+1}, \mathbf{x}_{t+2}) = (\mathbf{x}_{t+2} - \mathbf{x}_t)^\top(\mathbf{W}\mathbf{x}_{t+1} + \mathbf{b})$ and verifies that every term of the inner product is non-negative under the update rule.

Proposition 3.3 lets the iteration be read in two regimes. During the ramp, the iterates descend a bivariate energy landscape whose diagonal is exactly $-2f$, and since the shift reshapes only the off-diagonal, period-two directions, a two-cycle is the only alternative to a fixed point. At the terminal shift $\mu = \hat{\mu} \leq \lambda_{\min}(\mathbf{Q})$ no two-cycle remains, each step maximizes a valid minorant, and Proposition 3.1 applies. The schedule is therefore a homotopy through a family of energy-minimizing dynamics, exploratory at $\mu = 0$ and guaranteed to ascend at $\mu = \hat{\mu}$.

3.4 PAMM, the Parallel Annealed MM Iteration

Given a budget of T iterations and a ramp fraction $\rho \in (0, 1)$, PAMM ramps the shift linearly and then holds it,

$$\mu_t \;=\; \frac{t}{\lceil \rho T \rceil}\,\hat{\mu} \quad (t < \lceil \rho T \rceil), \qquad \mu_t \;=\; \hat{\mu} \quad (t \geq \lceil \rho T \rceil), \tag{16}$$

where $\hat{\mu} \leq \lambda_{\min}(\mathbf{Q})$ is obtained from a two-phase batched power iteration with a small safety margin, as detailed in Appendix J. Early iterations, for which $\mu_t \approx 0$, realize the synchronous threshold dynamics $\mathbf{x}_{t+1} = \mathbf{1}[2\mathbf{Q}\mathbf{x}_t > 0]$, that is the discrete-time Hopfield limit, in which the inertia term is absent and every coordinate follows the sign of its field. Late iterations, for which $\mu_t = \hat{\mu}$, maximize the minorant of Section 3.1, so Proposition 3.1 guarantees monotone ascent and finite termination of the final phase, and the stationarity condition (13) holds when the pass terminates.

Three practical components complete the solver, and Appendix J details them. The first is a set of R *parallel restarts*, stacked as a tensor dimension, so that the whole ensemble advances with one $n \times n$ by $n \times R$ product per iteration. The second is *greedy hardening* (PAMM-GF), which passes the final iterate to BGF and closes the slack between (13) and 1-opt stationarity in a few flips, as Corollary 3.2 guarantees. The third is the treatment of *warm starts*, which seed the restart ensemble and are driven through the schedule with it, since with $\mu = \hat{\mu}$ held fixed the inertia that guarantees ascent keeps a warm start at its initial point, as Section 3.2 predicts and Appendix I.2 confirms.

3.5 iPAMM, Iterated Annealing With a Solution Archive

A single pass, however wide its restart ensemble, spends its whole budget on one descent through the schedule (16). Classical search closes this gap by alternating perturbation with descent [44], and

the engine of Section 3.4 admits the same construction with all cycles batched. Figure S1 gives an overview.

The solver iPAMM runs an initial PAMM pass and then repeats the following cycle. The R columns are seeded round-robin from an archive holding the $P = 4$ best distinct solutions found so far. Each coordinate of column r is then flipped independently with a graduated probability $p_r \in [p_{\mathrm{lo}}, p_{\mathrm{hi}}]$, while column 0 keeps the incumbent unperturbed in order to preserve the best solution obtained up to that point. The ensemble is driven through a shorter schedule of T_c iterations, the k best columns are hardened by BGF, and the archive and the incumbent are updated. If the incumbent fails to improve for three cycles, the perturbation strengths are temporarily doubled, which we refer to as a "kick".

Two properties carry over from the guarantees of Sections 3.1 and 3.2. The incumbent objective is monotone in the cycle index, and every cycle ends in a phase at the admissible shift followed by 1-opt hardening, so that every solution iPAMM returns, and every member of its archive, satisfies both (13) and 1-opt stationarity; Appendix J derives this consequence of Corollary 3.2 in detail. In short, iPAMM is a batched perturbation–descent search whose local-search primitive is a PAMM pass, not a serial sequence of flips. It runs throughout in the spin variables of Appendix D, which are the canonical setting for this scheme.

4 Experimental Results

In this section we evaluate iPAMM against a broad suite of learned, hybrid, relaxation-based, sampling-based and physics-derived solvers on dense QUBO benchmarks of up to $7{,}000$ variables.

Compared methods. We benchmark our algorithm against twelve established and recent QUBO solvers. The classical baselines are **MFA (1989)** [45], mean-field annealing, and **SDP-GW (1995/2003)** [37, 38], the semidefinite relaxation with hyperplane rounding; the heuristics are the rule-based classifier **Diag (2020)** [26] and the greedy flip search **BGF (2024)** [34] of Section 2. The learned solvers are **PI-GNN (2022)** [31], **VCM (2024)** [34], **QRF-GNN (2024)** [32] and **Deep k-grouping (2025)** [33]; the recent samplers and continuous solvers are **iSCO (2023)** [46], **PQQA (2024)** [47], **PDBO (2025)** [48] and **mQO (2026)** [49]. Appendix B describes what each of them does and the limitation it brings into this comparison. For VCM, trained by the greedy-guided self trainer at $n_{\mathrm{train}} \in \{10, 20, 50, 100\}$, we report the raw classifier and its greedy-hybrid form, in which the output is polished by BGF.

Our own solvers are the members of the family developed in Section 3, all reported so that the contribution of each ingredient can be read off directly, namely **MM**, a single pass of the update (11) at a constant shift; **PAMM**, the scheduled pass of Section 3.4 with parallel restarts; **PAMM-GF**, the same pass followed by greedy hardening; **iPAMM** (Section 3.5), 16 perturb-anneal-harden cycles over a solution archive with $R = 128$ columns and top-48 hardening; and **iPAMM-L**, the longer-budget variant with 32 cycles and top-64 hardening. The single-pass members and the classifier-seeded combinations appear in Appendix G.

Evaluation metrics. We assess all solvers with the two metrics standard in the QUBO literature [44, 50–52, 34], complemented by a paired significance test. The **optimality gap** (GAP, lower is better) of a solution $\mathbf{x}$ is its relative shortfall $\mathrm{GAP} = 100\,(f^{\mathrm{bk}} - f(\mathbf{x}))/f^{\mathrm{bk}}$ against the best-known value f^{bk}, in percent, averaged over a family; on the benchmark suites f^{bk} is the best value reported in the literature [53, 44, 52], while on generated instances we use the strong per-instance reference described under Benchmarks below, so that a negative gap means a method has beaten it. The **average running time** (ART, lower is better) is the wall-clock duration of a complete single-instance solve, averaged over a family; it is essential here because the budgets compared differ by more than two orders of magnitude, and a gap not read together with a time is not interpretable [52]. Since an average over ten or twenty instances can be dominated by one outlier, we add the paired **Wilcoxon signed-rank test** [54] over the per-instance gaps (Appendix H.8), as is standard when heuristic solvers are compared on a common benchmark [52, 34].

Benchmarks. We evaluate on the three benchmark suites that are standard in this literature. The **B set** consists of the ten ORLIB bqp2500 instances ($n = 2500$, density 0.1) [53]. The **P set** consists of the 21 Palubeckis instances p3000.1–p7000.3 (n up to 7000, densities 0.5–1.0) [44]. The **G sets** are

Table 1: Results on the B and P benchmarks. **Bold** = best GAP, underlined = second best.

Algorithm	B2500		P3000		P4000		P5000		P6000		P7000	
	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓
MFA (1989) [45]	0.082	84	0.184	95	0.154	121	0.144	147	0.161	188	**0.119**	222
SDP-GW (1995/2003) [37, 38]	3.701	290	4.201	71	4.273	88	4.250	110	4.637	137	4.161	168
Diag (2020) [26]	99.564	3	99.155	0	97.348	0	98.305	0	99.278	0	98.906	0
PI-GNN (2022) [31]	1.755	6.5 s	3.665	1.4 s	3.108	1.8 s	3.370	2.8 s	3.567	3.7 s	3.358	4.8 s
iSCO (2023) [46]	**0.031**	66.0 s	0.126	116 s	0.152	155 s	0.245	175 s	0.244	201 s	0.257	230 s
BGF (2024) [34]	0.807	164	0.979	208	0.834	281	0.914	342	0.966	427	0.730	504
VCM10 (2024) [34]	0.944	7	1.410	8	1.282	13	1.384	19	1.602	25	1.399	34
VCM20 (2024) [34]	0.622	7	0.902	9	0.781	13	0.988	19	1.070	25	0.913	34
VCM50 (2024) [34]	0.444	7	0.726	9	0.619	13	0.735	19	0.854	25	0.665	34
VCM100 (2024) [34]	0.357	7	0.515	9	0.476	13	0.552	19	0.573	25	0.524	34
VCM10-BGF (2024) [34]	0.303	17	0.454	27	0.345	39	0.482	45	0.538	70	0.401	77
VCM20-BGF (2024) [34]	0.278	14	0.386	22	0.310	29	0.360	41	0.438	53	0.308	68
VCM50-BGF (2024) [34]	0.205	12	0.285	19	0.269	24	0.323	33	0.379	48	0.292	54
VCM100-BGF (2024) [34]	0.205	11	0.268	15	0.228	22	0.302	30	0.298	42	0.265	55
QRF-GNN (2024) [32]	0.656	18.1 s	0.691	21.1 s	0.645	30.1 s	0.674	38.6 s	0.748	53.1 s	0.682	35.3 s
PQQA (2024) [47]	0.199	15.9 s	0.347	21.8 s	0.318	33.5 s	0.401	43.4 s	0.457	63.1 s	0.390	81.4 s
Deep k-grouping (2025) [33]	3.701	12.3 s	7.779	12.4 s	8.153	16.0 s	8.391	20.9 s	8.575	27.4 s	7.636	34.6 s
PDBO (2025) [48]	2.746	3.2 s	6.041	3.8 s	5.678	4.9 s	6.149	5.8 s	6.852	7.9 s	6.349	9.7 s
mQO (2026) [49]	0.349	8.5 s	0.443	10.1 s	0.427	13.2 s	0.457	16.0 s	0.501	20.8 s	0.475	26.4 s
iPAMM (ours)	0.057	427	0.092	572	0.096	805	0.159	1.1 s	**0.114**	1.4 s	0.124	1.8 s
iPAMM-L (ours)	0.042	1.0 s	**0.036**	1.4 s	**0.069**	2.2 s	**0.092**	3.3 s	0.120	4.0 s	0.130	5.4 s

generated instances with $n \in \{20, \ldots, 1000\}$, symmetric with uniform integer entries in $[-100, 100]$, 100 per size, following the generator used in previous evaluations [52, 34]. For B and P the per-family averages of the best-known values match the published ones exactly, confirming our data conventions; for the G sets the reference is the better of PAMM-GF with 64 restarts and BGF on each instance. All three suites are dense, the regime our claims target; Appendix F details the recovery of the P set from its max-cut form in MQLib.

Protocol. All solve times are reported as ART and measured on a single NVIDIA RTX 3090. Every method is implemented in PyTorch and computes in float32; the reported objective values are recomputed in float64 and, since all matrices and solutions are integer-valued, are exact. Every competitor is run through the same interface, on the same hardware and on identical instances, in the configuration its own authors prescribe for problems of this kind. Appendix F documents the training setup of the learned models, and Appendix H.3 compares iPAMM with the strongest sampler at a matched budget.

Main results on the B and P benchmarks. Table 1 reports the complete comparison, from which we draw the following four observations. First, iPAMM leads every learned, hybrid and relaxation-based solver, reaching average gaps of 0.057–0.159% across B2500 and all five P families in 0.4–1.8 s per instance, and iPAMM-L, with its longer budget, reaches 0.036–0.130%. The margin is uniform and large, since the best learned hybrid (VCM100-BGF, 0.21–0.30%) is behind by a factor of 2 to 4 and the strongest raw classifier by 3.5 to 6, while PQQA (0.20–0.46%), mQO (0.35–0.50%), QRF-GNN (0.65–0.75%), PDBO (2.7–6.9%) and Deep k-grouping (3.7–8.6%) trail by factors of 2 to 85 on budgets 5 to 45 times larger, with $p < 10^{-7}$ in every pairing. PQQA drives a relaxation towards the cube and rounds at the end, and on the first B2500 instance, where its configuration returns 0.27%, it stays near that value even when run for 33 s, so its gap is structural and does not close with more compute. PDBO and QRF-GNN were designed for sparse instances and are run here exactly as specified, so their figures show how they carry over to the dense regime, and Appendix H.5 tests the converse direction on sparse max-cut, where the learned classifier fails outright. Second, time matching separates iPAMM from the recent samplers. Although iSCO, at its paper-specified configuration, posts a competitive gap on B2500 (0.031%), it needs 66 to 155 times the wall-clock time of iPAMM-L and iPAMM to do so. It also degrades with n even at full budget, to 0.245% on P5000 and 0.244% on P6000 at 175 and 201 s, so its advantage is confined to the smallest and sparsest family, and it does not survive a matched budget, at which iPAMM leads on every family (Appendix H.3). Third, the quality frontier belongs to annealed dynamics, and no learned classifier reaches it. Methods of this class hold the top of the table throughout, namely iPAMM-L, iPAMM and MFA, joined by iSCO at a budget two orders of magnitude larger. Our two methods give the best result on most families and come within a few hundredths of a point of it elsewhere, and every learned and hybrid entry, including models trained offline for up to forty minutes, sits below both of them on every family. Even mean-field annealing, decades older than the neural combinatorial optimization literature, outperforms each of these entries here, family by family, at 0.08–0.18% in 0.08–0.22 s, so the standard that learned solvers must meet is set by annealed dynamics. Within that class iPAMM leads, with a lower mean gap over all 31 instances than mean-field annealing (0.097% against 0.131%, $p \approx 5 \times 10^{-3}$), and it adds a property no other member has, in that each terminal-phase step is

the exact maximizer of a valid minorant (Proposition 3.1), so its schedule carries a guarantee of monotone ascent. The guarantee also buys reliability. Over 5 seeds on all 31 instances iPAMM never fails ($0.102 \pm 0.006\%$ overall), whereas MFA fails to leave the uniform state on 2 of its 55 runs at $n \geq 5000$, returning 96–97% gaps (Appendix H.4). Two roles remain for learning, namely the regime below our own running time, where a single forward pass returns a solution in milliseconds at gaps of 0.36–0.57%, and classifier outputs as seeds, since warm-starting with VCM50 improves most families, to 0.045–0.145% for iPAMM and 0.040–0.137% for iPAMM-L, within a few hundredths of a point of what the learning-free solver reaches on its own (Appendix G). Fourth, the iteration works because of its guaranteed core, and relaxation is not a productive detour. What separates iPAMM from a generic multistart procedure is that each cycle ends in a monotone phase, the solution archive only ever improves, and the hardening step closes the slack of (13), whose removal degrades every family, from 0.085% on B2500 to 0.327% on P6000 (Appendix G). Appendix I.2 confirms a further prediction of Section 3.2, namely that a warm start is improved only when it is driven through the schedule again. Every relaxation-based entry, meanwhile, lands behind, SDP-GW and PI-GNN by an order of magnitude or more, at 3.7–4.6% and at 1.8% on B2500 respectively, the latter at more than two hundred times the cost of a single pass, and the rest at the gaps quoted above. On a dense, near-full-rank $\mathbf{Q}$ the lift-and-round detour is outperformed by direct dynamics on the cube.

