Driving Ising Machines

- From QUBO Visualization and Modeling to Hyperparameter Tuning -

イジングマシンの活用方法に関する研究

― QUBO可視化と定式化から
ハイパーパラメータのチューニングまで ―

February, 2023

Matthieu PARIZY
パリジ　マチュー

Driving Ising Machines

- From QUBO Visualization and Modeling to Hyperparameter Tuning -

イジングマシンの活用方法に関する研究

― QUBO可視化と定式化から
ハイパーパラメータのチューニングまで ―

February, 2023

Waseda University Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering, Research on Information System Design

Matthieu PARIZY
パリジ　マチュー

I would like to dedicate my dissertation to my family. I have deep gratitude for my loving wife and son, Ayako and Léon Parizy, who supported me daily during this adventure. My parents, Michèle and Bruno Parizy, for pushing me to do my PhD and being a source of inspiration. My brother and sister, Vincent and Julie Parizy, for always being there for me.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background

With climate change and the COVID19 pandemic, sustainability has become a great challenge of this century. To tackle this social issue there are several fields to focus on such as logistics and drug discovery. One of the key common research fields between logistics and drug discovery is Operations Research (OR). [45, 12] OR is the application of analytical methods to help make better decisions. It has been an active field of research since the 1940s. Breakthroughs such as the Branch-and-Cut algorithm in the 1990s have made linear programming a widely adapted method to solve combinatorial optimization problems encountered in various applications [47]. Certain categories of combinatorial problems remain difficult to tackle and are still actively researched: NP-hard problems.

Ten years ago, the first Quantum Annealer by D-Wave [36], a dedicated quantum computer architecture to solve Quadratic Unconstrained Binary Optimization problems (QUBO) which are NP-hard, was released [6]. QUBO is represented as a polynomial on binary variables of the second order and quantum annealers find a combination of binary variable values to minimize (or maximize) the polynomial. Since the release of the Quantum Annealer, silicon based dedicated architectures have also emerged to solve QUBOs such as Hitachi's CMOS Annealer[62] or Fujitsu's Digital Annealer (DA)[19, 46]. Those quantum and silicon based architectures built to solve QUBOs have been called Ising machines. A common trait to the fore-mentioned Ising machines is they are all related, to a certain extent, to the well-known metaheuristic: Simulated Annealing (SA)[37].

The main characteristic of SA, compared to greedy metaheuristics, is it allows to make decisions when solving a problem which can worsen the current cost value $E_{current}$ of a solution considered, with a certain probability P depending on current temperature T as

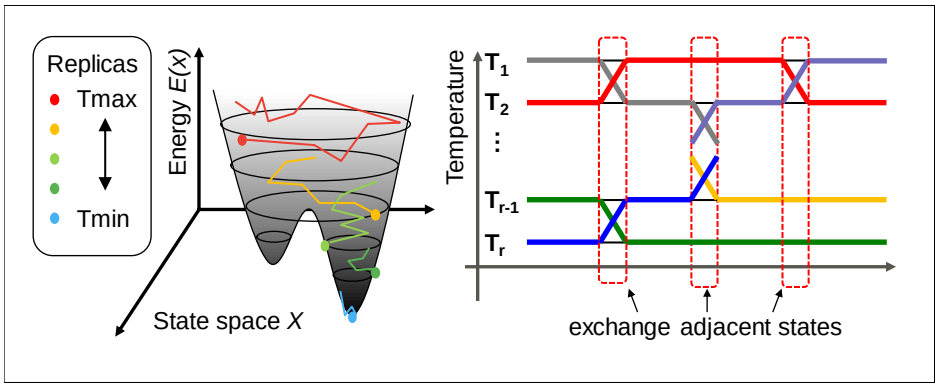

Fig. 1.1 Replica Exchange mechanism used in Digital Annealer

described by the metropolis criterion represented in Equation (1.1) and [44].

$$P = \min\left(1, \exp\left\{-\frac{(E_{next} - E_{current})}{T}\right\}\right) \tag{1.1}$$

where E_{next} is the cost of the considered new solution. When annealing, T starts with at T_{max} and then gradually lowers to T_{min} where both T_{max} and T_{min} are set by the user, $T_{max} \geq T_{min}$ and $T_{max}, T_{min} \in \mathbb{R}$. Thus at the beginning of the annealing process, decisions worsening E are more likely to happen than at the end of the process. Given enough solving time and proper SA parameter setting, SA guarantees to reach optimal solution but the required solving time could be astronomical thus many improvements to SA have been proposed since it was created.

An example of Ising machine, the Digital Annealer, operates using parallel tempering illustrated in Figure 1.1 and [42]. The main difference with SA is "replicas" explore the state (solution) space X in parallel, where each replica operate at a fixed T_r value where $T_{min} < T_r < T_{max}$. Once a replica r finds a state with a lower E value than adjacent replica $r-1$ lowest found E value, they both exchange their lowest E state and resume their search.

Most combinatorial optimization problems can be modeled as QUBO, with a varying amount of effort required [40], thus solving those problems on Ising machines has been actively researched for 10 years, especially comparing solving performance with traditional architectures and algorithms [32, 42]. A QUBO can be described by the following Equation (1.2):

$$E(x) = \sum_{i,j} Q_{ij} x_i x_j \tag{1.2}$$

with $x_i \in \mathbb{B}$ and coefficients $Q_{ij} \in \mathbb{R}$ for $1 \leq i \leq j$. Regardless of the Ising machine used, it has appeared clearly that depending on how a problem is modeled as QUBO, solving

performance can vary greatly. Likewise, solving performance also depends on Ising machine's hyperparameters, parameters which are used to control the QUBO solving process.

To maximize combinatorial problem-solving performance on Ising machines, this dissertation aims at tacking both the QUBO modeling and the hyperparameter setting parts of the QUBO solving process. This dissertation proposes a QUBO solving visualization technique to help identifying bottlenecks in the solving process to improve the modeling as well as an automated efficient Ising machine hyperparameter tuning framework.

Combinatorial problems used in this dissertation are the well known Quadratic Knapsack Problem (QKP) [11], Cardinality Constrained Portfolio Optimization Problem [15], Traveling Salesman Problem [51] and Quadratic Assignment Problem [13]. They all represent different kind of difficulties when using an Ising machine making them relevant for the topics of this dissertation. The Quadratic Knapsack Problem was first formulated as a QUBO in [25], at the time of writing Chapter 2, results of running its QUBO on an Ising machine had yet to be published. Since then, it has been studied in [58] which focuses on integer to binary variable encoding, and [55] which focuses on proposing a new kind of Ising machine. For Cardinality Constrained Portfolio Optimization Problem, although researches had been done on using Ising machines for financial applications such as index tracking with cardinality constraints [1], [21] and trading trajectory problem [53], it had never been solved on an Ising machine at the time Chapter 3 was written. Traveling Salesman Problem as well as Quadratic Assignment Problem had previously been studied in [43] and [42] respectively, and are used for benchmarking purpose.

