Constrained QUBO Optimization with FM Surrogates + PROTES

A practical implementation guide, including constraint handling and benchmark design

Date: 2026-01-19

How to use this guide

This guide describes an FMQA-style black-box optimization loop where a Factorization Machine (FM) is trained as a quadratic surrogate (QUBO) and optimized using PROTES (a TT-based probabilistic optimizer). It focuses on constraint handling and benchmarking.

Document map

- Section 1: formalize the constrained black-box task.
- Section 2: explain why FM -> QUBO and why PROTES fits the loop.
- Section 3: the full algorithm (outer loop + constraint handling choices).
- Section 4: implementation details and code skeletons.
- Section 5: benchmark suite and experimental protocol.
- Section 6: practical tips and common failure modes.
- Section 7: references.

Quick start checklist

1. Implement constraint checker feasible(x) and (optionally) violation magnitude viol(x).
2. Train an FM regressor on feasible data to get Q (QUBO coefficients).
3. Define a scalar energy E(x) = QUBO(x) + rho * viol(x) (start with penalties).
4. Call PROTES with a batched objective f_batch(I) that returns E(x) for a batch.
5. Collect top-K candidates, filter by feasibility, evaluate the true oracle on feasible ones.
6. Append new samples and iterate; log best objective vs oracle calls.

Key hyperparameters (typical starting points)

Parameter	Typical value	Notes
FM rank (k)	8-64	Higher can fit richer interactions but may overfit with little data.
FM regularization	L2, tune	Tune on a validation split of the current dataset.
PROTES batch size (k)	128-512	Larger batches can stabilize updates but cost more compute.
PROTES elite count (k_top)	10-50	How many best samples drive the distribution update.
Outer-loop new points (K)	10-100	Feasible oracle evaluations per iteration (plus infeasible checks).
Penalty weight (rho)	10-1e4	Increase if feasibility rate is low; decrease if overly conservative.

Tip: The headings are set with Word styles (Heading 1/2/3). In Microsoft Word you can insert an automatic Table of Contents via: References -> Table of Contents.

1. Task definition

We consider a constrained black-box binary optimization problem. Each candidate solution is a binary vector x in {0,1}^d. A (possibly expensive) oracle returns a cost f(x) for feasible x.

Goal: find a feasible x* that minimizes f(x) using a limited number of oracle evaluations.

1.1 Inputs and outputs

- Decision variables: x = (x1,...,xd), xi in {0,1}.
- Feasibility constraints: g_j(x) <= 0 and/or h_k(x) = 0; or a feasibility checker feasible(x) -> {True, False}.
- Initial dataset D0: a set of (x, f(x)) pairs. In your setting these are valid/feasible configurations.
- Evaluation budget: total number of oracle calls allowed.
- Output: best feasible solution found, plus logs for benchmarking (objective vs calls, feasibility rate, runtime).

1.2 Why this is hard

- The search space is exponential (2^d).
- Constraints can make the feasible region tiny.
- The oracle is expensive, so we cannot evaluate many candidates.
- We want a solver that proposes good candidates while respecting feasibility.

2. High-level idea

We combine (i) a quadratic surrogate model trained from data and (ii) a high-performance discrete optimizer. The loop is similar to FMQA / FMA-style black-box optimization:

7. Train a Factorization Machine (FM) on the current dataset to predict the cost.
8. Convert the FM into a quadratic function (QUBO form): E_sur(x) = x^T Q x + const.
9. Optimize (or sample low-energy points) using PROTES, a tensor-train (TT)-based probabilistic optimizer.
10. Check constraints; evaluate the oracle on feasible candidates; add new labeled samples to the dataset.
11. Repeat until budget is exhausted or improvement stalls.

2.1 Why an FM becomes a QUBO

For binary inputs, an FM regression model has the form: y_hat(x) = w0 + sum_i w_i x_i + sum_{i<j} <v_i, v_j> x_i x_j. This is quadratic in x and therefore equivalent to a QUBO.

2.2 Why PROTES

PROTES (Probabilistic Optimization with Tensor Sampling) maintains a probability distribution over the discrete search space in TT format, samples batches of candidates, keeps the best ones, and updates the TT distribution to concentrate probability mass near good solutions.

3. The FM-PROTES constrained optimization algorithm

3.1 Data structures

- Df (feasible regression set): pairs (x, f(x)) for feasible x.
- Dc (constraint classification set): pairs (x, label) where label=1 if feasible, else 0.
- Optional: store constraint violation magnitudes for learning a continuous feasibility score.

Important: if your initial dataset contains only feasible samples, you will not have negative examples. You can collect infeasible examples during the iterations by evaluating constraints on proposed candidates (no oracle cost required).