Further results. Five further studies are reported in the appendix, namely the *time-matched comparison* against the strongest GPU sampler (Appendix H.3); *size generalization* over $n \in [20, 1000]$ (Appendix I.1); *ablations of the engine*, covering the schedule, the budget split and greedy hardening (Appendix I.2); the *PAMM engine inside the trainer* as a label source (Appendix I.3); and *sparse instances* on Gset max-cut, a regime outside our dense scope that several of the competitors were designed for, in which iPAMM seeded with the output of mean-field annealing gives the best result on almost every instance (Appendix H.5).

5 Limitations

Problem (1) is non-convex, and no algorithm can guarantee a global maximizer [55]. As Section 3.2 makes precise, PAMM and iPAMM, built on the iteration analysed there, establish monotone ascent and the stationarity condition (13) at termination, strengthened to 1-opt by hardening.

6 Conclusion

This paper has presented iPAMM, a QUBO solver with no learned component, and used it to calibrate the advantage reported for learned models. Its engine is an exact separable minorant on the binary cube, obtained by shifting the diagonal below the smallest eigenvalue and maximized by thresholding at one matrix-vector product per iteration, so that the variables stay discrete, no tightness is lost, and the objective never decreases. The fixed points of this iteration are characterized exactly, which exposes the trade-off between the curvature shift and the stationarity it guarantees, and treating the shift as a homotopy parameter turns the iteration into a path of energy-minimizing dynamics, canonically expressed in the spin parameterization. Batched perturb-anneal-harden cycles over an archive of the best solutions found so far complete the solver. On dense benchmarks it outperforms every learned, hybrid, relaxation-based and sampling-based solver we evaluate, runs faster than the samplers among them, and stays ahead of a recent GPU sampler when the wall-clock budget is held equal.

The results also place annealed dynamics as a class ahead of the learned solvers proposed for this problem on every benchmark family, a lead visible even in mean-field annealing, which is decades older than the neural combinatorial optimization literature. iPAMM and its long-budget variant head that class, with the two lowest mean gaps and the best results on most families, and the guarantee they carry at the level of the individual update removes the failures that MFA shows on a small fraction of runs at the largest sizes. A role for learning remains, narrower than the benchmarks suggest, in inference at millisecond latency and in informative basins that improve iPAMM when used as seeds. What our results revise is the baseline itself, since a learned QUBO solver must improve not on a sequential heuristic, but on annealed dynamics running on the arithmetic it already occupies.

AI USE STATEMENT

AI-assisted editing was used for language editing and proofreading of the main paper and supplementary materials. The authors take full responsibility for the final content of this paper.

REPRODUCIBILITY STATEMENT

The solver is specified completely in Section 3 and in Algorithm S1 of Appendix J; the assumptions behind Proposition 3.1, Corollary 3.2 and Proposition 3.3 are stated where the results are, and the proofs are given in complete form in Appendix J. The benchmarks and the provenance of every instance, including the reconstruction of the P set from its max-cut form, are described in Section 4 and Appendix F, which also lists every hyperparameter of our own methods. We provide as supplementary material a zip archive containing the source code of our solver, the instance loaders and generators, the evaluation and analysis scripts, and the result files behind every number we report, together with a README that states how each table is reproduced.

REFERENCES

[1] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt, "An application of combinatorial optimization to statistical physics and circuit layout design," *Operations Research*, vol. 36, no. 3, pp. 493–513, 1988.

[2] P. M. Pardalos and G. P. Rodgers, "Computational aspects of a branch and bound algorithm for quadratic zero-one programming," *Computing*, vol. 45, no. 2, pp. 131–144, 1990.

[3] B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang, "A new approach for modeling and solving set packing problems," *European Journal of Operational Research*, vol. 186, no. 2, pp. 504–512, 2008.

[4] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego, "A unified modeling and solution framework for combinatorial optimization problems," *OR Spectrum*, vol. 26, no. 2, pp. 237–250, 2004.

[5] G. Kochenberger, F. Glover, B. Alidaee, and C. Rego, "An unconstrained quadratic binary programming approach to the vertex coloring problem," *Annals of Operations Research*, vol. 139, no. 1, pp. 229–241, 2005.

[6] M. Lewis, B. Alidaee, F. Glover, and G. Kochenberger, "A note on xqx as a modelling and solution framework for the linear ordering problem," *International Journal of Operational Research*, vol. 5, no. 2, pp. 152–162, 2009.

[7] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European Journal of Operational Research*, vol. 176, no. 2, pp. 657–690, 2007.

[8] G. Rosenberg, P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, and M. López de Prado, "Solving the optimal trading trajectory problem using a quantum annealer," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 6, pp. 1053–1060, 2016.

[9] L. Braine, D. J. Egger, J. Glick, and S. Woerner, "Quantum algorithms for mixed binary optimization applied to transaction settlement," *IEEE Transactions on Quantum Engineering*, vol. 2, 2021.

[10] P. L. Hammer and S. Rudeanu, "Pseudo-boolean programming," *Operations Research*, vol. 17, no. 2, pp. 233–261, 1969.

[11] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 155–225, 2002.

[12] M. Anthony, E. Boros, Y. Crama, and A. Gruber, "Quadratic reformulations of nonlinear binary optimization problems," *Mathematical Programming*, vol. 162, no. 1–2, pp. 115–144, 2017.

[13] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: a survey," *Journal of Combinatorial Optimization*, vol. 28, pp. 58–81, 2014.

[14] F. Glover, G. Kochenberger, R. Hennig, and Y. Du, "Quantum bridge analytics i: a tutorial on formulating and using qubo models," *Annals of Operations Research*, vol. 314, no. 1, pp. 141–183, 2022.

[15] A. P. Punnen, ed., *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications*. Springer, 2022.

[16] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, p. 5, 2014.

[17] T. Leleu, Y. Yamamoto, P. L. McMahon, and K. Aihara, "Destabilization of local minima in analog spin systems by correction of amplitude heterogeneity," *Physical Review Letters*, vol. 122, no. 4, p. 040607, 2019.

[18] F. Barahona, M. Jünger, and G. Reinelt, "Experiments in quadratic 0–1 programming," *Mathematical Programming*, vol. 44, no. 1-3, pp. 127–137, 1989.

[19] C. Helmberg and F. Rendl, "Solving quadratic (0,1)-problems by semidefinite programs and cutting planes," *Mathematical Programming*, vol. 82, pp. 291–315, 1998.

[20] F. Rendl, G. Rinaldi, and A. Wiegele, "Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations," *Mathematical Programming*, vol. 121, pp. 307–335, 2010.

[21] N. Krislock, J. Malick, and F. Roupin, "BiqCrunch: A semidefinite branch-and-bound method for solving binary quadratic problems," *ACM Transactions on Mathematical Software*, vol. 43, no. 4, 2017.

[22] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[23] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 130, pp. 1–61, 2023.

[24] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[25] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[26] M. Chen, Y. Chen, Y. Du, L. Wei, and Y. Chen, "Heuristic algorithms based on deep reinforcement learning for quadratic unconstrained binary optimization," *Knowledge-Based Systems*, vol. 207, p. 106366, 2020.

[27] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[28] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3243–3250, 2020.

[29] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in *International Conference on Learning Representations*, 2019.

[30] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21188–21198, 2020.

[31] M. J. Schuetz, J. K. Brubaker, and H. G. Katzgraber, "Combinatorial optimization with physics-inspired graph neural networks," *Nature Machine Intelligence*, vol. 4, no. 4, pp. 367–377, 2022.

[32] D. Pugacheva, A. Ermakov, I. Lyskov, I. Makarov, and Y. Zotov, "Enhancing gnns performance on combinatorial optimization by recurrent feature update," *arXiv preprint arXiv:2407.16468*, 2024.

[33] S. Bai, C. Yang, X. Bai, X. Zhang, and Z. Jiang, "Deep k-grouping: An unsupervised learning framework for combinatorial optimization on graphs and hypergraphs," *arXiv preprint arXiv:2505.20972*, 2025.

[34] M. Chen, J. Chun, S. Xiang, L. Wei, Y. Du, Q. Wan, Y. Chen, and Y. Chen, "Learning to solve quadratic unconstrained binary optimization in a classification way," *Advances in Neural Information Processing Systems*, vol. 37, 2024.

[35] D. R. Hunter and K. Lange, "A tutorial on mm algorithms," *The American Statistician*, vol. 58, no. 1, pp. 30–37, 2004.

[36] Y. Sun, P. Babu, and D. P. Palomar, "Majorization-minimization algorithms in signal processing, communications, and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 794–816, 2017.

[37] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.

[38] S. Burer and R. D. Monteiro, "A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization," *Mathematical Programming*, vol. 95, no. 2, pp. 329–357, 2003.

[39] P. Babu, P. Stoica, and A. Saini, "Fair principal component analysis (pca): minorization-maximization algorithms for fair pca, fair robust pca and fair sparse pca," *Transactions on Machine Learning Research*, 2025.

[40] M. M. Omati, P. Babu, P. Stoica, and A. Amini, "A max-min approach to the worst-case class separation problem," *Transactions on Machine Learning Research*, 2025. `https://openreview.net/forum?id=EEmwBd4tfZ`.

[41] P. M. Pardalos and S. Jha, "Complexity of uniqueness and local search in quadratic 0–1 programming," *Operations Research Letters*, vol. 11, no. 2, pp. 119–123, 1992.

[42] E. Goles-Chacc, F. Fogelman-Soulié, and D. Pellegrin, "Decreasing energy functions as a tool for studying threshold networks," *Discrete Applied Mathematics*, vol. 12, no. 3, pp. 261–277, 1985.

[43] J. Bruck, "On the convergence properties of the hopfield model," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1579–1585, 1990.

[44] G. Palubeckis, "Multistart tabu search strategies for the unconstrained binary quadratic optimization problem," *Annals of Operations Research*, vol. 131, pp. 259–282, 2004.

[45] C. Peterson and B. Söderberg, "A new method for mapping optimization problems onto neural networks," *International Journal of Neural Systems*, vol. 1, no. 1, pp. 3–22, 1989.

[46] H. Sun, K. Goshvadi, A. Nova, D. Schuurmans, and H. Dai, "Revisiting sampling for combinatorial optimization," in *International Conference on Machine Learning (ICML)*, pp. 32859–32874, 2023.

[47] Y. Ichikawa and Y. Arai, "Optimization by parallel quasi-quantum annealing with gradient-based sampling," *arXiv preprint arXiv:2409.02135*, 2024.

[48] W. Liu, A. Wang, D. Ma, H. Jiang, J. Wu, and W. Yang, "Smoothing binary optimization: A primal-dual perspective," *arXiv preprint arXiv:2509.21064*, 2025.

[49] Y. Sun, I. Alkhouri, C.-H. Huang, A. Velasquez, S. Jha, and R. Wang, "Mutation-guided differentiable quadratic combinatorial optimization," *arXiv preprint arXiv:2605.06921*, 2026.

[50] K. Katayama and H. Narihisa, "Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem," *European Journal of Operational Research*, vol. 134, no. 1, pp. 103–119, 2001.

[51] Y. Wang, Z. Lü, F. Glover, and J.-K. Hao, "Backbone guided tabu search for solving the ubqp problem," *Journal of Heuristics*, vol. 19, pp. 679–695, 2013.

[52] I. Dunning, S. Gupta, and J. Silberholz, "What works best when? a systematic evaluation of heuristics for max-cut and qubo," *INFORMS Journal on Computing*, vol. 30, no. 3, pp. 608–624, 2018.

[53] J. E. Beasley, "Obtaining test problems via internet," *Journal of Global Optimization*, vol. 8, pp. 429–433, 1996.

[54] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[55] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[56] Z. Lü, J.-K. Hao, and F. Glover, "A study of memetic search with multi-parent combination for ubqp," in *European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 154–165, 2010.

[57] Y. Wang, Z. Lü, F. Glover, and J.-K. Hao, "Probabilistic grasp-tabu search algorithms for the ubqp problem," *Computers & Operations Research*, vol. 40, pp. 3100–3107, 2013.

[58] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[59] M. C. Angelini and F. Ricci-Tersenghi, "Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set," *Nature Machine Intelligence*, vol. 5, no. 1, pp. 29–31, 2023.