1.2 Dissertation Overview

The rest of this dissertation is organized as follows:

Chapter 2 [Analysis and Acceleration of the Quadratic Knapsack Problem on an Ising Machine] proposes a QUBO search space landscape visualization technique which uses two local minima solutions found to represent the multi-dimensional space in between and understand what makes a QUBO hard to solve. This technique is applied to the Quadratic Knapsack Problem (QKP) where the goal is to maximize the value of items inserted in a knapsack. This is known to be an NP-hard problem. Chapter 2 shows that moving in and out heavy items from the knapsack is difficult due to the nature of representing linear inequality constraints as QUBO. With this insight, a solution mending method is proposed to help Ising machines stuck in local minima which raises the chances of finding the optimal solution with an Ising machine from only 6.7% to 60.7% for an Ising machine used with the proposed solution mending method. Chapter 2 also compares the proposed Ising machine

with solution mending against SA. Results show that for all problem instances evaluated on, optimal solution is hard to reach for SA.

Chapter 3 [Cardinality Constrained Portfolio Optimization on an Ising Machine] proposes a novel integer variable to binary variables modeling technique. Proposed landscape technique from Chapter 2 helped us to identify another bottleneck when solving problems which include integer variables. When using classic encoding methods, those encodings either create large differences in the cost function when changing a binary variable value representing a large integer value, which makes those moves unlikely to happen, or require a large quantity of binary variables, creating both computing complexity and a larger memory footprint. A "base10" encoding is proposed as a tradeoff between all fore-mentioned encodings. Proposed encoding is applied to the Cardinality Constrained Mean Variance Portfolio Optimization Problem (CCMVPOP), which is an NP-hard problem using real number variables. An efficient QUBO model for the CCMVPOP is thus proposed by first converting real number variables to integer variables using coefficient multiplication with rounding combined with integer to binary variable encoding. Solving performance using the proposed encoding is compared to classic encodings. Results show the time to the best-known solutions can be improved by a factor of up to 10x for several CCMVPOP instances. Results also show the proposed encoding is the only one which allows to reach the best-known solutions for the largest CCMVPOP instance available in the used benchmark data.

Chapter 4 [Fast Hyperparameter Tuning for Ising Machines] proposes a novel Ising machine hyperparameter tuning framework. It is based on machine learning state of the art hyperparameter tuning method called Tree-structured Parzen Estimator (TPE), which is a kind of Bayesian optimization technique. After extending TPE to Ising machines, as well comparing TPE to random parameter sampling, an enhanced TPE called "FastConvergence" is proposed. FastConvergence reduces the time required to find parameters which enable the same level of performance as TPE. Random sampling, TPE, and "FastConvergence" are compared using DA to solve Travel Salesman Problem (TSP) and Quadratic Assignment Problem (QAP), two well-known NP-hard problems often used for Ising machine benchmarking. Results show that the proposed FastConvergence can find parameters which give solving performance equivalent or better than TPE with two to three times faster tuning time.

Chapter 5 [Conclusions] summarizes the dissertation. In conclusion, performance when solving combinatorial problems using Ising machines vary greatly depending on the QUBO modeling technique as well as the used Ising machine's hyperparameters. It is thus vital to keep developing bottleneck analysis techniques to inspire future Ising machine architectures and solving algorithm techniques. It is also critical to have robust hyperparameter tuning framework to both maximize Ising machines' performance but also, in general, have fair

comparisons between metaheuristics-based solvers which have highly sensitive parameters, as one could easily disregard some solver in favor of another due to poor or lack of parameter tuning. Continuing to improve the proposed hyperparameter tuning framework and QUBO solving visualization are our future works.

Chapter 2

Analysis and Acceleration of the Quadratic Knapsack Problem on an Ising Machine[1]

2.1 Introduction

2.1.1 The Binary Quadratic Knapsack Problem

The binary quadratic knapsack problem (QKP) is the problem of finding the combination items giving the highest profit within the capacity C of a knapsack for a given triangular profit matrix $P = \{p_{ij}\}$ of positive values. Let x_i be a binary variable showing whether the item i is chosen or not. If $x_i = 1$, the item i is chosen. p_{ij} is the profit when both item i and item j are chosen. p_{ii} is the profit when only item i is chosen. Each item i has a weight w_i:

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_i x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq C \quad x_i \in \{0, 1\} \tag{2.2}$$

It is a generalization of the binary knapsack problem (where $p_{ij} = 0$ for $i \neq j$) as well as several other combinatorial problems which can also be expressed as a QKP such as the clique problem, and is considered a difficult NP-hard problem [14] [50].

[1]Technical contents in this chapter have been presented in the publication ⟨1⟩ and in the conference ⟨7⟩.

Its known applications are VLSI design [22], compiler design [35], budgeting problem [39] and network flows [52].

QKP has been investigated for more than 40 years since its introduction by Gallo et al. [23]. Various approaches ranging from exact algorithms, heuristics, meta-heuristics, linearization have been tried. The most efficient current approach to our knowledge comes from C. Patvardhan et al. using a quantum inspired evolutionary algorithm (QIEA) [49] and will thus be the base for comparing our results. Although the QKP was first formulated as an Ising model in [25] by Glover et al., to our knowledge, results of running this Ising model on an Ising machine have yet to be published.

2.1.2 Ising Machines

Ising machines have been gaining momentum over the course of the last 10 years with the advent of commercialization of quantum annealers [36] and other specialized architectures targeting the same problem. They take for input an Ising spin model and search for the lowest energy configuration of spins (which can have an "up" or "down" value, translatable to 1 or 0) i.e. the ground state. Numerous kinds of combinatorial problems can be mapped to an Ising model, such as QKP. Solving these mapped problems on Ising machines allows for certain categories of problem to be solved more efficiently than conventional methods. They represent an attempt to circumvent Moore's Law [5] [30].

QKP's quadratic objective function seemed like a good fit as it can be translated "as is" in an Ising model, although the capacity constraint had to be converted to an objective function using binary auxiliary variables. The broad applications of QKP, the fit to Ising model and the novelty of Ising machines are what motivated our research.

2.1.3 Our Proposal

In this chapter we propose to analyze the results of running Ising models of QKP on an Ising machine on benchmark instances. Firstly, we run them using a naive formulation without any particular enhancements, secondly we analyze the bottlenecks of this first run, finally we propose software solution improvement algorithms to improve the results.

2.1.4 Contributions

Our main contributions are the following to improve convergence to optimal solution:

- Visualization techniques of Ising models which inspired the following contribution.

- Low α coefficient for the constraint part of the Hamiltonian coupled with a software solution improvement technique to "mend" non-feasible solutions and improve feasible ones into high quality feasible solutions.

Our benchmark used in this chapter relies on instances from the work of Billionnet et al. [11]. To our knowledge, since the publication of [50], quantum inspired evolution algorithm approach from Patvardhan et Al. [49] is the approach yielding the best results both time wise and chance of solving an instance, it will thus be used for comparison with our results.

We do not claim superiority of Ising machines for QKP, but we want to share the various shortcomings we had and the techniques we came up with to overcome those and benchmark those techniques efficiency within the paradigm of finding the best solution using Ising models.

2.2 Simple QKP Trial on an Ising machine

In this section we present the results of the Ising models formulation found in [25] on instances taken from [11] using an Ising machine called a "Digital Annealer" (DA) [3],[42] as well as a "simulated annealing sampler"(SA) [18] for a baseline comparison.

2.2.1 Benchmark Instances Description

We used QKP instances from [11] which are all combinations of variable number $n = [100, 200, 300]$, profit matrix density $\Delta = [0.25, 0.5, 0.75, 1]$ randomly generated 10 times. For $n = 300$ only $\Delta = [0.25, 0.5]$ are present. Weight values have a normal distribution between 1 and 50. Profit values have a normal distribution between 1 and 100. Capacity is randomly distributed in $[50, \sum_i w_i]$.