3.2 Outer-loop pseudocode

```
Inputs: initial feasible dataset D0, constraint checker feasible(x), budget B
Hyperparams: iterations T, new points per iter K, PROTES batch k, k_top
Initialize Df = D0; Dc = {(x,1) for x in D0}
best = argmin_{(x,y) in Df} y
for t = 1..T:
  Train FM_reg on Df -> Q_t (QUBO matrix)
  (Optional) Train FM_cls / classifier on Dc -> p_t(feasible | x)
  Define scalar energy E_t(x):
     - penalty version: x^T Q_t x + rho_t * viol(x)
     - or feasibility-weighted: x^T Q_t x - alpha_t * log(p_t(feasible|x)+eps)
  Use PROTES to minimize E_t(x) (batched calls). Get a pool P of candidates.
  Select top-K (and optionally diverse) candidates from P.
  For each candidate x:
     if feasible(x):
        y = oracle_cost(x)   # counts toward budget B
        add (x,y) to Df; add (x,1) to Dc
        update best if y improves
     else:
        add (x,0) to Dc
  Stop if budget exhausted or no improvement in N_stop iters
return best
```

3.3 Constraint handling in the PROTES step

PROTES itself optimizes an unconstrained scalar function. To enforce constraints you can (A) add penalties, (B) bias or restrict sampling to the feasible set using an initial TT distribution, or (C) combine both.

A) Penalty constraints (recommended first implementation)

- Define viol(x) >= 0 with viol(x)=0 for feasible x.
- Use E(x) = QUBO(x) + rho * viol(x).
- Adapt rho during the run: increase rho if too many infeasible proposals; decrease if feasibility is very high.

B) Hard constraints via feasible-set TT / initialization

If your constraints have structure (e.g., exact cardinality, one-hot groups, small linear bounds), you can construct a tensor-train indicator I(x) in TT/MPS form (I(x)=1 if feasible else 0) and initialize the PROTES sampling distribution to be proportional to I(x).

- In practice, start with protes_general because it accepts a standard list of TT cores more naturally than the fast protes implementation.
- This approach can dramatically reduce wasted samples when the feasible region is tiny.

C) Hybrid constraints

- Hard-mask simple constraints (one-hot, per-group cardinality) by re-parameterizing variables.
- Penalty the remaining global constraints.
- This tends to keep TT ranks manageable and avoids sensitive penalty tuning.

4. Implementation details

4.1 Suggested software stack

- FM training: PyTorch implementation, fastFM/xlearn/libFM (choose based on speed and control).
- PROTES: anabatsh/PROTES (JAX-based).
- Baseline QUBO solvers: dwave-neal (simulated annealing), qbsolv (tabu/decomposition).
- Exact/near-exact baselines for small cases: MILP/MIQP solvers (SCIP, OR-Tools CP-SAT, Gurobi/CPLEX if available).

4.2 Converting an FM to QUBO

Given FM parameters w0, w (linear weights), and latent vectors v_i in R^k, define Q such that: x^T Q x reproduces the quadratic and linear terms. One common construction for x in {0,1}^d is:

```
Initialize Q = zeros(d,d)
For i: Q[i,i] += w_i                # linear term can go on diagonal for 0/1
variables
For i<j: Q[i,j] += dot(v_i, v_j)    # pairwise interaction
(Optionally symmetrize: Q = Q + Q^T - diag(diag(Q)))
Constant term w0 can be ignored for argmin.
```

Note: there are multiple equivalent QUBO encodings depending on whether you represent variables as {0,1} or {-1,+1}. Be consistent across FM training, energy evaluation, and baselines.

4.3 Writing the PROTES objective function

PROTES expects a batched objective f(I) where I is an array of shape (batch, d) containing the index per dimension. For binary optimization, use n=2 and interpret each I[:,i] as bit 0 or 1. Return a vector of energies.

```
def f_batch(I):
  # I: (B, d) with entries 0 or 1
  X = np.array(I, dtype=np.int8)
  e = qubo_energy(X, Q)
  v = violation(X)          # 0 if feasible, >0 otherwise
  return e + rho * v
```

4.4 Example violation functions

Below are common constraint patterns and simple penalty forms.

(i) One-hot per group

Groups G1,...,Gm with constraint sum_{i in Gg} x_i = 1.

```
viol = 0
for G in groups:
  s = X[:, G].sum(axis=1)
  viol += (s - 1)**2
```

(ii) Cardinality constraint

Constraint sum_i x_i = K.

```
s = X.sum(axis=1)
viol = (s - K)**2
```

(iii) Knapsack / linear inequality

Constraint a^T x <= b.

```
Ax = X @ a
viol = np.maximum(0, Ax - b)
```

4.5 Candidate selection and dataset updates

- Take the top-K candidates by predicted energy, but also add diversity (e.g., enforce a minimum Hamming distance).
- Always run a true feasibility check; do not trust the surrogate for constraints.
- Add infeasible candidates to Dc as negative samples (no oracle evaluation needed).
- To reduce stagnation, cap the training dataset size (e.g., keep most recent N points, or best+diverse).