[60] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[62] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Supplementary Material

These appendices collect the background, the proofs and the experimental evidence that support the main text, and they are organized so that they can be read on their own. In every table that follows, bold marks the best entry and underlining the second best, and shading identifies the rows of our main methods. Appendix A fixes the notation used throughout the paper. Appendix B places our work in the literature on exact methods, classical heuristics, relaxation-based approaches, annealed dynamics and learned solvers. Appendix C reviews the minorization-maximization principle in the form in which Section 3 uses it, and Appendices D and E give the spin parameterization and the cost of the resulting iteration relative to neighbouring solver families. Figure S1 gives an overview of iPAMM as a whole.

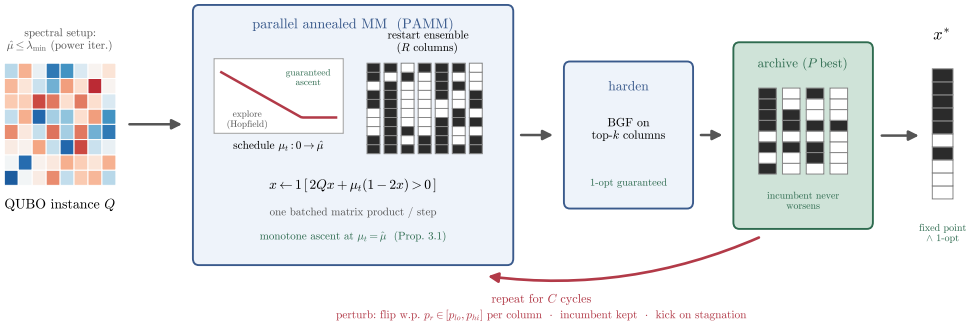

Figure S1: Overview of iPAMM, in which a restart ensemble evolves under the update (11) as μ_t ramps from 0 to $\hat{\mu}$, and each cycle re-seeds from an archive of the best solutions found so far under graduated perturbations.

The remaining appendices concern the experiments and the analysis. Appendix F documents the datasets, the configuration of our own methods, the training of the learned baselines and the timing protocol. Appendix G reports the members of the MM family and the component ablation of iPAMM. Appendix H contains the additional studies, namely distribution generalization, very large instances, the time-matched comparison, the seed-dispersion study, sparse instances, the parameter studies of the learned baseline, the batched increment evaluation and the initialization of BGF, the significance tests and the value-feature visualization, and Appendix I the size-generalization study, the ablations of the PAMM engine and the PAMM engine used inside the trainer. Appendix J gives the algorithm in pseudocode together with the proofs of Proposition 3.1, Corollary 3.2 and Proposition 3.3, and Appendices K and L report the label-oracle microbenchmark and the per-instance results.

A Notation

Boldface lowercase letters such as $\mathbf{a}$ denote vectors and boldface uppercase letters such as $\mathbf{Q}$ denote matrices, while $(\cdot)^\top$ is the transpose. The all-ones vector is written $\mathbf{1}$, the identity matrix $\mathbf{I}$, and the k-th coordinate vector $\mathbf{e}_k$. The (i, j) entry of $\mathbf{Q}$ is q_{ij}.

Two operators are attached to the diagonal, and they are distinguished by their capitalization. The first, $\mathrm{diag}(\mathbf{Q})$, returns the *vector* of diagonal entries of $\mathbf{Q}$, whereas $\mathrm{Diag}(\mathbf{a})$ returns the diagonal *matrix* carrying the vector $\mathbf{a}$ on its diagonal. Composing the two gives $\mathrm{Diag}(\mathbf{Q}) \triangleq \mathrm{Diag}(\mathrm{diag}(\mathbf{Q}))$, which keeps the diagonal of $\mathbf{Q}$ and zeroes the rest.

The smallest and largest eigenvalues of $\mathbf{Q}$ are $\lambda_{\min}(\mathbf{Q})$ and $\lambda_{\max}(\mathbf{Q})$, and $\mathbf{Q} \succeq 0$ states that $\mathbf{Q}$ is positive semidefinite. The symbol $\odot$ is the Hadamard, that is elementwise, product, and $\mathbf{1}[\cdot]$ and $\mathrm{sign}(\cdot)$ are the indicator and the sign, both applied elementwise. Finally, $\{0, 1\}^n$ is the binary cube and $\{-1, 1\}^n$ the spin cube.

B Related Work

Exact methods and classical heuristics. QUBO descends from pseudo-Boolean optimization [10, 11]. Exact approaches pair branch-and-bound or branch-and-cut with polyhedral or semidefinite bounds [2, 18, 19], an effort continued in solvers such as BiqMac [20] and BiqCrunch [21]. Their reach on dense instances is a few hundred variables, so none of them applies at the $n \in [2500, 7000]$ sizes of our benchmarks, whose best-known values remain unproven after two decades of attention [53, 44, 52]. An exact baseline is thus unavailable in this regime, and any comparison here is necessarily among approximate solvers. Heuristics accordingly dominate practice, among them simulated annealing [50] and memetic and path-relinking methods [56, 57], together with systematic evaluations of these families [52]. What they share is a serial acceptance rule, so that the number of sequential steps they need grows with the problem size however wide the hardware they run on.

Relaxation-based approaches. The classical convex route to (1) is the semidefinite relaxation with Goemans–Williamson rounding [37], made tractable at scale by the low-rank Burer–Monteiro factorization [38], which provides our SDP-GW baseline. Its weakness is that solving the semidefinite relaxation, which loses tightness, and then rounding its solution does not guarantee optimality [39, 40], so that the recovered binary point may be far from a maximizer of the original problem, and the lifting to matrix variables costs an order of magnitude more arithmetic per iteration than the problem itself carries. Our construction keeps the variables discrete throughout, so that no tightness is lost.

Annealed and physics-derived dynamics. On the non-convex side, Hopfield networks [58], mean-field annealing [45] and amplitude-corrected Ising machines [17] evolve continuous or synchronous binary dynamics instead of solving a relaxation. Their weakness is that no guarantee attaches to an individual update, so a trajectory that leaves the uniform state late can end arbitrarily far from a local maximum, and Appendix H.4 records exactly such failures for mean-field annealing at the largest sizes. Our Proposition 3.3 adapts the convergence theory of synchronous threshold networks [42, 43] to a shift that varies along a schedule. Our method belongs to this family, and what distinguishes it is the terminal phase, in which the update maximizes an exact minorant of the objective and therefore ascends it at every step. We also compare mean-field annealing with the learned solvers on the same accelerator, and Section 4 shows that annealed dynamics, led by iPAMM on most families, are ahead of every learned solver.

Learning-free GPU-parallel solvers. Several recent methods drop learning altogether and compete on parallel search. iSCO [46] cools a path-auxiliary discrete Langevin sampler whose Metropolis correction is spread over long flip paths, so it needs a large number of steps before it returns anything usable and has a budget below which it cannot function at all. PQQA [47] drives an entropy penalty on a relaxation, starting from a half-integral state and moving towards the cube, and since its iterates are integral only at the end, the rounding step lies outside the quantity being optimized and the gap saturates well short of the benchmark values however long it is run. PDBO [48] solves a primal-dual minimax smoothing by parallel gradient descent-ascent, so what it descends is a smoothed surrogate of the objective, and binarity is recovered only in the limit. Finally, mQO [49] adds mutation-based global resets to a differentiable solver, which leaves the relaxation and its closing rounding step in place and supplies no guarantee for any individual update. These four methods are benchmarked almost exclusively on sparse Gset max-cut instances. We evaluate all of them on dense QUBO at their own budget (Section 4), and the strongest of them, iSCO, also at the budget of iPAMM (Appendix H.3).

Neural combinatorial optimization. This field spans pointer networks [24, 25], attention models for routing [29, 30], and value learning with graph neural networks [27]. For QUBO and max-cut in particular, S2V-DQN [27] and ECO-DQN [28] learn flip policies in an MDP, DRLH [26] combines problem-specific features with a pointer network, and PI-GNN [31] trains a graph network per instance on the relaxed objective. Because each of them scores one flip at a time, or else improves a relaxed proxy that the rounding step never enters, the cost of a trajectory grows with the number of variables and the quantity actually optimized is not the objective. A critical assessment of this literature [59] showed that such models can fail to beat simple greedy algorithms, and our results reinforce this warning quantitatively for QUBO. Unsupervised graph networks have since improved on PI-GNN, in that QRF-GNN [32] feeds intermediate predictions back as recurrent features, and

Deep k-grouping [33] extends the recipe to k-way assignments with a Gini-coefficient annealing penalty whose $k = 2$ case is a plain QUBO. Both are still trained against a relaxed objective and were tuned on sparse graphs, and neither of them supersedes the VCM [34] described in Section 1, which sidesteps the MDP by classifying all variables at once and remains the strongest learned QUBO solver in our study. It therefore anchors the learned side of our comparison.

The compared methods. The solvers benchmarked in Section 4 are the following, each with the limitation that motivates our own construction. **MFA (1989)** [45] is mean-field annealing, which evolves continuous magnetizations under a damped synchronous mean-field update while a temperature parameter is ramped, and rounds them at the end, with no guarantee attached to any single update, so that a trajectory can end far below the quality its budget would allow. **SDP-GW (1995/2003)** [37, 38] is the semidefinite relaxation with Goemans–Williamson hyperplane rounding, made tractable at these sizes by the low-rank Burer–Monteiro factorization, and since that relaxation is not guaranteed tight, the rounded point carries no optimality claim for the original problem. **Diag (2020)** [26] is the rule-based value classifier that assigns each variable from its diagonal entry, and it therefore discards every off-diagonal coupling, which is where the difficulty resides. **PI-GNN (2022)** [31] is a graph convolutional network [60] trained per instance, without labels, on a continuous relaxation of the objective, with the relaxed solution rounded at the end, so that the rounding is never seen by the training loss and the model has to be retrained for every new instance. **iSCO (2023)** [46] is a discrete Langevin sampler that anneals the temperature and proposes paths of single-site flips accepted by a Metropolis correction, so its cost per unit of progress is high and its quality falls away sharply once the budget is reduced. **BGF (2024)** [34] is the batch greedy flip heuristic of Section 2, started from the all-ones vector, which returns a 1-opt local maximum; its acceptance rule is serial, so the number of sequential steps grows with the problem size, and it halts at the first such maximum it meets. **VCM (2024)** [34] is the value classification model of Section 1, trained by the greedy-guided self trainer, and nothing in its forward pass reads the objective at test time, so its output cannot be improved on the instance at hand and it does not survive a large shift away from its training distribution (Appendix H.5). **QRF-GNN (2024)** [32] is an unsupervised graph network that feeds intermediate predictions back as recurrent node features and polishes the rounded output by greedy flips, and it inherits both the relaxed training objective of PI-GNN and a design tuned for sparse graphs. **PQQA (2024)** [47] is a parallel annealing scheme that starts from a half-integral state and drives a relaxation towards the cube by increasing an entropy penalty, so its iterates become feasible only at the end and its gap saturates short of the benchmark values. **Deep k-grouping (2025)** [33] is an unsupervised k-way grouping network with a Gini-coefficient annealing penalty, run at the $k = 2$ setting that a plain QUBO induces, and the annealed term enforces feasibility and is separate from the objective. **PDBO (2025)** [48] is a primal-dual method that smooths the binarity constraint and performs parallel gradient descent-ascent on the resulting minimax problem, so the iterate it advances is not a feasible assignment until the smoothing parameter is driven to zero. **mQO (2026)** [49] is a differentiable solver over a relaxation with mutation-based global resets that restart replicas which have stopped improving, and those resets remove neither the relaxation nor the rounding that closes it.

C An Overview of the MM Framework

Since the algorithm developed in this paper is built on the minorization-maximization principle, we begin by stating that principle in the form in which we shall use it. Consider the constrained maximization problem

$$\max_{\mathbf{x}\in\mathcal{X}} \; f(\mathbf{x}), \tag{S1}$$

where $\mathbf{x}$ is the optimization variable, f is the objective to be maximized, and $\mathcal{X}$ denotes the feasible set. When f is non-concave, or $\mathcal{X}$ is discrete, or both, problem (S1) cannot be attacked directly. The principle replaces it by a sequence of simpler problems. At iteration t, given the current iterate $\mathbf{x}_t \in \mathcal{X}$, one constructs a surrogate function $g(\mathbf{x} \mid \mathbf{x}_t)$, called a minorizer of f, which is required to satisfy the two conditions

$$g(\mathbf{x} \mid \mathbf{x}_t) \le f(\mathbf{x}), \qquad \forall\, \mathbf{x} \in \mathcal{X}, \tag{S2}$$

$$g(\mathbf{x}_t \mid \mathbf{x}_t) = f(\mathbf{x}_t). \tag{S3}$$

Condition (S2) states that the surrogate lies below the objective everywhere on the feasible set, and condition (S3) states that the bound is tight at, or "touches", the current iterate. The next iterate is

then obtained by maximizing the surrogate in place of the objective,

$$\mathbf{x}_{t+1} \in \arg\max_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x} \mid \mathbf{x}_t), \tag{S4}$$

and the two operations of constructing and maximizing the surrogate are repeated until a stopping criterion is met. Figure S2 illustrates three consecutive iterations for a scalar objective, in which each surrogate touches f at the current iterate and lies below it elsewhere, and the maximizer of the surrogate becomes the next iterate, from which a new surrogate is built.