2.2.2 QKP Ising Formulation

The first formulation of QKP as an Ising model or Hamiltonian was done by Glover et al. [25]. It is the following:

$$H = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(-p_{ij} x_i x_j\right) + \alpha \left(\sum_{i=1}^{n} w_i x_i - C + y \right)^2 \tag{2.3}$$

where p_{ij} is the profit when both item i and item j are chosen, x_i is the binary variable representing if item i is chosen or not, w_i is the weight of item i, C is the weight capacity of

the knapsack and y is the auxiliary integer variable for when the capacity is not fully used with $0 \leq y < C$. H is called Hamiltonian energy or energy for short.

The integer auxiliary variable y following a one-hot encoding is expressed by binary variables y_i as follows:

$$y = \sum_{i=0}^{m} i y_i \text{ with } y \in \mathbb{Z} \text{ and } y_i \in \{0, 1\} \tag{2.4}$$

$$\text{subject to: } \left(\sum_{i=0}^{m} y_i - 1 \right)^2 = 0 \tag{2.5}$$

Auxiliary variables only need to compensate for values up to the maximum value minus one of an item in an instance of [11] as any values above would mean another item could be added in the knapsack as the max possible capacity of the knapsack in any instance is $\sum_i w_i$. Therefore $m = \max w_i - 1$. The reason for limiting the auxiliary variable range is to limit the number of iterations consumed flipping those variables.

We set the constraint coefficient α in Eqn. (2.3) as $strength \times n \times \Delta$ with *strength* initial value set to 0.1, n being the problem size and Δ the profit matrix density. The larger and denser a problem is, the more adding an item to the knapsack can generate higher profit. Thus, α must be set to a higher value accordingly as Δ and n grow to prevent penalty violation.

If all the replicas (detailed will be explained in Section 2.2.3) of the DA converged to an infeasible solution, we judged that $strength$ was too low and increased it by 50%. We repeat this process until at least one feasible solution was reached.

2.2.3 Digital Annealer and its Hyper-Parameters

The DA operating principle is searching for the ground state of an Ising model using the basis of the Markov chain Monte Carlo (MCMC) method [42]. In the model used for the presented experiments, up to 8,192 variables can be defined with up to 67,108,864 couplers to define the weight binding the variables with a precision up to 64 bits (i.e. any variable x_i can be unconditionally connected to any variable x_j for a "full connectivity"). We choose the DA as its full connectivity makes it attractive for QKP with a high profit matrix density as they can be run without any transformation.

Another specificity of the DA is to have multiple processes, or replicas, doing the same MCMC search, at different temperatures, to speedup finding the ground state [42]. All the r replicas are given the same Ising model with their own different temperatures, evenly spread on a logarithmic scale between T_{max} and T_{min}, set by the user. Temperature wise

adjacent replicas exchange their state, which is their lowest energy found solution, every "exchange rate" iterations. This parallel search is also known as parallel tempering [34]. Its main merit is to be less sensible to temperature scheduling in comparison to simulated annealing, as search will not halt on a local minimum if temperature is too low. Another DA specific mechanism is the "dynamic offset" [3] : every time a replica does not flip any binary variable in an iteration, DA applies an energy offset allowance, set by a corresponding hyper parameter called "`offset_increase_rate`", letting DA accept a binary flip causing an energy increase less or equal than the accumulated offset allowance, regardless of temperature. When a flip occurs, the offset allowance is reset to 0.

For our experiments we used parallel tempering with $r = 26$ replicas with $T_{max} = 9000$ and $T_{min} = 0$, without any temperature decay, with a replica temperature exchange rate of 100 iterations and an `offset_increase_rate` value of 100.[2] Thus every time a replica does not flip any binary variable in an iteration, we apply an energy offset allowance of +100 and then reset when a flip occurs. All problems are ran over 750 million iteration which takes about 39 seconds with the above settings. Every result is evaluated over 20 different random seeds. Finally, the initial value of every binary variable is set to 0.

2.2.4 Simulated Annealing

To provide a baseline to compare with DA results, we chose the SA from D-Wave [18], which can perform simulated annealing on an Ising model. There are two reasons for this choice. First, as far as we know, no SA implementation specialized for the QKP have been proposed. The closest implementations we found were respectively for the quadratic multiple knapsack problem [17] and the generalized quadratic multiple knapsack [16] but they are not the same as our QKP. Secondly, the advantage of Ising model solvers is their ability to solve any combinatorial problem mapped to an Ising model, whereas specialized SA implementations are limited to the problem they are applied to. Thus, for a fair comparison, we decided to choose a software implementation of SA for Ising models.

We ran the SA with the same α, T_{max} and T_{min} as DA. We adjusted the SA number of iterations so that it would take 39 seconds, the same time as our DA experiments. This gave an SA number of iterations of 2,750,000. As for DA, every result is evaluated over 20 different random seeds and every binary variable initial value is set to 0. We also ran another experiment with double the number of iterations for a total of 5,500,000 for reference.

Because some SA runs would lead to a constraint-violating solution, we applied the lightweight Algorithm 1 to mend these solutions to feasible solutions, whereas for DA, we

[2] Those parameters are the one which worked best for us over all instances after tuning them using *hyperopt* [10].

Algorithm 1 SA Solution Mending Algorithm

procedure SOLUTIONMENDING
input:
    Initial SA solution: $s = \{x_1, x_2, ..., x_n\}$
output:
    Mended solution: $s' = \{x_1', x_2', ..., x_n'\}$
main:
    $s' = s$;
    for every x_k in $s' \mid x_k = 1$ **do**
        Evaluate x_k removal cost loss;
    Accept the item removal giving the minimum cost loss;
    Repeat the above process until s' is feasible;
    return s'

keep as solution the best non constraint violating solution within all the replicas and our α policy ensures at least one replica reached a feasible solution.

2.2.5 First Benchmark Results

Tables 2.1, 2.2 and 2.3 show the number of times optimal answer was reached over the 20 random seeds for each QKP instance of size 100, 200 and 300 respectively as "DA success%", "SA1 success%" and "SA2 success%", respectively for a 39s DA run, 39s SA run and 78s SA run, when trying to find the ground state of the Ising model. Likewise, "DA gap%", "SA1 gap%" and "SA2 gap%" are the mean gap achieved for the 20 seeds used. Gap is measured as described in Eqn. 2.6:

$$Gap = \frac{opt\_val - obtainedValue}{opt\_val} \times 100 \tag{2.6}$$

where *opt_val* is the known optimal profit value in every QKP instance from [49] and *obtainedValue* is the value reached by DA or SA. "Instance name" is the name of the QKP instance used. Δ is the profit matrix density expressed in %. The experiments on SA1 and SA2 were conducted on an Intel Core i7-9700K, 8 cores CPU with a base frequency of 3.6 GHz, using python 3.7.9 and dwave-neal 0.5.7.

We observed that in all problem instances SA could not reach optimal solution once, even for 78s runs. We also observed that in all instances, except r_100_75_1, r_200_100_1, r_200_100_8 and r_300_50_3, DA gap is much closer to optimal than SA1 and SA2. The average gap% over all instances for SA1 is 18.58%, for SA2 18.00% against 8.15% for DA. The average success% over all instances for SA1 and SA2 is none against 6.7% for DA.