4.6 Logging for reproducibility

- Log (best_cost vs oracle_calls) every iteration.
- Log feasibility rate of proposed candidates.
- Log surrogate metrics on a held-out set (R^2 and rank correlation) to diagnose surrogate quality.
- Log random seeds, hardware, PROTES parameters (k, k_top, TT rank limits if any), and FM hyperparameters (rank k, regularization).

5. Benchmarks

To evaluate the algorithm, benchmark on both (A) synthetic constrained QUBO instances and (B) standard combinatorial optimization datasets. The key is to measure performance as a function of oracle evaluations (sample efficiency), not just runtime.

5.1 Benchmark categories

A) Synthetic constrained QUBO

- Generate random or structured Q matrices (dense and sparse) with known scaling.
- Add constraints such as: cardinality (sum x_i = K), knapsack (a^T x <= b), multiple linear constraints (A x <= b), and one-hot groups.
- Optionally plant a known optimum by constructing Q around a planted solution.
- Use the true objective as the oracle. This allows exact evaluation of sample efficiency and regret.

B) Standard datasets mapped to binary

- 0/1 (multi-dimensional) knapsack instances (OR-Library).
- Set covering / set partitioning instances (OR-Library).
- Quadratic Assignment Problem (QAPLIB): structured quadratic objective with assignment constraints.
- Max-Cut instances (Gset) for unconstrained QUBO/Ising benchmarking (useful for sanity checks of the optimizer component).

5.2 Recommended public benchmark sources

Benchmark	Problem type	Why it is useful
OR-Library	Knapsack, set covering, set partitioning, etc.	Classic standardized instances with known best results
QAPLIB	Quadratic assignment	Quadratic objective + one-hot constraints; good test for hard feasibility
Gset (Max-Cut)	Max-Cut / Ising	Large sparse instances widely used in Ising/QUBO benchmarking
MaxCutBench	Max-Cut benchmark suite	Unified interface and curated hard instances for fair comparison

5.3 Baselines to compare against

- Random feasible sampling (with repair if needed).
- Greedy + local search (bit-flip hillclimb) with feasibility repair.
- Simulated annealing on penalized QUBO (e.g., D-Wave neal).
- Tabu/decomposition QUBO solver (e.g., qbsolv).
- Cross-Entropy Method / EDAs (probability vector over bits) with constraints via penalty or repair.
- Exact solvers for small sizes: MIQP/MILP or CP-SAT (serves as a quality upper bound).
- Ablations: (i) PROTES directly on the oracle (no FM) and (ii) FM + a different optimizer (SA) to isolate the benefit of PROTES.

5.4 Metrics

- Best feasible objective value vs number of oracle calls (primary metric).
- Feasibility rate of proposed candidates (how much budget is wasted).
- Time per iteration and total wall-clock time.
- Stability: mean and std across multiple random seeds.

- Surrogate quality: R^2 and Spearman rank correlation on held-out samples.

5.5 Experimental protocol

- Fix a budget (e.g., 1k, 5k, 20k oracle calls) and run each method with the same budget.
- Use at least 10 random seeds per instance to measure variability.
- Tune hyperparameters on a separate validation set of instances; report results on held-out test instances.
- Report both learning curves (best vs calls) and final best objective at each budget.

6. Practical tips and failure modes

- Penalty tuning: if feasibility is very low, increase rho aggressively or switch to hard-masked sampling.
- Mode collapse: if PROTES keeps sampling near one region, enforce diversity when adding new points to the dataset.
- Surrogate drift: periodically re-train with reweighting or a rolling window of recent data.
- Scaling: normalize the target costs before FM training; keep the QUBO coefficients numerically stable.

7. References (starting points)

- A. Batsheva et al., 'PROTES: Probabilistic Optimization with Tensor Sampling', NeurIPS 2023 / arXiv:2301.12162.
- anabatsh/PROTES (reference implementation): https://github.com/anabatsh/PROTES
- S. Rendle, 'Factorization Machines', ICDM 2010.
- R. Tamura, 'Black-box optimization using factorization and Ising machines' (FMQA overview), arXiv:2507.18003.
- J. E. Beasley, OR-Library: https://people.brunel.ac.uk/~mastjjb/jeb/info.html
- QAPLIB (Quadratic Assignment Problem Library): https://coral.ise.lehigh.edu/data-sets/qaplib/
- Gset Max-Cut instances (Stanford mirror): https://web.stanford.edu/~yyye/yyye/Gset/
- D-Wave neal (simulated annealing): https://github.com/dwavesystems/dwave-neal
- D-Wave qbsolv (tabu/decomposition): https://github.com/dwavesystems/qbsolv
- MaxCutBench / MaxCut-Bench (open-source benchmark suites): see OpenReview entries and repositories.
- J. Lopez Piqueres et al., constrained MPS training and perfect sampling (for hard constraint tensor networks), SciPost Phys. 2025.