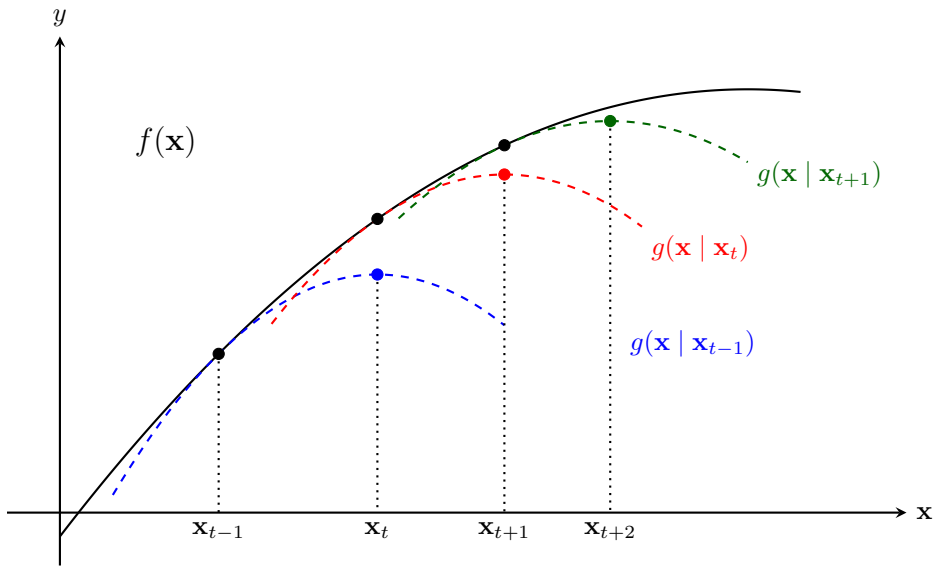

Figure S2: Three iterations under the MM principle. Each dashed curve is a minorizer $g(\cdot \mid \mathbf{x}_t)$, which touches the objective f at the current iterate and lies below it everywhere else; its maximizer is taken as the next iterate.

The value of this construction is the ascent property that it guarantees for free. Combining (S2), (S4) and (S3) in that order gives

$$f(\mathbf{x}_{t+1}) \geq g(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \geq g(\mathbf{x}_t \mid \mathbf{x}_t) = f(\mathbf{x}_t), \tag{S5}$$

where the first inequality is the lower-bound property, the second follows from the fact that $\mathbf{x}_{t+1}$ maximizes the surrogate whereas $\mathbf{x}_t$ is merely feasible for it, and the final equality is the tangency condition. The sequence $\{f(\mathbf{x}_t)\}$ is therefore non-decreasing, and it converges whenever f is bounded above on $\mathcal{X}$, which holds in particular when $\mathcal{X}$ is finite.

Two properties of the framework explain how it is used in the remainder of the paper. First, the ascent chain (S5) holds for *any* valid minorizer, and MM is consequently a design principle and not a particular algorithm, since what one obtains from it depends entirely on the surrogate one is able to construct, and two surrogates built for the same objective may lead to procedures of completely different cost and behaviour. Second, and for the same reason, the whole difficulty of applying the framework to a new problem lies in the choice of that surrogate, which must be simple enough to be maximized in closed form over $\mathcal{X}$ and at the same time tight enough for the resulting steps to make appreciable progress on f. A surrogate that is easy to maximize but far below f produces a slow iteration, whereas one that hugs f closely is usually as hard to maximize as f itself.

The construction of Section 3.1 resolves this tension for QUBO, since the minorizer is affine and therefore separable across coordinates, so that (S4) is solved *exactly* and in closed form by an elementwise thresholding rule, and the maximization takes place over the original binary set, with no relaxation and hence no loss of tightness. A more detailed exposition of the MM principle, together with a catalogue of techniques for constructing surrogates, may be found in [35, 36].

D THE SPIN PARAMETERIZATION

The binary identity (4) has a counterpart in spin space that is strictly stronger. Writing $\mathbf{x} = (\mathbf{1} + \mathbf{s})/2$ with $\mathbf{s} \in \{-1, 1\}^n$ turns (1) into $f = \mathbf{s}^\top \mathbf{J}\mathbf{s} + \mathbf{h}^\top \mathbf{s} + \text{const}$, with $\mathbf{J} = (\mathbf{Q} - \text{Diag}(\mathbf{Q}))/4$ and $\mathbf{h} = \mathbf{Q}\mathbf{1}/2$. On the spin cube $\mathbf{s}^\top \mathbf{s} = n$ is a constant, whereas on the binary cube $\mathbf{x}^\top \mathbf{x}$ equals the linear function of (4), so the diagonal shift $\mathbf{J} - \mu\mathbf{I}$ changes the objective only by a constant. The entire construction of Sections 3.1 to 3.3 goes through verbatim, with the update $\mathbf{s} \leftarrow \text{sign}(2(\mathbf{J} - \mu_t \mathbf{I})\mathbf{s} + \mathbf{h})$ and keep-on-tie. There is, however, one important difference. Since $\mathbf{J}$ is traceless by construction, its eigenvalues sum to zero, and therefore $\lambda_{\min}(\mathbf{J}) < 0$ strictly for every nonzero $\mathbf{J}$. The admissible window of shifts $[\,\lambda_{\min}, 0\,]$ therefore always has positive length.

The difference is decisive on positive semidefinite instances, among them max-cut in QUBO form, where $\mathbf{Q} = \text{Diag}(\mathbf{W}\mathbf{1}) - \mathbf{W}$ is the graph Laplacian and the binary form therefore has $\lambda_{\min}(\mathbf{Q}) \geq 0$. In the binary form the schedule (16) would then have no range over which to act, and the fixed-point condition (13) would reduce to $\text{OFI}_i \leq q_{ii}$. The spin form keeps a strictly negative terminal shift, and with it the inertia on which the stationarity guarantee rests, on every instance, and it costs nothing where the binary form already has one. For this reason iPAMM always runs in spin form.

E COST AND POSITION WITHIN THE ANNEALED FAMILY

PAMM costs $O(TRn^2)$ floating-point operations, entirely in the form of dense products; this is the same primitive that a depth-d forward pass of a value network uses dh times, with h the hidden width. Unlike the semidefinite route of Appendix B, the iteration never leaves the feasible cube, so its "rounding" is exact by construction and no tightness can be lost.

The spin view of Appendix D also locates the method within the annealed dynamics descended from Hopfield networks. The limit $\mu = 0$ of (11) is zero-temperature synchronous mean-field, and replacing the hard sign by $\tanh(\beta \cdot)$ while scheduling β in place of μ recovers classical mean-field annealing (MFA) [58, 45], with amplitude corrections giving Ising-machine solvers [17]. What distinguishes our scheme within this family is the status of its individual updates, in that during the terminal phase each update is the exact maximizer of a valid minorant of f, and therefore cannot decrease the objective (Propositions 3.1 and 3.3). This property allows the engine to serve as a label oracle inside a self-training loop and as a warm-start refiner, and Appendix H.4 shows that it also makes the solver reliable at the sizes where MFA fails on some runs. For this reason our experiments include MFA as a baseline, since annealed dynamics are the natural reference for iPAMM. Appendix J expands the cost comparison.

F DATASETS AND IMPLEMENTATION DETAILS

Data. The B set consists of the ten ORLIB bqp2500 instances [53] (density 0.1, integer coefficients in $[-100, 100]$). The P set consists of the 21 Palubeckis instances p3000.1–p7000.3 [44] (densities 0.5–1.0). We obtained the instances from the public MQLib library [52], which stores them in max-cut form on $n + 1$ nodes, and we inverted the standard QUBO-to-max-cut transformation. With root node $r = n + 1$, we set $q_{ij} = -w_{ij}$ for $i \neq j \leq n$ and $q_{ii} = w_{ir} - \sum_{j \neq i} q_{ij}$, and the sign convention is auto-detected by requiring the recovered diagonal to lie in the $[-100, 100]$ range of the generator. The recovered matrices are symmetric and integer with the documented densities, and the per-family averages of their best-known values match the published ones exactly, confirming the reconstruction. The G instances follow the generator used in previous evaluations [52, 34], being symmetric with entries uniform in $[-100, 100]$.

Configuration of our methods. All our methods use the shift estimate of Appendix J, and every scheduled pass ramps the shift over the first 70% of its iterations ($\rho = 0.7$). MM runs the update (11) at the constant shift $\hat{\mu}$ from the all-ones vector, and PAMM runs the schedule (16) for $T = 300$ iterations over the all-ones vector and 31 random restarts. iPAMM opens with a pass of 300 iterations over $R = 128$ columns and continues with cycles of $T_c = 120$ iterations, with perturbation probabilities spaced evenly over $[p_{\text{lo}}, p_{\text{hi}}] = [0.02, 0.25]$, an archive of $P = 4$ solutions and BGF hardening of the $k = 48$ best columns, for 16 cycles in all, while iPAMM-L runs 32 cycles and hardens the $k = 64$ best columns.

Training the learned baselines. Each learned baseline is trained at the hyperparameters its own paper prescribes, with Adam [61] and Xavier initialization [62] as specified there, and nothing in any of them is retuned by us. Appendix H.6 confirms, module by module and parameter by parameter, that VCM, the strongest of them, is at the optimum of its architecture, the hidden size and the training depth both peaking at the configuration used throughout. Training a model of this kind takes 8–40 minutes per size on our hardware, and the training corpus is a fifth of the published one.

Competitors. Every competitor is taken at the configuration its own paper prescribes.

Timing. Each running time is measured with batch size 1 on an otherwise idle GPU with 24 GB of memory, synchronized around the full solve call.

G Ablations of the MM Family and of the iPAMM Components

Table S1 reports the single-pass members of the MM family (MM, PAMM, PAMM-GF), the single-pass classifier seeding (VCM50+PAMM, in which the restart ensemble is seeded with the VCM50 output and 2%-perturbed copies), and the component ablation of iPAMM, which consists of removing the per-cycle BGF hardening (iPAMM without hardening) and of seeding the initial ensemble with the VCM50 output under graduated perturbations (iPAMM + VCM50 seed, and the long variant). We draw three conclusions from this table. First, each ingredient of iPAMM contributes. Iterated cycles improve a hardened single pass by a factor of 4 to 7 in gap at about an order of magnitude more running time, and removing the hardening costs up to nearly a factor of 3 on the largest families. Second, classifier seeding composes with iteration, as iPAMM+VCM50 improves on iPAMM across almost the whole suite (for instance 0.045% against 0.057% on B2500, and 0.059% against 0.092% on P3000), and iPAMM-L+VCM50 has the lowest average over the six families (0.077% against 0.082%), while iPAMM-L on its own remains ahead of it on P3000 and P6000. The learned classifier is thus informative where one is to hand, and the iterated search converts that information into better solutions than either component alone; the results of Table 1 are reached without it. Third, the same holds for the single-pass hybrid (VCM50+PAMM), but the iterated variants remain ahead of it throughout. We therefore recommend seeding iPAMM when a trained classifier is available and running plain iPAMM otherwise.

Table S1: MM-family variants and iPAMM component ablation (GAP % / ART). Same protocol and marking as Table 1.

Algorithm	B2500		P3000		P4000	
	GAP (%) ↓	ART (ms) ↓	GAP (%) ↓	ART (ms) ↓	GAP (%) ↓	ART (ms) ↓
MM (single pass)	0.780	24	1.050	25	1.010	35
PAMM (+31 restarts)	1.017	27	0.828	31	0.858	43
PAMM-GF (+hardening)	0.425	30	0.615	40	0.533	58
VCM50+PAMM (single-pass seed)	0.125	331	0.180	498	0.180	813
iPAMM without hardening	0.085	375	0.168	372	0.156	520
iPAMM	0.057	427	0.092	572	0.096	805
iPAMM + VCM50 seed	0.045	605	0.059	723	<u>0.069</u>	1.1 s
iPAMM-L	<u>0.042</u>	1.0 s	**0.036**	1.4 s	<u>0.069</u>	2.2 s
iPAMM-L + VCM50 seed	**0.040**	1.5 s	<u>0.045</u>	2.1 s	**0.056**	3.1 s
Algorithm	P5000		P6000		P7000	
	GAP (%) ↓	ART (ms) ↓	GAP (%) ↓	ART (ms) ↓	GAP (%) ↓	ART (ms) ↓
MM (single pass)	1.008	48	1.069	67	0.992	85
PAMM (+31 restarts)	0.911	59	0.979	80	0.953	98
PAMM-GF (+hardening)	0.633	75	0.514	114	0.641	125
VCM50+PAMM (single-pass seed)	0.225	1.3 s	0.291	1.8 s	0.228	2.4 s
iPAMM without hardening	0.253	716	0.327	969	0.222	1.3 s
iPAMM	0.159	1.1 s	**0.114**	1.4 s	0.124	1.8 s
iPAMM + VCM50 seed	<u>0.084</u>	1.4 s	0.145	1.8 s	<u>0.109</u>	2.1 s
iPAMM-L	0.092	3.3 s	<u>0.120</u>	4.0 s	0.130	5.4 s
iPAMM-L + VCM50 seed	**0.076**	4.0 s	0.137	5.8 s	**0.108**	6.7 s

H Additional Studies

Evaluations of QUBO solvers customarily go beyond a single benchmark table, and the studies collected here follow that practice [52, 44, 50, 31, 34]. We report generalization across instance distributions, behaviour on very large instances, a time-matched comparison, a seed-dispersion study, results on sparse instances, the module and parameter studies that document the configuration of the learned baseline we train ourselves, the efficiency of the batched increment evaluation together with the initialization of BGF, the significance tests, and a visualization of the value features of the classifier. Wherever a comparison between solvers is meaningful, we extend the study to iPAMM and to the other members of the family of Section 3.

H.1 Distribution generalization

A learned solver is trained on one family of coefficient matrices and applied to another, so its behaviour under a shift of the generating distribution is a natural object of study, and it is one that evaluations of heuristic and learned QUBO solvers alike have examined [52, 31, 34]. We generate 200 instances per cell from five distribution variants of the G generator at sizes $n \in \{20, 50, 100\}$, namely standard-normal entries (RN-1) and uniform entries with elements zeroed independently with probability 90% (R-0.1), 70% (R-0.3), 40% (R-0.6) and 10% (R-0.9). The default training distribution is R-1, in which all entries are nonzero, so that the five variants probe two distinct shifts, a change of the entry law and a progressive sparsification. The VCM models are tested zero-shot, using the size-matched models trained on R-1, since retraining per cell would answer a different question. Table S2 reports gaps against a strong per-instance reference, taken as the best of all listed methods together with a hardened 64-restart PAMM.