Table 2.1 100 variables problem results.

Instance name	Δ	opt_val	DA gap%	SA1 gap%	SA2 gap%	DA success%	SA1 success%	SA2 success%
r_100_25_1	25.0	18,558.0	0.6	12.1	12.3	5.0	0	0
r_100_25_10	25.0	24,930.0	0.0	8.7	8.4	30.0	0	0
r_100_25_2	25.0	56,525.0	2.9	12.3	12.5	0.0	0	0
r_100_25_3	25.0	3,752.0	0.0	32.0	31.0	100.0	0	0
r_100_25_4	25.0	50,382.0	4.5	14.2	14.6	0.0	0	0
r_100_25_5	25.0	61,494.0	1.7	6.7	6.2	0.0	0	0
r_100_25_6	25.0	36,360.0	0.7	9.5	8.6	0.0	0	0
r_100_25_7	25.0	14,657.0	0.4	10.4	11.3	0.0	0	0
r_100_25_8	25.0	20,452.0	0.4	12.9	11.3	20.0	0	0
r_100_25_9	25.0	35,438.0	2.6	12.0	11.3	0.0	0	0
r_100_50_1	50.0	83,742.0	1.8	10.2	9.9	0.0	0	0
r_100_50_10	50.0	88,634.0	1.4	9.3	8.6	0.0	0	0
r_100_50_2	50.0	104,856.0	2.7	14.2	13.0	0.0	0	0
r_100_50_3	50.0	34,006.0	0.0	12.6	13.7	55.0	0	0
r_100_50_4	50.0	105,996.0	0.8	6.0	5.5	0.0	0	0
r_100_50_5	50.0	56,464.0	8.3	29.9	30.0	0.0	0	0
r_100_50_6	50.0	16,083.0	0.0	13.5	13.1	100.0	0	0
r_100_50_7	50.0	52,819.0	3.4	15.1	14.3	0.0	0	0
r_100_50_8	50.0	54,246.0	2.9	14.8	14.1	0.0	0	0
r_100_50_9	50.0	68,974.0	3.2	13.5	12.1	0.0	0	0
r_100_75_1	75.0	189,137.0	4.2	3.7	3.5	0.0	0	0
r_100_75_10	75.0	143,740.0	1.1	7.4	6.7	0.0	0	0
r_100_75_2	75.0	95,074.0	2.5	12.3	12.0	0.0	0	0
r_100_75_3	75.0	62,098.0	2.2	10.6	12.1	0.0	0	0
r_100_75_4	75.0	72,245.0	1.0	9.3	8.8	0.0	0	0
r_100_75_5	75.0	27,616.0	0.2	14.9	17.6	10.0	0	0
r_100_75_6	75.0	145,273.0	4.7	16.8	17.3	0.0	0	0
r_100_75_7	75.0	110,979.0	6.2	21.4	19.7	0.0	0	0
r_100_75_8	75.0	19,570.0	8.0	43.1	46.4	0.0	0	0
r_100_75_9	75.0	104,341.0	7.7	24.8	25.8	0.0	0	0
r_100_100_1	100.0	81,978.0	8.4	28.5	27.8	0.0	0	0
r_100_100_10	100.0	193,262.0	1.0	7.0	6.6	0.0	0	0
r_100_100_2	100.0	190,424.0	1.0	5.8	6.0	0.0	0	0
r_100_100_3	100.0	225,434.0	0.2	4.0	3.7	0.0	0	0
r_100_100_5	100.0	230,076.0	0.0	3.0	2.9	30.0	0	0
r_100_100_6	100.0	74,358.0	11.3	36.5	33.2	0.0	0	0
r_100_100_7	100.0	10,330.0	0.0	28.6	26.0	100.0	0	0
r_100_100_8	100.0	62,582.0	10.5	35.5	34.7	0.0	0	0
r_100_100_9	100.0	232,754.0	5.0	6.6	6.7	0.0	0	0

Whereas in [49] success% of 100% for all instances within less than 0.02 seconds is achieved with QIEA.

In Table 2.2 and Table 2.3, having larger problem instances only make things worse: optimal value solution found for only a single instance with DA.

Table 2.2 200 variables problem results.

Instance name	Δ	opt_val	DA gap%	SA1 gap%	SA2 gap%	DA success%	SA1 success%	SA2 success%
r_200_25_1	25.0	204,441.0	2.4	6.7	6.6	0.0	0	0
r_200_25_10	25.0	48,459.0	9.5	28.8	27.2	0.0	0	0
r_200_25_2	25.0	239,573.0	0.4	2.9	2.6	0.0	0	0
r_200_25_4	25.0	222,361.0	4.9	11.0	10.8	0.0	0	0
r_200_25_5	25.0	187,324.0	3.3	7.8	7.8	0.0	0	0
r_200_25_6	25.0	80,351.0	7.3	17.7	15.8	0.0	0	0
r_200_25_7	25.0	59,036.0	10.7	25.4	25.3	0.0	0	0
r_200_25_8	25.0	149,433.0	4.4	11.1	10.2	0.0	0	0
r_200_25_9	25.0	49,366.0	8.2	23.8	24.9	0.0	0	0
r_200_50_1	50.0	372,097.0	4.5	8.9	8.5	0.0	0	0
r_200_50_10	50.0	284,751.0	8.6	14.7	15.6	0.0	0	0
r_200_50_2	50.0	211,130.0	7.4	15.7	14.6	0.0	0	0
r_200_50_3	50.0	227,185.0	18.1	31.3	28.4	0.0	0	0
r_200_50_4	50.0	228,572.0	18.9	31.3	31.8	0.0	0	0
r_200_50_5	50.0	479,651.0	0.3	2.0	2.2	0.0	0	0
r_200_50_6	50.0	426,777.0	1.6	4.6	4.5	0.0	0	0
r_200_50_7	50.0	220,890.0	11.4	21.3	19.1	0.0	0	0
r_200_50_8	50.0	317,952.0	5.9	10.5	9.7	0.0	0	0
r_200_50_9	50.0	104,936.0	11.2	25.5	23.0	0.0	0	0
r_200_75_1	75.0	442,894.0	16.5	22.0	22.5	0.0	0	0
r_200_75_10	75.0	142,694.0	8.6	21.3	19.1	0.0	0	0
r_200_75_2	75.0	286,643.0	15.1	24.1	21.9	0.0	0	0
r_200_75_3	75.0	61,924.0	4.0	26.0	23.2	0.0	0	0
r_200_75_4	75.0	128,351.0	18.8	36.6	35.8	0.0	0	0
r_200_75_5	75.0	137,885.0	10.2	26.8	24.8	0.0	0	0
r_200_75_6	75.0	229,631.0	21.2	33.5	32.3	0.0	0	0
r_200_75_7	75.0	269,887.0	9.9	20.9	19.0	0.0	0	0
r_200_75_8	75.0	600,858.0	11.1	13.8	13.3	0.0	0	0
r_200_75_9	75.0	516,771.0	14.2	20.4	18.8	0.0	0	0
r_200_100_1	100.0	937,149.0	7.2	6.1	5.3	0.0	0	0
r_200_100_10	100.0	378,442.0	10.0	15.8	14.5	0.0	0	0
r_200_100_2	100.0	303,058.0	14.7	23.1	22.3	0.0	0	0
r_200_100_3	100.0	29,367.0	0.0	21.3	18.9	100.0	0	0
r_200_100_4	100.0	100,838.0	8.3	27.1	26.8	0.0	0	0
r_200_100_5	100.0	786,635.0	7.7	9.8	9.7	0.0	0	0
r_200_100_6	100.0	41,171.0	11.3	31.9	29.9	0.0	0	0
r_200_100_7	100.0	701,094.0	6.3	9.5	9.0	0.0	0	0
r_200_100_8	100.0	782,443.0	14.2	14.1	14.1	0.0	0	0
r_200_100_9	100.0	628,992.0	6.3	9.3	9.0	0.0	0	0

Table 2.3 300 variables problem results.