Two conclusions follow. First, the classifier holds up only while the shift is small, its gaps remaining at 0.1–0.4% across these variants of the random family on which it was trained, and the sparse Gset results of Appendix H.5 show how sharply this changes once the shift is large. Second, iPAMM is distribution-independent by construction, since nothing in the update (11) refers to the law of the entries of $\mathbf{Q}$, and it attains the best or the tied-best gap in every cell. The comparison thus separates a property that has to be established empirically for a trained model from one that is structural for ours.

Table S2: Distribution generalization. Average GAP (%) against the per-cell reference, 200 instances per cell.

Size	Algorithm	RN-1	R-0.1	R-0.3	R-0.6	R-0.9
		GAP (%) ↓				
G20	MFA (1989) [45]	0.147	0.790	0.332	0.169	0.197
	BGF (2024) [34]	2.178	1.440	1.521	1.391	1.630
	VCM (2024) [34]	0.102	0.291	0.293	0.396	0.099
	VCM-BGF (2024) [34]	0.012	0.085	0.057	0.102	0.033
	iPAMM (ours)	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
G50	MFA (1989) [45]	0.147	0.326	0.491	0.150	0.457
	BGF (2024) [34]	1.367	1.984	1.548	1.459	1.446
	VCM (2024) [34]	0.198	0.167	0.176	0.213	0.172
	VCM-BGF (2024) [34]	0.066	0.085	0.086	0.062	0.089
	iPAMM (ours)	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
G100	MFA (1989) [45]	0.126	0.266	0.158	0.132	0.478
	BGF (2024) [34]	1.394	1.832	1.397	1.463	1.297
	VCM (2024) [34]	0.308	0.262	0.283	0.194	0.270
	VCM-BGF (2024) [34]	0.129	0.183	0.135	0.088	0.131
	iPAMM (ours)	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**

H.2 Very large instances

The benchmark suites of Section 4 stop at $n = 7000$, and the ordering they induce is worth testing beyond that point, since the cost of every method in the study grows with n but not at the same rate. Table S3 therefore evaluates generated dense instances with $n = 10{,}000$ (3 instances) and $n = 20{,}000$

(2 instances). Gaps are measured against the best method per instance, since no best-known values exist for generated matrices.

Annealed dynamics outperform the learned family at these sizes as well, so the central ordering is preserved, and iPAMM is level with the best entry on G10000 (0.040% against 0.039%). Within the annealed family the two methods are close at the largest size, where MFA holds a narrow edge on G20000, since at $n \geq 10^4$ a single matrix product is expensive enough that the cycles of iPAMM complete only a few passes within the allotted time, whereas the damped mean-field update spends the same arithmetic on one long trajectory. These are single-run figures, and Appendix H.4 shows that the advantage of MFA at large n holds in the typical case only, since at $n \geq 5000$ on the P set 2 of its 55 runs fail outright, a failure mode that the guaranteed iteration does not have. On the learned side, the strongest classifier hybrid (VCM50-BGF) trails iPAMM by a factor of 4 to 5 in gap, and the raw VCM50 and BGF trail by more than a factor of 15.

Table S3: Very large generated instances ($n = 10{,}000$ and $n = 20{,}000$).

Algorithm	G10000		G20000	
	GAP (%)↓	ART↓	GAP (%)↓	ART↓
MFA (1989) [45]	**0.039**	406 ms	**0.000**	1.4 s
BGF (2024) [34]	0.791	730 ms	0.593	1.5 s
VCM50 (2024) [34]	0.678	102 ms	0.610	316 ms
VCM50-BGF (2024) [34]	0.216	109 ms	0.129	317 ms
iPAMM (ours)	<u>0.040</u>	2.4 s	<u>0.034</u>	7.8 s

H.3 TIME-MATCHED COMPARISON

Table 1 gives every method its paper-specified configuration, so the wall-clock times vary by two orders of magnitude, and Table S4 compares iPAMM with iSCO, the strongest of the samplers at its own budget, when both are given the same time. We re-tune the single budget parameter of iSCO per instance, so that its measured wall-clock time matches the ART of iPAMM on that family, and leave every other setting untouched. The parameter is found by a binary search with six probes, accepting the largest value whose measured time does not exceed the ART of iPAMM for that family; the reported times confirm that the match lands within roughly 15% in every case, which is well inside the margin needed for the conclusion.

At this budget the two methods separate clearly, and iPAMM is better on every family. iSCO is then outside the regime for which it was designed, as its path-auxiliary sampler needs long flip paths, each carrying a Metropolis correction; at a sub-second budget it completes only 16 to 24 sampler steps and returns gaps of 51.7–71.1%, so that it cannot be regarded as a sub-second method at these sizes. The contrast is structural, since the per-step cost of iPAMM is a single batched matrix-vector product and every step of its terminal phase is a guaranteed ascent, so its quality falls away smoothly as the budget shrinks, whereas a sampler that spreads a Metropolis correction over long paths has a floor below which it cannot function. The observation concerns the class of method, since solvers whose inner loop is one batched matrix product are the ones suited to the sub-second regime, and the sampling-based solvers that share our learning-free premise are one to three orders of magnitude away from it.

Table S4: Time-matched comparison. The budget parameter of iSCO is re-tuned per instance to the ART of iPAMM. **Bold** = best, <u>underlined</u> = second best.

Algorithm (at iPAMM's budget)	B2500	P3000	P4000	P5000	P6000	P7000
	GAP (%)↓					
iSCO (2023) [46]	<u>51.736</u>	<u>53.701</u>	<u>56.839</u>	<u>60.355</u>	<u>68.484</u>	<u>71.095</u>
iPAMM (ours)	**0.057**	**0.092**	**0.096**	**0.159**	**0.114**	**0.124**

H.4 SEED VARIANCE AND THE RELIABILITY OF THE ASCENT GUARANTEE

Table 1 reports a single run per method, which is the prevailing convention in this literature [44, 31, 34]. The convention is defensible when the solvers compared are deterministic or nearly so, and misleading otherwise, because a single run reports one draw from a distribution whose spread is never shown. Since the solvers of this family are stochastic, through their random restarts, their perturbations or both, we repeated all of them over 5 random seeds on all 31 B+P instances at their main-table configurations, and Table S5 gives the per-family mean $\pm$ standard deviation.

First, the ordering within our own family is stable, in that iPAMM-L stays ahead of iPAMM on every family under seed averaging, and the overall mean of iPAMM is $0.102 \pm 0.006\%$. The single-seed entries of Table 1 sit inside the seed spread on B2500 to P5000. On P6000 and P7000 they read 0.114% and 0.124% against seed means of 0.169% and 0.151%, both of which Table S5 reports.

Second, mean-field annealing is not reliable at this scale. On four of the five seeds MFA returns its tabled value exactly (0.144% on P5000 and 0.119% on P7000), while on the fifth it fails. At seed 2 on p5000.1 it returns an objective of $247{,}456$ against a best-known value of $8{,}559{,}680$, a gap of 97.1%, and at seed 1 on p7000.3 it returns a gap of 96.2%. These failures are reproducible and are not caused by numerical overflow. The returned assignment is near-balanced, with 2485 ones out of 5000 against 3124 on the converged runs. The mean-field trajectory never leaves the neighbourhood of the uniform state before the temperature ramp ends. Averaged over seeds, this places MFA at $4.02 \pm 7.75\%$ on P5000 and at $6.52 \pm 12.81\%$ on P7000.

These failures are the main empirical argument for a guarantee at the level of the individual update. MFA carries no such guarantee, and 2 of its 55 runs at $n \geq 5000$ are unusable, each failure being unbounded and invisible to a single published run. The terminal phase of iPAMM ascends the exact objective at every step by Proposition 3.1, so no run can end below the quality of its own initialization and hardened archive, and across all 31 instances and 5 seeds its overall standard deviation is 0.006 points. A practitioner choosing a solver of this family on the basis of a single published run should therefore be aware that the mean-field entry carries a heavy tail.

Table S5: Seed variance over 5 seeds on all 31 B+P instances, per family. **Bold** = best, underlined = second best.

Algorithm	B2500	P3000	P4000	P5000	P6000	P7000
	GAP (%) ↓, mean ± std over 5 seeds					
MFA (1989) [45]	0.082 ± 0.000	0.183 ± 0.001	0.154 ± 0.000	4.020 ± 7.752	0.160 ± 0.001	6.522 ± 12.805
iPAMM (ours)	0.055 ± 0.010	0.089 ± 0.010	0.101 ± 0.019	0.140 ± 0.021	0.169 ± 0.036	0.151 ± 0.013
iPAMM-L (ours)	**0.047** ± 0.008	**0.064** ± 0.020	**0.082** ± 0.016	**0.115** ± 0.021	**0.133** ± 0.037	**0.131** ± 0.022

H.5 SPARSE GSET MAX-CUT INSTANCES

Our claims target dense QUBO and every suite in the main paper is dense, and since several of the recent competitors were designed for sparse max-cut, we also ran seven sparse Gset max-cut instances ($n = 800$ to $10{,}000$) [52], converted to QUBO by the standard transformation so that the cut equals $\mathbf{x}^\top \mathbf{Q}\mathbf{x}$. Table S6 reports the approximation ratios to the best-known cuts.

First, the learned classifier does not transfer at all. VCM50 evaluated at depth 100 returns the all-zeros assignment on every sparse instance, for a cut of exactly zero. A random split does better, scoring $9{,}602$ out of $11{,}624$ on G1 (82.6%). The model was trained on dense random $\mathbf{Q}$ with entries in $[-100, 100]$. On a sparse max-cut matrix, whose diagonal carries the weighted degree and whose off-diagonal part is overwhelmingly zero, the value features fall entirely on one side of the threshold of the classifier. Our study exhibits here, in its most pronounced form, the transfer limitation that the dense benchmarks conceal. A learned solver that is competitive in-distribution can thus fail outright, whereas every other method in the table stays between 92% and 100%.

Second, the annealed family again occupies the top of the table, and the central calibration of the paper holds outside the dense regime in which it was established. The two annealed methods are complementary here, since seeding iPAMM with the MFA output puts it ahead of MFA on every instance of the set and makes it the leading method almost throughout, reaching the best-known cut

exactly on G1 and staying on a par with the greedy BGF baseline on G50, while from a random ensemble iPAMM remains competitive with MFA and is level with it on G14.

Third, the margin iPAMM holds on the dense suites narrows on sparse instances when it starts from a random ensemble, and our own theory says why. The curvature shift is set by $\mu \leq \lambda_{\min}(\mathbf{Q})$, and the inertia term of Section 3.2 scales as $|\hat{\mu}| = \Theta(\sigma\sqrt{n})$; on a sparse $\mathbf{Q}$ the spectral bound is loose relative to the objective scale, so a larger share of coordinates is frozen at each step of the schedule and the cycle explores less per unit budget. The prediction that follows is that iPAMM needs a more informative starting ensemble here, and the experiments confirm it, since seeding the initial ensemble with the MFA output (iPAMM+MFA) recovers 1.1 to 2.5 points on the three largest instances and lifts the method to the top of the table recorded above. The theory thus identifies both the regime in which the schedule is budget-limited and the corresponding remedy.

Table S6: Sparse Gset max-cut instances. Approximation ratio to the best-known cut; higher is better. VCM50 is evaluated here at depth 100.

Algorithm	G1	G14	G22	G43	G50	G55	G70
n	800	800	2000	1000	3000	5000	10000
	approximation ratio to best known (%)↑						
MFA (1989) [45]	<u>99.85</u>	<u>99.02</u>	<u>99.57</u>	<u>99.61</u>	98.88	<u>98.68</u>	<u>98.62</u>
BGF (2024) [34]	97.67	96.18	96.06	96.89	**100.00**	93.51	92.75
VCM50 (2024) [34]	0.00	0.00	0.00	0.00	0.00	0.00	0.00
iPAMM (ours)	99.23	<u>99.02</u>	98.91	99.43	97.21	97.75	96.17
iPAMM-L (ours)	99.23	98.99	99.00	99.43	98.10	97.95	96.94
iPAMM+MFA (ours)	**100.00**	**99.12**	**99.81**	**99.71**	<u>98.95</u>	**98.87**	**98.63**

H.6 Module ablation and parameter studies of the learned baseline

Since we train the learned baseline ourselves, we document its configuration with the ablation and parameter studies that are standard for architectures of this kind [31, 34], so that the strength of the baseline we compare against can be judged independently. Figure S3 gives them at $n = 50$, around the configuration used throughout, $\alpha = 4$, $h = 128$ and $d = 40$. Panel (a) removes each of ReLU, the two tanh saturations, and the memory units M_1 and M_2 from the iteration of the depth value network; panel (b) varies the input-scaling constant $\alpha \in \{\text{none}, 2, 4, 6\}$; and panel (c) varies the hidden size $h \in \{32, 64, 128, 256\}$ and the training depth $d \in \{10, 40, 100\}$. The scaling function is by far the most critical ingredient, because without it the training gap settles at roughly 10.7%, whereas $\alpha \in \{2, 6\}$ matches $\alpha = 4$ (0.195–0.201%). Every module except tanh-1 contributes, removing M_1 costing 0.296%, ReLU 0.279%, M_2 0.268% and tanh-2 0.250%, against 0.201% for the complete model, while removing tanh-1 is free at this size (0.187%). The hidden size peaks at $h = 128$, the values being 0.391, 0.226, 0.201 and 0.273% for $h = 32$, 64, 128 and 256. The training depth behaves as a threshold, since $d = 10$ is insufficient (roughly 13%), $d = 40$ is best, and $d = 100$ trains stably but generalizes worse (0.471%). This is consistent with the depth-extension phenomenon of Figure S4, by which additional depth is best added at test time.

H.7 Batch-OFI efficiency and BGF initialization

The closed-form increments (3) are the primitive on which BGF, our hardening step and the greedy label oracle all rest, so it is worth quantifying what the batched evaluation of (3) buys. Table S7 compares the traditional greedy flip (TGF), which evaluates the increments serially per instance and per variable, against the batched BGF at batch sizes 1, 4 and 20, on 20 instances per size. The batched calculation delivers speedups of several orders of magnitude, which is what makes greedy hardening affordable inside every cycle of iPAMM.

The second study concerns the starting point. Since BGF terminates at the first 1-opt point it reaches, its output depends on where it begins, and Table S8 reports it started from initial solutions containing between 0 and 100% ones. The all-ones start, used throughout, is the best in both quality and time, and it is also the initialization of the first column of the restart ensemble of PAMM in Section 3.4.

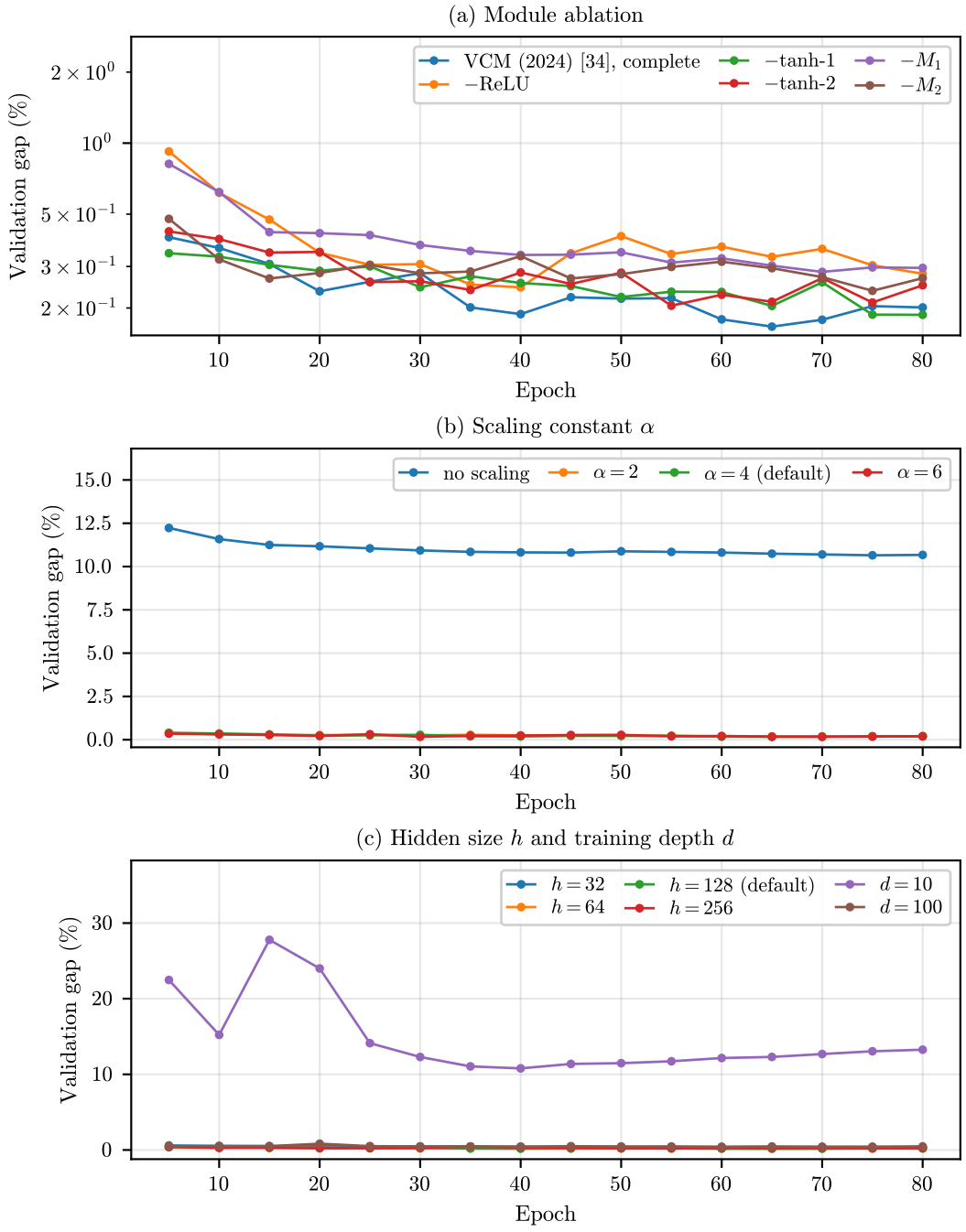

Figure S3: Module ablation and parameter studies of the learned baseline at $n = 50$. Validation gap against training epoch.

H.8 Significance tests

Averages over ten or twenty instances can be moved by a single outlier, so the ordering of Table 1 should not be read from the means alone. Table S9 therefore reports the Wilcoxon signed-rank test [54] over the 31 paired per-instance gaps of the B and P sets, which is the standard non-parametric instrument when heuristic solvers are compared over a common instance set [52, 34]. The test pairs the two solvers instance by instance, ranks the absolute differences and tests whether the signed ranks are symmetric about zero, so it uses only the ordering of the per-instance differences and makes no

Table S7: Efficiency of the batched increment evaluation. Total time for 20 instances.

Instances	TGF	BGF (2024) [34] batch 1	batch 4	batch 20
		total time (s) ↓		
G20(20)	0.31	0.03	0.01	**0.00**
G50(20)	1.94	0.07	0.02	**0.00**
G100(20)	9.83	0.18	0.05	**0.01**
G200(20)	55.59	0.44	0.11	**0.02**

Table S8: BGF (2024) [34] from different initial solutions. Mean OFV and total time over 100 instances per size.

Instances	ones in the initial solution 0%	25%	50%	75%	100%
			mean OFV ↑		
G100(100)	33,163	32,993	33,313	33,604	**33,777**
G500(100)	376,165	377,203	377,142	377,646	**379,486**
G1000(100)	1,067,545	1,067,696	1,069,752	1,070,399	**1,073,446**
Instances	ones in the initial solution 0%	25%	50%	75%	100%
			total time (s) ↓		
G100(100)	0.02	0.01	0.01	0.01	0.01
G500(100)	0.08	0.07	0.06	0.05	0.04
G1000(100)	0.17	0.14	0.12	0.11	0.09

assumption about their distribution. The rows are ordered chronologically with our own methods at the bottom, and every entry is tested against iPAMM, which occupies the last row.

The outcome is unambiguous, in that iPAMM is significantly better than every learned, hybrid and relaxation-based entry, at $p < 10^{-7}$ throughout, and it also beats MFA and iSCO significantly, at $p \approx 5 \times 10^{-3}$; the weaker significance in the latter two comparisons reflects the smaller effect size, since these are the only competitors whose mean gaps are of the same order as ours. Within our own family the ordering is likewise significant, with iPAMM-L ahead of iPAMM at $p \approx 5 \times 10^{-3}$ and the improvement concentrated on the B2500 to P5000 families, where the extra cycles have room to act.

H.9 Value-feature visualization

The learned baseline improves when a network trained at one depth is evaluated at a larger one (Figure S4), and the representation shows what that gain consists of. Figure S5 shows t-SNE embeddings of the value features of our trained VCM50 on the first B2500 instance across testing depths. At shallow depth the features organize into filament-like structures, in which the two classes are interleaved and the threshold of the classifier cuts across them; as the testing depth grows the filaments separate progressively into two clusters aligned with the eventual binary decision. Additional depth therefore acts as further rounds of message passing that sharpen an already-formed decision boundary without adding capacity, which is consistent with the fact that the improvement is obtained at test time and with weights that were never trained at that depth. The ceiling this implies is the one the sparse instances of Appendix H.5 reach, where no amount of test-time depth recovers a boundary that the training distribution never placed.

I Further Experiments on the Proposed Solver

This appendix reports three of the studies summarized in Section 4, namely size generalization across two orders of magnitude in n, the ablations of the PAMM engine, and the PAMM engine used as the label oracle inside the trainer of the learned baseline.

Table S9: Wilcoxon signed-rank tests over the 31 B and P instances. Our methods are grouped at the bottom, with iPAMM as the reference.

Algorithm	mean GAP (%) ↓	p-value vs. iPAMM
MFA (1989) [45]	0.131	5.09×10^{-3}
SDP-GW (1995/2003) [37, 38]	4.097	9.31×10^{-10}
Diag (2020) [26]	98.846	9.31×10^{-10}
PI-GNN (2022) [31]	2.872	9.31×10^{-10}
iSCO (2023) [46]	0.143	5.44×10^{-3}
BGF (2024) [34]	0.864	9.31×10^{-10}
VCM10 (2024) [34]	1.252	9.31×10^{-10}
VCM20 (2024) [34]	0.823	9.31×10^{-10}
VCM50 (2024) [34]	0.626	9.31×10^{-10}
VCM100 (2024) [34]	0.470	9.31×10^{-10}
VCM10-BGF (2024) [34]	0.395	9.31×10^{-10}
VCM20-BGF (2024) [34]	0.332	9.31×10^{-10}
VCM50-BGF (2024) [34]	0.273	1.86×10^{-9}
VCM100-BGF (2024) [34]	0.249	9.31×10^{-10}
QRF-GNN (2024) [32]	0.674	9.31×10^{-10}
PQQA (2024) [47]	0.318	2.79×10^{-9}
Deep k-grouping (2025) [33]	6.686	9.31×10^{-10}
PDBO (2025) [48]	5.045	9.31×10^{-10}
mQO (2026) [49]	0.421	9.31×10^{-10}
MM (ours)	0.946	9.31×10^{-10}
PAMM (ours)	0.934	9.31×10^{-10}
PAMM-GF (ours)	0.536	9.31×10^{-10}
VCM50+PAMM (ours)	0.185	1.93×10^{-7}
iPAMM-L (ours)	**0.069**	5.46×10^{-3}
iPAMM (ours)	<u>0.097</u>	–

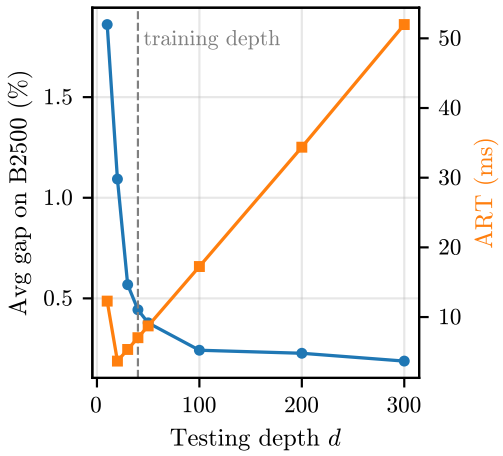

Figure S4: Depth extension of VCM50 (2024) [34] on B2500, where quality improves monotonically while the time grows linearly.

I.1 Size generalization on G sets

The B and P suites are large but narrow in size, spanning only a factor of three in n, so Table S10 repeats the comparison off-benchmark and across two orders of magnitude, on generated instances with n from 20 to 1000 at 100 instances per size. Three points emerge. First, size is not where the learned classifier breaks down, since a model trained at $n = 10$ stays within roughly 0.9% up to $n = 1000$; the dense-to-sparse transfer examined in Appendix H.5 is where it does. Second, iPAMM beats the strong per-instance reference of Section 4 at every size, with gaps from -0.007 to

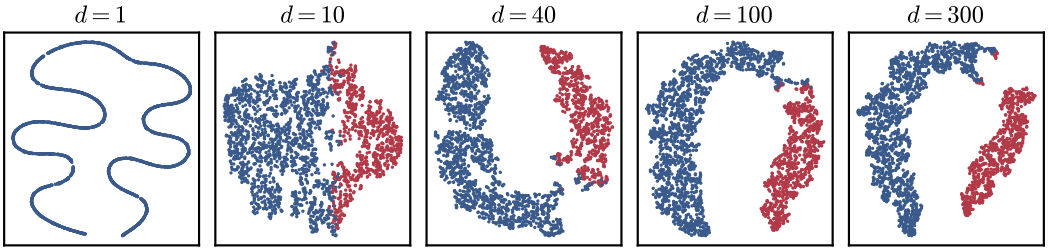

Figure S5: t-SNE of the value features of VCM50 (2024) [34] on the first B2500 instance across testing depths d, the number of value-network iterations run at test time. Colors give the final classification.

-0.277% that become more negative as n grows, so it finds strictly better solutions than the reference protocol on average and its advantage widens with the problem size; it also stays ahead of every learned entry and hybrid throughout. Third, Figure S6 plots the runtime scaling and separates the two computational regimes cleanly, since the matvec-bound methods, among them the value classifiers and iPAMM, grow slowly with n because their work is a fixed number of dense products, whereas the flip-sequential methods such as BGF pay an $O(n)$ serial depth that no amount of parallel hardware removes.

Table S10: Generated G instances (100 per size). Gaps against the per-instance reference of Section 4; negative means the method beat it.

Algorithm	G20		G50		G100	
	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓
Diag (2020) [26]	88.856	0	91.087	0	93.554	0
BGF (2024) [34]	1.812	1	1.529	4	1.578	6
VCM10 (2024) [34]	0.402	3	0.614	3	0.688	3
VCM20 (2024) [34]	0.165	3	0.380	3	0.293	3
VCM50 (2024) [34]	0.278	3	0.243	3	0.316	3
VCM100 (2024) [34]	0.620	3	0.289	3	0.315	3
VCM10-BGF (2024) [34]	0.030	3	0.235	3	0.241	3
VCM20-BGF (2024) [34]	0.057	3	0.125	3	0.110	3
VCM50-BGF (2024) [34]	0.075	3	0.110	3	0.129	3
VCM100-BGF (2024) [34]	0.102	3	0.090	3	0.122	3
iPAMM (ours)	**-0.021**	128	**-0.007**	129	**-0.008**	133
iPAMM-L (ours)	**-0.021**	239	**-0.007**	245	**-0.008**	259
Algorithm	G200		G500		G1000	
	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓	GAP (%)↓	ART (ms)↓
Diag (2020) [26]	94.355	0	97.707	0	97.410	0
BGF (2024) [34]	1.398	13	0.844	33	0.860	67
VCM10 (2024) [34]	0.866	3	0.868	3	0.899	4
VCM20 (2024) [34]	0.505	3	0.531	3	0.551	3
VCM50 (2024) [34]	0.304	3	0.326	3	0.325	3
VCM100 (2024) [34]	0.236	3	0.243	3	0.198	3
VCM10-BGF (2024) [34]	0.281	4	0.220	5	0.136	7
VCM20-BGF (2024) [34]	0.173	3	0.110	4	0.052	6
VCM50-BGF (2024) [34]	0.080	3	0.035	4	-0.017	6
VCM100-BGF (2024) [34]	0.057	3	0.016	4	-0.067	5
iPAMM (ours)	**-0.055**	145	-0.156	181	-0.277	223
iPAMM-L (ours)	**-0.055**	291	**-0.158**	385	**-0.286**	526

I.2 Ablations of the PAMM engine

Warm starts. The analysis of Section 3.2 predicts, and the experiments confirm, that at these sizes MM held at the fixed admissible shift $\mu = \hat{\mu}$ leaves a warm start unchanged, since the inertia $|\hat{\mu}| = \Theta(\sigma\sqrt{n})$ holds every coordinate in place and the starting point is already a fixed point of (11). iPAMM therefore improves a warm start by perturbing it and driving it through the schedule again, through its graduated perturbations and the ramp $\mu_t : 0 \to \hat{\mu}$.