Instance name	Δ	opt_val	DA gap%	SA1 gap%	SA2 gap%	DA success%	SA1 success%	SA2 success%
r_300_25_1	25.0	29,140.0	15.2	43.9	43.7	0.0	0	0
r_300_25_10	25.0	383,377.0	9.4	14.5	14.0	0.0	0	0
r_300_25_2	25.0	281,990.0	20.8	30.9	29.3	0.0	0	0
r_300_25_4	25.0	444,759.0	13.2	19.8	18.8	0.0	0	0
r_300_25_5	25.0	14,988.0	5.6	36.8	37.4	0.0	0	0
r_300_25_6	25.0	269,782.0	22.9	34.0	33.3	0.0	0	0
r_300_25_7	25.0	485,263.0	4.2	7.8	7.4	0.0	0	0
r_300_25_8	25.0	9,343.0	0.0	30.4	27.6	100.0	0	0
r_300_25_9	25.0	250,761.0	12.8	20.1	20.3	0.0	0	0
r_300_50_1	50.0	513,379.0	35.0	36.4	36.6	0.0	0	0
r_300_50_10	50.0	996,070.0	4.2	5.9	5.9	0.0	0	0
r_300_50_2	50.0	105,543.0	41.7	52.3	50.1	0.0	0	0
r_300_50_3	50.0	875,788.0	17.5	16.8	16.9	0.0	0	0
r_300_50_4	50.0	307,124.0	40.7	41.2	42.9	0.0	0	0
r_300_50_5	50.0	727,820.0	22.2	24.2	23.7	0.0	0	0
r_300_50_6	50.0	734,053.0	22.7	23.4	23.4	0.0	0	0
r_300_50_7	50.0	43,595.0	2.3	24.3	21.2	0.0	0	0
r_300_50_8	50.0	767,977.0	22.7	23.1	22.9	0.0	0	0
r_300_50_9	50.0	761,351.0	9.3	12.6	12.2	0.0	0	0

2.3 Identifying Bottlenecks Using Visualization

Starting from the DA benchmark results from Section 2.2, we picked instance "r_100_25_10", with penalty Hamiltonian $\alpha = 5.5$, where optimal solution was difficult to reach for some random seeds and came up with the following algorithm for analyzing what makes reaching optimal solution difficult.

Algorithm 2 Visualization

procedure EnergyPerspective
input:
  $start \leftarrow (x_1, \ldots, x_n)$
  $end \leftarrow (y_1, \ldots, y_n)$
output:
  $(x_a, x_b, \ldots, x_k) \mid start_{a,b,\ldots,k} \oplus end_{a,b,\ldots,k} = 1$
  $energy_a, energy_{a,b} \ldots energy_{a,b\ldots k}$
main:
  $PerspectiveOrder = \Phi$;
  $Energies = \Phi$;
  while $start \neq end$ **do**
    $diff \leftarrow \{x_k \mid start_k \oplus end_k = 1\}$
    for all $x_k \in diff$ **do**
      Try all x_k flips and record the index k in *PerspectiveOrder* and $energy_k$ in *Energies* of the one giving the largest energy decrease (or smallest energy increase).
      Do the actual flip of the best x_k
  return PerspectiveOrder, Energies

The key goal is to approximate the Hamiltonian energy landscape from an easy to reach sub-optimal solution to an optimal solution to be able to pinpoint what makes a problem hard to solve and eventually give us idea on how to tackle this difficulty. The procedure to approximate the landscape is as follows in Algorithm 2.

In Algorithm 2, a set $\{x_1, \ldots, x_n\}$ of binary variables shows the sub-optimal solution for QKP and a set $\{y_1, \ldots, y_n\}$ of binary variables shows the final optimal solution. The procedure can be described as:

1. Extract the non-common variables between the sub-optimal solution "start" and optimal solution "end".

2. Calculate and record which flip of non-common variable in start would cause the lowest energy increase.

3. Do the actual flip and repeat from 2. until start = end.

Thus $(x_a, x_b, \ldots, x_k)$ is the order in which to flip bits such that the Hamiltonian energy, given by equation (2.3), increase is the minimum possible and $energy_a, energy_{a,b} \ldots energy_{a,b\ldots k}$ are the respective energies of flipping x_a, x_a then x_b,x_a then x_b ... then x_k from start.

We then plot in order $energy_a, energy_{a,b} \ldots energy_{a,b\ldots k}$ on the y axis, and on the x axis the number of flips done from start. It gives us the perspective in Figure 2.1.

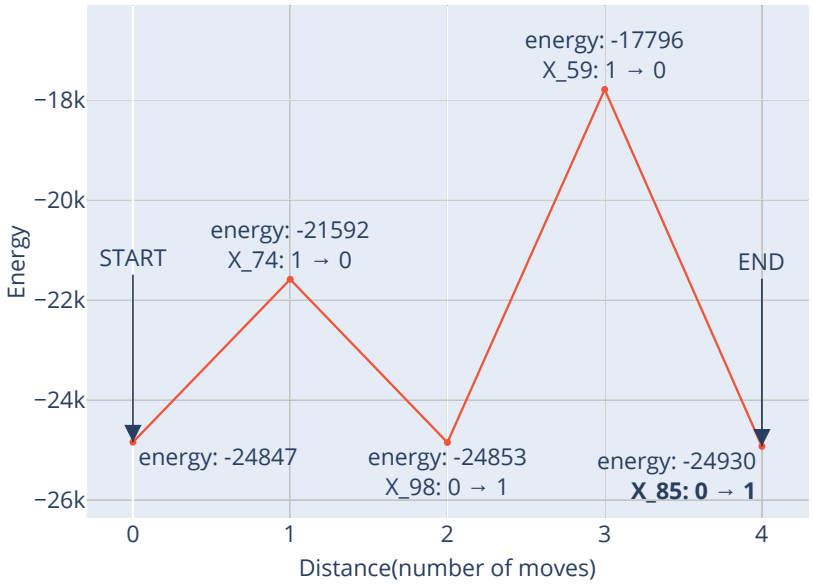

Fig. 2.1 Energy landscape example: r_100_25_10 instance with $\alpha = 5.5$.

We annotated the variables which were switched at each move in order to go from our best non-optimal solution(START) to one optimal solution (END). We also annotated their energy (y axis value) for clarity.

The key takeaways from this perspective are:

1. START and END are only two variable swap away from each other: $x_{74} \leftrightarrow x_{98}$ and $x_{59} \leftrightarrow x_{85}$.

2. $x_{59} \leftrightarrow x_{85}$ causes the largest spike on the landscape. ($\Delta E \approx 7 \times 10^3$)

Tackling both takeaways will be the focus of respectively Sections 2.4.1 and 2.4.3.