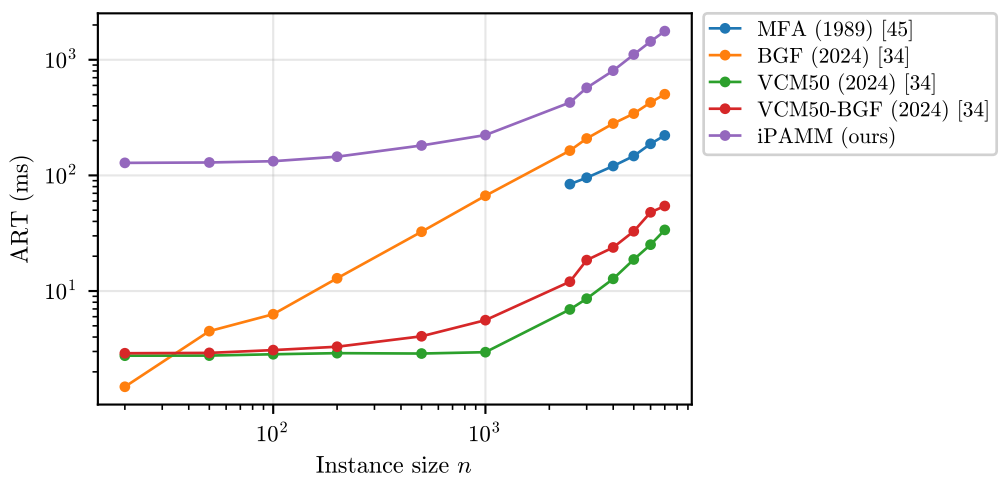

Figure S6: Runtime scaling (log–log) across $n \in [20, 7000]$, over the G sets and the B and P benchmarks.

Figure S7 examines, on B2500, the PAMM engine on which iPAMM is built. The schedule is essential, since with $\rho = 0$, that is with a constant $\mu = \hat{\mu}$, a single pass settles at a gap of 20–25% regardless of the budget, the curvature–stationarity trade-off in action, whereas any substantial ramp ($\rho \in [0.3, 0.9]$) recovers gaps below 3% at $T = 50$, below 2% by $T = 100$, and of 0.57–0.74% at $T = 1000$, with a mild sensitivity within that range, the values $\rho \in [0.7, 0.9]$ being the best. The budget also saturates within a cycle, as the quality of a single pass improves roughly logarithmically in T, and iPAMM therefore spends its budget on repeated short cycles. Finally, parallel restarts are inexpensive, as moving from 1 to 128 of them lowers the gap of a hardened single pass from 0.54% to 0.33% at little wall-clock cost, the ensemble advancing as one batched matrix product, and greedy hardening adds a consistent 0.15 to 0.25 points by closing the residual stationarity slack of (13). Appendix G completes the picture with the component ablation of iPAMM (single-pass variants, hardening removal, VCM seeding).

I.3 MM-GST, the PAMM engine inside the trainer

We replace BGF with PAMM (warm-started at the VCM output, $T = 150$) as the label-improvement oracle inside the greedy-guided self trainer (GST). Figure S8 collects the training curves, panel (a) showing that the validation gap of the baseline improves steadily with the training size and settles within an epoch budget that is the same for all four models, and panel (b) comparing the trainers at $n = 50$. MM-GST tracks GST closely, with a final validation gap of 0.24% against 0.20%, both are ahead of the unsupervised objective-as-loss trainer UnS (0.31%), and the historical-best archive matters, as LGF, which drops it, is less stable. What the comparison establishes is that the self-trainer is indifferent to which improvement heuristic supplies its labels, since two oracles of quite different construction carry it to the same place, and Appendix K quantifies the oracle time and the label quality per 512-instance batch that make them interchangeable at these sizes. The engine's own speed advantage belongs to a different regime, that of benchmark scale, where it acts as the core of iPAMM in Table 1.

J Proof Details and Additional Analysis

This appendix gives the solver in pseudocode, describes its three practical components and its cost, and proves the results stated in Section 3, namely Proposition 3.1, Corollary 3.2 and Proposition 3.3. It closes with the two remarks on the choice of the shift that Sections 3.2 and 3.4 refer to.

The three practical components of PAMM. The three components introduced in Section 3.4 are the following.

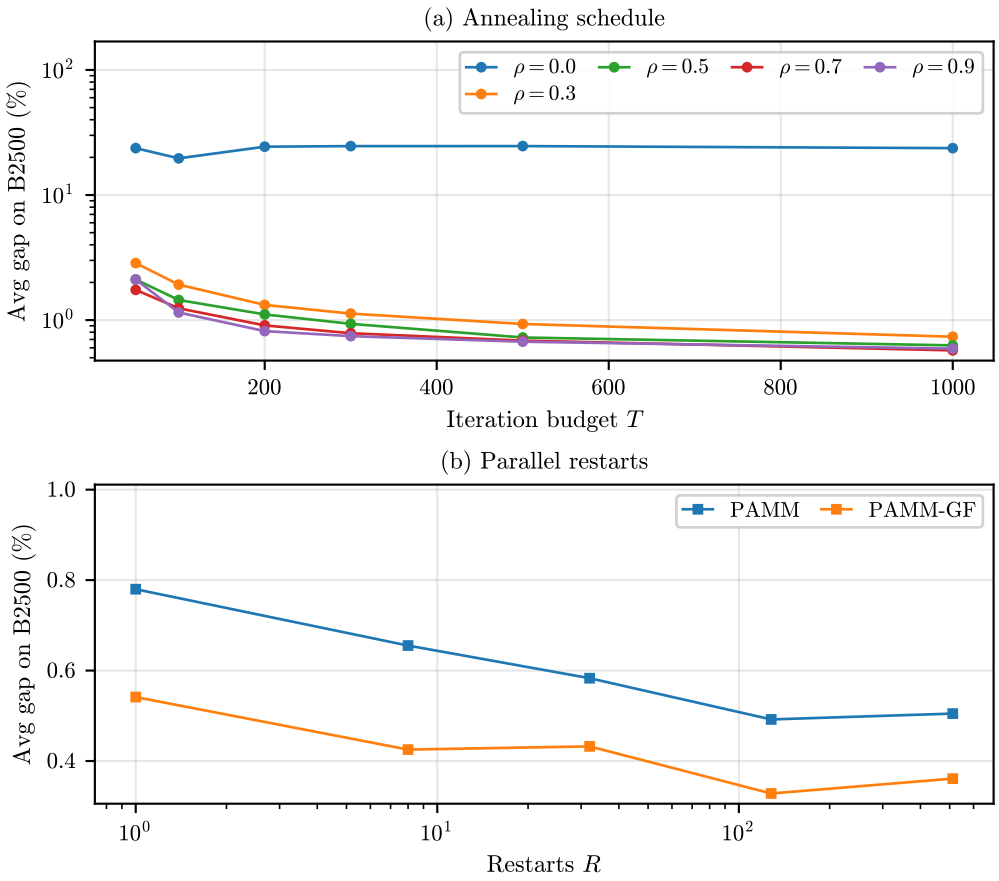

Figure S7: Ablations of the PAMM engine on B2500. (a) Gap against the budget T for several ramp fractions ρ. (b) Gap against the number of parallel restarts.

Algorithm S1 PAMM (per batch of instances; all operations batched on GPU)

Require: $\mathbf{Q}$, budget T, ramp fraction ρ, restarts R

$\hat{\mu} \leftarrow$ power-iteration lower bound on $\lambda_{\min}(\mathbf{Q})$ (with safety margin)

$\mathbf{X} \leftarrow [\mathbf{1},\ \text{Bernoulli}(1/2)^{\times(R-1)}] \in \{0,1\}^{n \times R}$

for $t = 0, \ldots, T-1$ **do**

  $\mu_t \leftarrow \hat{\mu} \cdot \min\{t/\lceil \rho T \rceil,\ 1\}$

  $G \leftarrow 2QX + \mu_t(\mathbf{1} - 2\mathbf{X})$ ▷ one batched matrix product

  $\mathbf{X} \leftarrow \mathbf{1}[G > 0]$ elementwise, keeping $\mathbf{X}$ where $G = 0$

  if $t \geq \lceil \rho T \rceil$ and $\mathbf{X}$ unchanged **then break** ▷ MM fixed point reached

end for

return per-instance best column of $\mathbf{X}$ by OFV (optionally BGF-hardened)

- *Parallel restarts.* R initial points (the all-ones vector plus $R-1$ Bernoulli$(1/2)$ draws) are stacked as an extra tensor dimension, so a whole restart ensemble advances with a single $n \times n$ by $n \times R$ product per iteration; the best final OFV per instance is returned. On a GPU this costs almost nothing until R saturates the matrix-multiply units.
- *Greedy hardening (PAMM-GF).* Because (13) is weaker than 1-opt, we optionally pass the final iterate to BGF, which closes the residual stationarity slack; started from a near-optimal point it terminates in a few flips.
- *Warm starts.* Any classifier output can serve as an initial point once it seeds the restart ensemble. With $\mu = \hat{\mu}$ held fixed, the inertia that guarantees ascent keeps a warm start at its initial point at benchmark scale, because the fixed-point condition (13) is satisfied immediately, as Appendix I.2

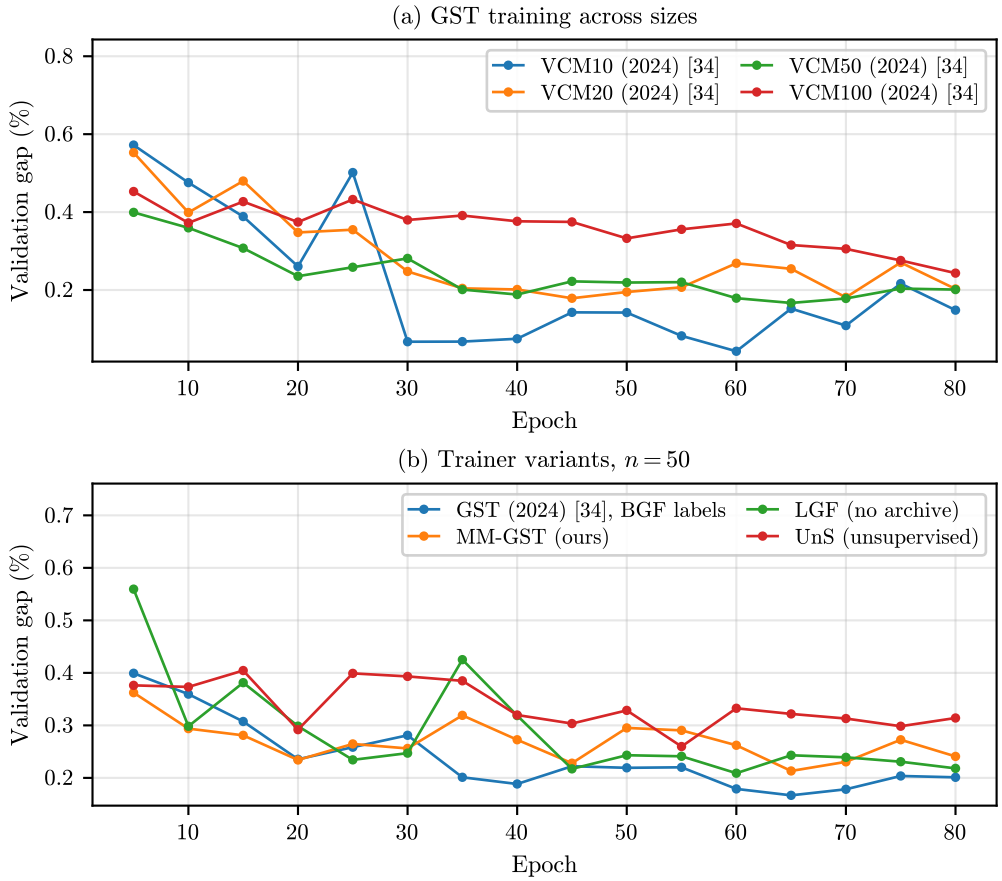

Figure S8: (a) Validation-gap curves across training sizes. (b) Trainer variants at $n = 50$.

confirms. Specifically, the classifier output together with lightly perturbed copies replaces the random initial columns and is driven through the schedule as usual. These are studied as the VCM-seeded variants in Appendix G. Inside the GST trainer, the warm-started PAMM serves as the label-improvement oracle, a variant we call MM-GST.

Complexity comparison. At the budget used here, $T = 300$, the $O(TRn^2)$ operations of Appendix E amount to a few hundred batched dense products per instance. The semidefinite relaxation, in contrast, lifts the problem to $n \times n$ matrix variables and requires work of order $O(n^3)$ per iteration, or else low-rank Burer–Monteiro factorizations followed by hyperplane rounding, whereas our iteration operates directly on the original binary variables. Unlike learned classifiers, it requires no training, transfers to any size or distribution by construction, and returns a solution that is provably 1-opt after hardening.