2.4 Algorithms for Solution Improvement

2.4.1 Solution Improvement Algorithm v1

This algorithm is a simplified version of "Procedure ImproveLocal(P)" described in [49] adapted for an Ising machine running parallel trials on different replicas such as we have here. For all replicas' solution obtained, we try all possible item swaps from inside and outside the knapsack, do the swap which generates the largest profit increase (while not violating the weight constraint), and iterate until no improvement can be done, resulting in a computing intensive process to improve current solutions.

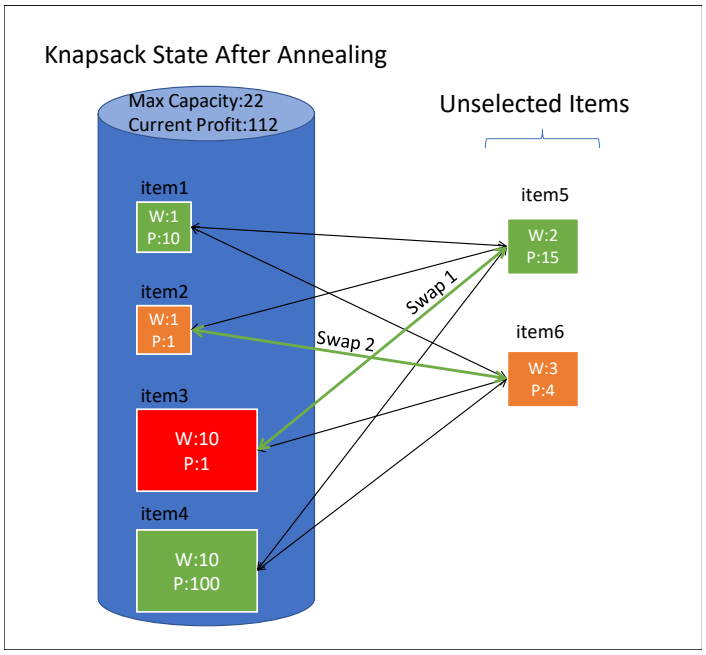

Fig. 2.2 SIA v1 example.

We illustrate an example of our solution improvement algorithm v1 (SIA v1) in Figure 2.2 using a non-quadratic knapsack problem to simplify explanations. In this example, "W" represents the weight of an item and "P" is the profit of this item and we have already a solution obtained. We consider all possible swaps to improve the profit while staying under the capacity limitation which leads us to "Swap1", replacing item3 by item5 and then "Swap2" which replaces item2 with item6. The item coloring represents profit over weight metric, green means a substantial amount of profit is generated for a weight unit whereas red is the opposite.

Unlike "ImproveLocal(P)" from [49], we do not consider the action of adding without removing an item to improve a solution. Our reasoning is that in a real world scenario, it is unlikely that a replacement of an heavy item inside the knapsack by a light item outside the knapsack would bring a profit increase, as heavy items tend to bring more value for one unit of weight than lighter ones, and saves 1 loop of trials per solution to be improved. We note that considering item adding might help on randomly generated instances such as here. The pseudo-code is shown on Algorithm 3.

2.4.2 Solution Improvement Algorithm v1 Results

We made the following experiment to evaluate our proposed solution improvement algorithm (SIA v1). We used exactly the same DA parameters as in Section 2.2. We allowed SIA v1

Algorithm 3 Solution Improvement Algorithm

procedure SOLUTIONIMPROVEMENT
input:
    Initial solutions: $S = \{s^1, s^2, ..., s^r\}$
output:
    Improved solutions: $S' = \{s^{1\prime}, s^{2\prime}, ..., s^{r\prime}\}$
main:
    $S' = S$;
    for every s^i in S' **do**
        for every pair (x^i_k, x^i_l) in $s^i \mid \{x^i_k, x^i_l\} = (1,0)$ **do**
            Evaluate (x^i_k, x^i_l) swap cost;
        Accept the possible swap giving the maximum cost increase and update the solution s^i;
    Repeat the above process until no further improvement possible within timeout value;
    return S'

Table 2.4 success% comparison.

n	DA	DA + SIA v1
100	11.54	68.33
200	2.56	61.67
300	5.26	13.95
mean	6.70	55.00

to do as many passes as it does for 40 seconds, which is the same timeout value as for the DA. The experiments were conducted on an Intel Core i7-9700K, 8 cores CPU with a base frequency of 3.6 GHz, using python 3.7.9 and NumPy 1.17.3 [31]. We recorded success% as described in Section 2.2. For space issues purposes we aggregated the average success% of each instance sizes in Table 2.4. The "mean" row is the average success% calculated over all instances from Tables 2.1, 2.2 and 2.3 regardless of their size for both DA and DA+SIA v1. Note that we will later include the full breakdown for the solution improvement algorithm v2.

We observe a very sharp increase in % of instances which can be solved using the SIA v1, especially for smaller instances (100,200 items). We note that for the largest instances (300) the increase is much smaller. We consider that it is due to the timeout of 40 seconds being enough for smaller instances to converge to optimal solution in the majority of instances whereas it is not enough for most 300 items instances.

Table 2.5 "r_100_25_10" instance with $\alpha = 5.5$ metrics comparison.

	$\sum_j p_{ij}$	w_i	RPD	$\sum_{\substack{j \\ j \neq i}} Q_{i,j}$	$Q_{i,i}$
mean	1,222.4	26.1	99.0	198,331.5	-53,403.9
std	255.7	13.6	175.9	102,745.1	27,551.5
min	728.0	1.0	18.2	7,666.0	-101,500.0
Q1	1,051.2	15.0	31.9	114,570.0	-77,596.0
med	1,212.0	25.5	45.8	194,233.0	-52,389.5
Q3	1,368.2	38.0	87.9	288,496.0	-30,975.0
max	2,206.0	50.0	1,201.0	378,400.0	-2,079.0
x_{59}	1,668.0	36.0	46.3	273,456.0	-73,584.0
x_{85}	1,162.0	38.0	30.6	288,496.0	-77,596.0

mean, std, min, Q1, med, Q3 and max are respectively the average, the standard deviation, the minimum, the first quartile, the median, the third quartile and the maximum.

2.4.3 Solution Improvement Algorithm v2

As we saw in the perspective from Figure 2.1, changing the values of some variables while trying to leave a local optimum in search of a global one can create energy spikes difficult to overcome. We first try to understand what causes them by analyzing the problem's data and its formalization for the same r_100_25_10 instance with $\alpha = 5.5$. First, we define a few metrics for statistical comparison:

- Potential sum of profit: $\sum_j p_{ij}$ if all items j connected to item i were also chosen.
- Item i 's weight: w_i
- Relative profit density (RPD): $\sum_j p_{ij}/w_i$ also described in [49].
- Constraint Hamiltonian's $\sum_{\substack{j \\ j \neq i}} Q_{i,j}$ where $Q_{i,j}$ is the penalty value (the second term in Eqn. (2.3)) added to the energy for item i if all items j connected to i were also chosen.
- Constraint Hamiltonian's $Q_{i,i}$: the value subtracted to the energy when item i is chosen.