Boundedness of the objective. Before proving the ascent results we record that the objective of (1) is bounded on its feasible set, which is what turns a monotone sequence into a convergent one. For every $\mathbf{x} \in \{0,1\}^n$, the Rayleigh bounds $\lambda_{\min}(\mathbf{Q})\,\mathbf{x}^\top\mathbf{x} \le \mathbf{x}^\top\mathbf{Q}\mathbf{x} \le \lambda_{\max}(\mathbf{Q})\,\mathbf{x}^\top\mathbf{x}$ combine with the cube identity (4) and with $0 \le \mathbf{1}^\top\mathbf{x} \le n$ to give

$$f_{\text{lo}} \triangleq n\min\{\lambda_{\min}(\mathbf{Q}), 0\} \le f(\mathbf{x}) \le n\max\{\lambda_{\max}(\mathbf{Q}), 0\} \triangleq f_{\text{up}}. \tag{S6}$$

Both bounds are finite and independent of $\mathbf{x}$, so f is bounded on $\{0,1\}^n$; moreover f takes at most 2^n distinct values there, so its range is a finite subset of $[f_{\text{lo}}, f_{\text{up}}]$. ■

Proof of Proposition 3.1 (monotone ascent, convergence of the objective and of the iterates). *(a) Ascent.* Write $L_t(\mathbf{x}) \triangleq g(\mathbf{x} \mid \mathbf{x}_t)$ for the minorant (8). By construction $f(\mathbf{x}_{t+1}) \ge L_t(\mathbf{x}_{t+1})$ and

$f(\mathbf{x}_t) = L_t(\mathbf{x}_t)$, which are the two minorizer conditions (S2) and (S3), and $\mathbf{x}_{t+1}$ maximizes L_t over the cube by (10). The generic chain (S5) therefore gives $f(\mathbf{x}_{t+1}) \geq f(\mathbf{x}_t)$.

(b) Strictness. Each coordinate that changes in (11) strictly increases the separable objective L_t, since a coordinate with $g_{t,i} > 0$ and $x_{t,i} = 0$ contributes $+g_{t,i} > 0$, and one with $g_{t,i} < 0$ and $x_{t,i} = 1$ contributes $-g_{t,i} > 0$, while unchanged coordinates contribute zero. Hence $\mathbf{x}_{t+1} \neq \mathbf{x}_t$ implies $L_t(\mathbf{x}_{t+1}) > L_t(\mathbf{x}_t)$ and thus $f(\mathbf{x}_{t+1}) > f(\mathbf{x}_t)$.

(c) Convergence of the objective sequence. By *(a)* the sequence $\{f(\mathbf{x}_t)\}_{t\geq 0}$ is non-decreasing, and by (S6) it is bounded above by f_{up}. A non-decreasing sequence that is bounded above converges, so $f(\mathbf{x}_t) \to f^\star$ for some $f^\star \in [f(\mathbf{x}_0), f_{\mathrm{up}}]$. Because the range of f on the cube is a finite set, the convergence is not merely asymptotic, since the increments $f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t)$ are either zero or bounded below by the smallest positive difference between two attainable values of f, so only finitely many strict increases can occur and the limit $f^\star$ is attained after finitely many steps.

(d) Convergence of the iterate sequence. Let $T^\star$ be the index after which no strict increase occurs, which exists by *(c)*. For $t \geq T^\star$ we have $f(\mathbf{x}_{t+1}) = f(\mathbf{x}_t)$, and by *(b)* this forces $\mathbf{x}_{t+1} = \mathbf{x}_t$. The update is deterministic under the keep-on-tie convention, so the whole sequence is constant beyond $T^\star$, that is $\mathbf{x}_t = \mathbf{x}^\star$ for all $t \geq T^\star$. The sequence of iterates therefore converges, and it does so in finitely many steps; since no configuration can repeat before $T^\star$, the crude bound $T^\star \leq 2^n$ holds, while in practice tens of steps suffice. ■

Proof of Corollary 3.2 (convergence to stationary points). *(i) Finite convergence.* By Proposition 3.1, $f(\mathbf{x}_{t+1}) > f(\mathbf{x}_t)$ whenever $\mathbf{x}_{t+1} \neq \mathbf{x}_t$. As f takes finitely many values on $\{0,1\}^n$, only finitely many strict increases can occur, so there is a finite $T^\star$ with $\mathbf{x}_{T^\star+1} = \mathbf{x}_{T^\star} =: \mathbf{x}^\star$; the update is deterministic and keep-on-tie, so $\mathbf{x}_t = \mathbf{x}^\star$ for all $t \geq T^\star$.

Stationarity of the limit. We first derive the identity (12). Fix $\mathbf{x} \in \{0,1\}^n$ and i. From the definition of the field in (11), $g_i = 2(\mathbf{Q}\mathbf{x})_i + \mu(1 - 2x_i)$. Multiplying by $(1 - 2x_i)$ and using $(1 - 2x_i)^2 = 1$,

$$(1-2x_i)\, g_i \;=\; 2(1-2x_i)(\mathbf{Q}\mathbf{x})_i + \mu \;=\; \big[\, q_{ii} + 2(1-2x_i)(\mathbf{Q}\mathbf{x})_i \,\big] - q_{ii} + \mu \;=\; \mathrm{OFI}_i(\mathbf{x}) - q_{ii} + \mu,$$

where the last step is the closed form (3) of the flip increment. This proves (12). Now observe that a coordinate of the update (11) changes at $\mathbf{x}$ exactly when $x_i = 0$, $g_i > 0$ or $x_i = 1$, $g_i < 0$, i.e. exactly when $(1 - 2x_i)g_i > 0$. Since $\mathbf{x}^\star$ is a fixed point, no coordinate changes, so $(1 - 2x_i^\star)g_i^\star \leq 0$ for all i, which by the identity is precisely $\mathrm{OFI}_i(\mathbf{x}^\star) \leq q_{ii} - \mu$, i.e. (13).

(ii) Hardening. BGF flips only coordinates with $\mathrm{OFI}_k > 0$, and each such flip increases f by exactly $\mathrm{OFI}_k(\mathbf{x}) > 0$. By finiteness of f on the cube, BGF performs finitely many flips and monotonicity gives $f(\mathbf{x}^{\star\star}) \geq f(\mathbf{x}^\star)$. Its stopping rule is $\max_i \mathrm{OFI}_i \leq 0$, so at termination $\mathrm{OFI}_i(\mathbf{x}^{\star\star}) \leq 0$ for all i; equivalently, for every $\mathbf{y}$ with Hamming distance 1 from $\mathbf{x}^{\star\star}$, say differing at coordinate i, $f(\mathbf{y}) = f(\mathbf{x}^{\star\star}) + \mathrm{OFI}_i(\mathbf{x}^{\star\star}) \leq f(\mathbf{x}^{\star\star})$, which is 1-opt stationarity.

Consequence for iPAMM. Every iPAMM cycle ends with the terminal phase at $\mu = \hat{\mu} \leq \lambda_{\min}(\mathbf{Q})$ followed by BGF hardening of the columns that enter the archive, so *(i)* and *(ii)* apply to each of them; the archive and the incumbent only ever replace entries by ones of larger f, preserving the property. This is the claim made in Section 3.5. ■

Proof of Proposition 3.3 (Lyapunov function along the schedule). *(a) Monotone descent of the energy.* A direct computation gives $E_\mu(\mathbf{x}_t, \mathbf{x}_{t+1}) - E_\mu(\mathbf{x}_{t+1}, \mathbf{x}_{t+2}) = (\mathbf{x}_{t+2} - \mathbf{x}_t)^\top \big(\mathbf{W}\mathbf{x}_{t+1} + \mathbf{b}\big)$ with $\mathbf{W} = 2(\mathbf{Q} - \mu\mathbf{I})$ and $\mathbf{b} = \mu\mathbf{1}$, using $\mathbf{W} = \mathbf{W}^\top$. By the update rule, whenever $(\mathbf{W}\mathbf{x}_{t+1} + \mathbf{b})_i > 0$ we have $x_{t+2,i} = 1 \geq x_{t,i}$, whenever $(\mathbf{W}\mathbf{x}_{t+1} + \mathbf{b})_i < 0$ we have $x_{t+2,i} = 0 \leq x_{t,i}$, and ties contribute zero; hence every term of the inner product is non-negative, and is positive for each coordinate where $x_{t+2,i} \neq x_{t,i}$ and the field is nonzero. With keep-on-tie, $\mathbf{x}_{t+2} \neq \mathbf{x}_t$ forces at least one such coordinate, so the energy strictly decreases.

(b) Boundedness of the energy. For all $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, the Cauchy–Schwarz inequality together with $\|\mathbf{u}\|, \|\mathbf{v}\| \leq \sqrt{n}$ gives $|E_\mu(\mathbf{u}, \mathbf{v})| \leq 2n \big(\|\mathbf{Q}\|_2 + |\mu|\big) + 2n|\mu| < \infty$, so E_μ is bounded on $\{0,1\}^n \times \{0,1\}^n$ and, being a function on a finite set, attains finitely many values.

(c) Finite convergence to a fixed point or a two-cycle. By *(a)* the sequence $E_\mu(\mathbf{x}_t, \mathbf{x}_{t+1})$ is non-increasing and by *(b)* it is bounded below, so it converges; and since it takes finitely many values, it is constant after finitely many steps. Once it is constant, *(a)* forces $\mathbf{x}_{t+2} = \mathbf{x}_t$ for every subsequent t,

so the trajectory is eventually periodic with period one or two, that is, a fixed point or a two-cycle. The even and odd subsequences of $(\mathbf{x}_t)$ therefore each converge.

(d) Diagonal identity and the absence of two-cycles. The identity on the diagonal follows from (4): $E_\mu(\mathbf{x}, \mathbf{x}) = -2\mathbf{x}^\top \mathbf{Q}\mathbf{x} + 2\mu\mathbf{x}^\top\mathbf{x} - 2\mu\mathbf{1}^\top\mathbf{x} = -2f(\mathbf{x})$. Finally, suppose $\mu \le \lambda_{\min}(\mathbf{Q})$ and let $\mathbf{x} \to \mathbf{y} \to \mathbf{x}$ be a two-cycle with $\mathbf{x} \neq \mathbf{y}$. Applying Proposition 3.1 to the step $\mathbf{x} \to \mathbf{y}$ gives $f(\mathbf{y}) > f(\mathbf{x})$, and applying it to the step $\mathbf{y} \to \mathbf{x}$ gives $f(\mathbf{x}) > f(\mathbf{y})$, a contradiction. Hence no genuine two-cycle exists in the admissible range, and by *(c)* the iteration converges to a fixed point. ■

Tightest scalar curvature. Among all μ for which $\mathbf{x}^\top(\mathbf{Q} - \mu\mathbf{I})\mathbf{x}$ is convex, $\mu = \lambda_{\min}(\mathbf{Q})$ is maximal, and by (13) it yields the strongest fixed-point condition of the family; smaller (safer) μ weakens it and so can only enlarge the fixed-point set. In this precise sense the trade-off of Section 3.2 is unavoidable for scalar diagonal shifts, motivating either a schedule on μ, which is the route taken here, or minorizers built from non-scalar diagonal shifts.

Estimating $\lambda_{\min}$. We use two rounds of 60 power iterations. The first, on $\mathbf{Q}$, estimates $\sigma \approx |\lambda|_{\max}$, and the second, on $\sigma\mathbf{I} - \mathbf{Q}$ (whose dominant eigenvalue is $\sigma - \lambda_{\min}$), recovers $\lambda_{\min}$, to which we add a $-10^{-3}\sigma$ safety margin. The cost (120 mat-vecs) is shared across all restarts and is included in every reported ART. The safety margin is what keeps the bound conservative, and float64 audits over every run reported in this paper record no violation of monotonicity. Where an exact bound is preferred, a Lanczos factorization supplies one with rigorous error bounds at $O(n^2)$ extra cost per iteration, and it substitutes into the construction of Section 3.1 without changing anything else.

K Label-Oracle Microbenchmark

Table S11 times one label-improvement call on a 512-instance training batch, starting from identical VCM outputs, for the BGF oracle and the warm-started PAMM oracle at two budgets. Warm-started BGF terminates after a few flips, since the classifier output is already near 1-opt, and PAMM produces labels of essentially the same mean OFV at these sizes, which is what makes the two oracles interchangeable inside the trainer. The runtime advantages of the PAMM engine appear at benchmark scale, where it acts as a standalone solver (Table 1) and where the serial flip loop of BGF must traverse $O(n)$ flips from a cold start.

Table S11: Oracle cost and label quality per 512-instance batch (identical VCM warm starts).

Algorithm	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
			time (s) ↓		
BGF (2024) [34]	**0.011**	**0.003**	**0.017**	**0.012**	**0.022**
PAMM, $T = 150$ (ours)	0.027	0.016	0.041	0.145	0.546
PAMM, $T = 300$ (ours)	0.018	0.024	0.045	0.212	0.796
Algorithm	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
			mean OFV ↑		
BGF (2024) [34]	**12,130**	**34,410**	**96,726**	**384,899**	**1,085,769**
PAMM, $T = 150$ (ours)	12,127	34,402	96,718	384,810	1,085,384
PAMM, $T = 300$ (ours)	12,128	34,401	96,724	384,861	1,085,620

L Per-instance Results

Per-instance gaps for all methods on the 31 benchmark instances are included in the released JSON artifacts alongside the code, together with Wilcoxon signed-rank tests over the 31 paired gaps. Ranked by mean gap, the ordering is iPAMM-L (0.069%), iPAMM (0.097%), MFA (0.131%), iSCO (0.143%), then our single-pass hybrid VCM50+PAMM (0.185%), and after it the remaining competitors, from 0.249% (VCM100-BGF) to 98.8% (Diag), among which PQQA (0.318%) is the strongest learning-free entry and Deep k-grouping (6.686%) the weakest learned one, and Appendix H.8 gives the significance of every comparison.