By comparing the distributions of the metrics described above with the items 59 and 85 (source of the energy spike in Figure 2.1), we observe that although x_{85} is chosen in most DA solutions, it has a weaker RPD than x_{59} and also weights heavier. We notice that the constraint Hamiltonian values are one order of magnitude larger than profit values. This difference of magnitude comes from the fact that we need a high enough α, to prevent having

Algorithm 4 Filtering for removal of items

```
1: procedure FILTERING
2: input:
3:     Profit: p_ij (1 ≤ i ≤ n, 1 ≤ j ≤ n)
4:     Weight: w_i (1 ≤ i ≤ n)
5:     filterLimit
6: output:
7:     Indexes: I = {i_1, i_2, ..., i_FilterLimit}
8: main:
9:     for every item i do
10:        Calculate the relative profit density rpd(i) = Σ_j p_ij / w_i;
11:    Sort rpd(i) and pick up 1st to filterLimit-th items into I;
12:    return I
```

the global ground state of the Ising model having constraint penalties, and we need a limited auxiliary variable range to limit the number of iterations spent on flipping auxiliary variable. This difference of magnitude produces an energy landscape which drives an Ising machine to fill the knapsack in priority and generating the highest profit second. The Hamiltonian thus gives priority to heavier items regardless of how much profit they bring, and generates a high penalty when trying to remove such item, which in turn makes escape from a local minimum difficult for an Ising machine.

It gave us the idea to use the RPD metric help escape from local minimum while preserving computing resources. The key idea is to give priority to the items with the lowest RPD for removal from the knapsack . How many of those worst items we consider for removal is set by a new parameter we introduce: "filterLimit".

We illustrate how the filtering works in Figure 2.3. Here with a filterLimit of 2 rather than considering all the swaps such as in Figure 2.2, we consider only swapping out the worst items RPD wise: item2 and item 3.

The above translates to the pseudo-code described in Algorithm 4 with Profit and Weight being the QKP input and filterLimit the parameter described above. For Figure 2.3 if we set $filterLimit = 2$, the algorithm would thus return $I = [3,2]$ since the item 3 has the smallest RPD and the item 2 has the second smallest RPD.

This new filtering procedure is called to filter x_k on for loop of line 9 of Algorithm 3 to only those belonging to the returned I. Note that x_l from the same line is not filtered. If we take Figure 2.3 for example, the filtering is only applied to the content inside the knapsack but not on the outside. We present its results in the following sub-section.

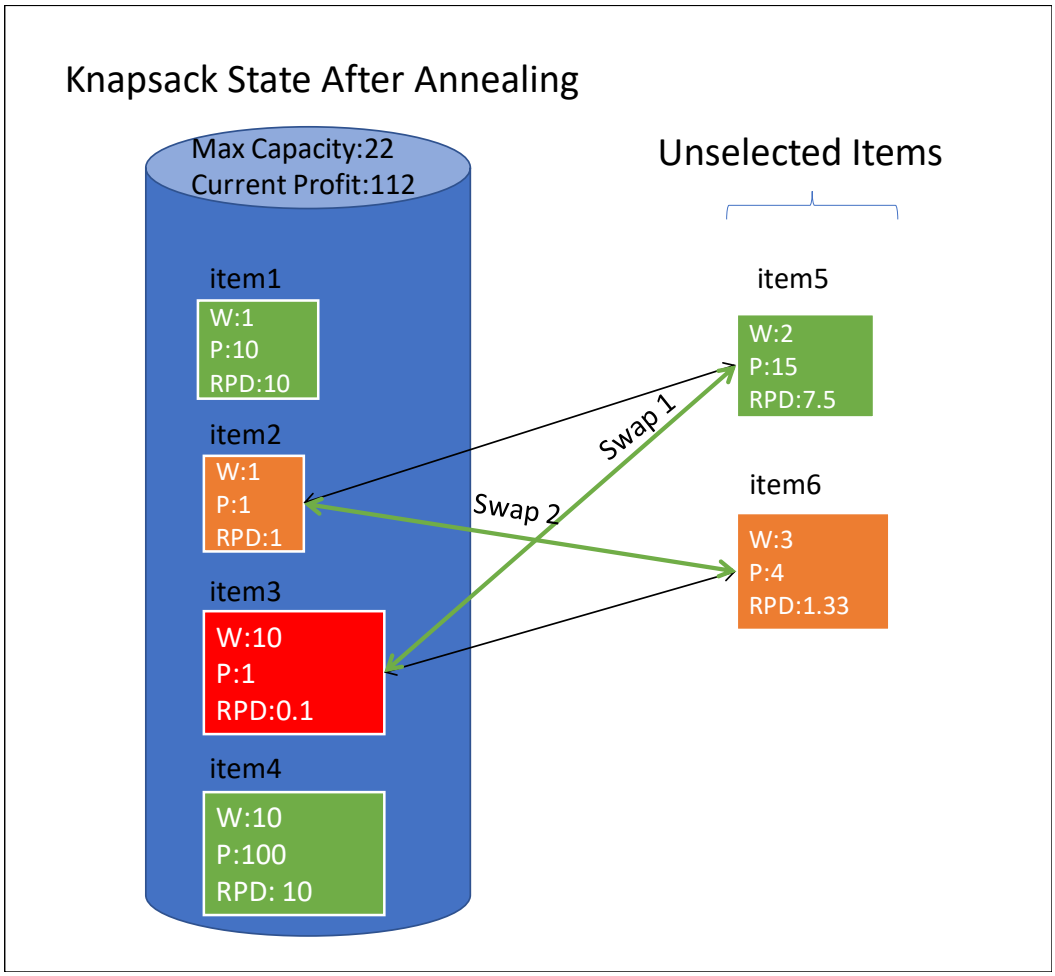

Fig. 2.3 SIA v2 example.

2.4.4 Solution Improvement Algorithm v2 Results

We use the environment described in Section 2.4.2 to experiment with different values for the newly introduced filterLimit parameter as well as the number of DA iterations. We observed its results in Figure 2.4. Best results are obtained with a filterLimit of 15. It represents an increase of 5% over the results from Section 2.4.2. There is also a significant variation of success% depending on the number of DA iterations.

Note that a high filterLimit does not allow to reach the results of SIA v1 which we could consider as "no filter limit". We assume it is due to the filtering being consuming due to the dot product and sorting involved, thus limiting the number of iterations which can be done within 40 seconds. We can also note that reducing the number of DA iterations does not degrade achieved success% much (going from 750 million iterations to 10 million only degrade success% of 7% at the best filterLimit value(15)).

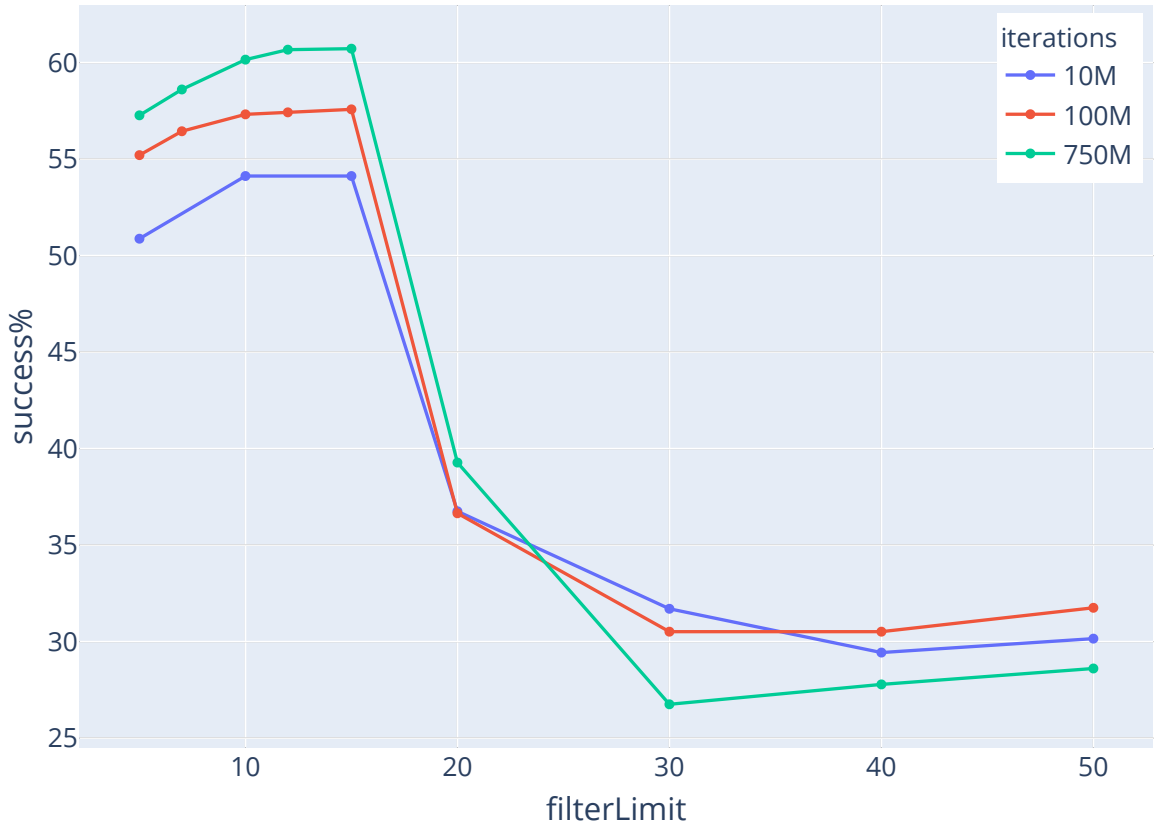

Fig. 2.4 Success% vs SIA v2 filterLimit.

We then focused on observing how success% evolves for a larger sample of DA iterations: [500k,...750M] and values for filterLimit: [7,10,15] as shown in Figure 2.5. We observed that although success% dramatically improves from 500k up to 200M iterations, it nearly plateaus (even on log scale) from 200M up to 750M with the best results being achieved for 500 million DA iterations.

We then focused on how success% evolves with the SIA v2 timeout value, for the sweet spot of 200M DA iterations, filterLimit:15 as shown in Figure 2.6. For small instances (100 items) even the minimum value of SIA v2 timeout we tried (5s) is enough to maximize success%. For average instances (200 items), 15 seconds is enough. For large instances (300 items), success% maxes out at 30 seconds.

2.5 Results Comparison for Each Instance

Here we finally compare the success% achieved for: DA only; SIA v1 timeout 40s; and SIA v2 filterLimit:15, DA iterations:750M, timeout:40s.

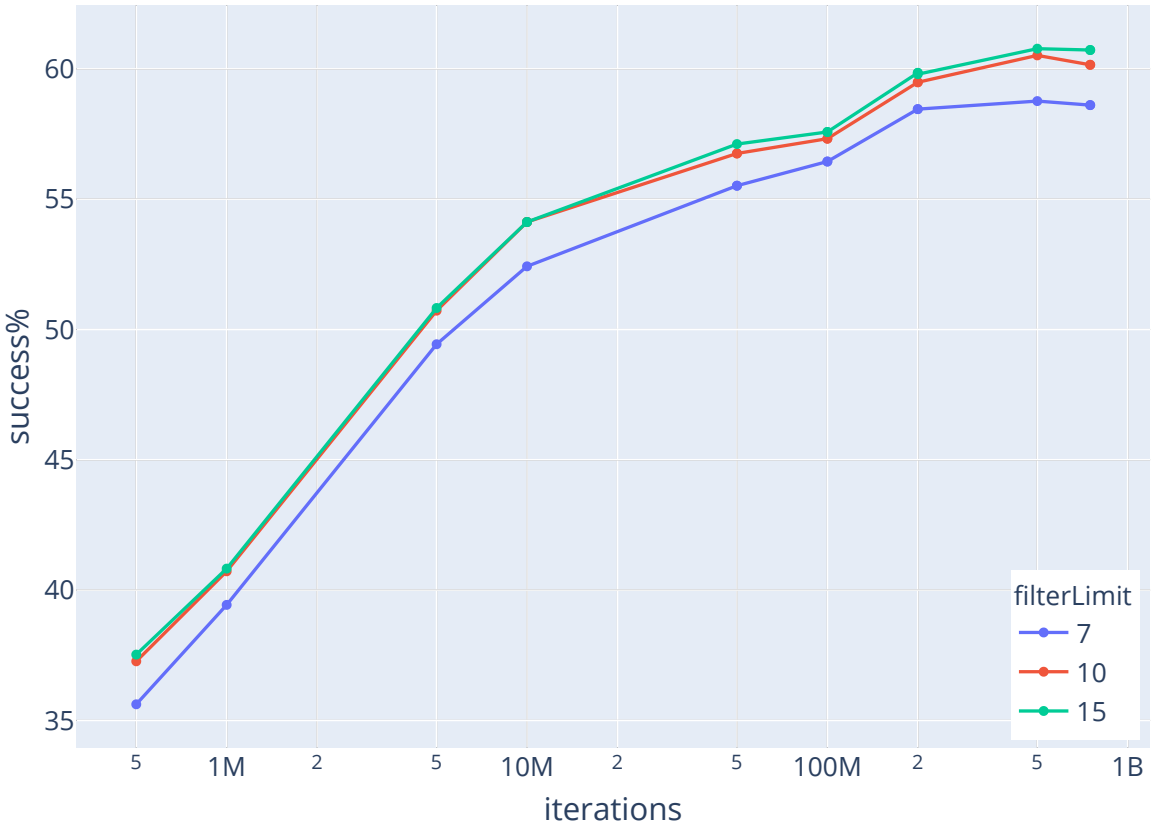

Fig. 2.5 Success% vs DA iterations (log scale).

From Table 2.6 we observe that SIAv1 and v2 improve the success% overall but for instances r_100_25_1, r_100_25_10, r_100_25_3, r_100_25_8, r_100_50_3 and r_100_100_7 DA alone performs better. We consider that it might be due to either the fact that SIA does not consider simply adding items in the knapsack and/or that the penalty Hamiltonian α being too weak makes all replicas converge to the same penalty violating state and from that state, using SIA we can only reach a feasible local minimum.

In future works, we will consider revising SIA to consider item adding and our α increasing policy. We will also keep the best DA solution for fallback in case SIA performs worse than SIA so that SIA results will be in the worst situation equal to DA results.

On Table 2.7 and Table 2.8, we observe a similar trend to Table 2.6: the only instance which used to be solved consistently by DA is not solved by either SIA. Overall success% is improved by SIA v1 and v2, and v2 gives the better results. Since overall SA performance is poor compared to DA as seen in Section 2.2.5 and Solution Improvement Algorithm (SIA) solution quality is assumed to be correlated to input solution quality, SA+SIA is expected to be poorer than DA+SIA and we did not run any SA+SIA experiments. Our proposed DA+SIA method is thus assumed to be superior to SA1 and SA2 as well as SA1+SIA and SA2+SIA.

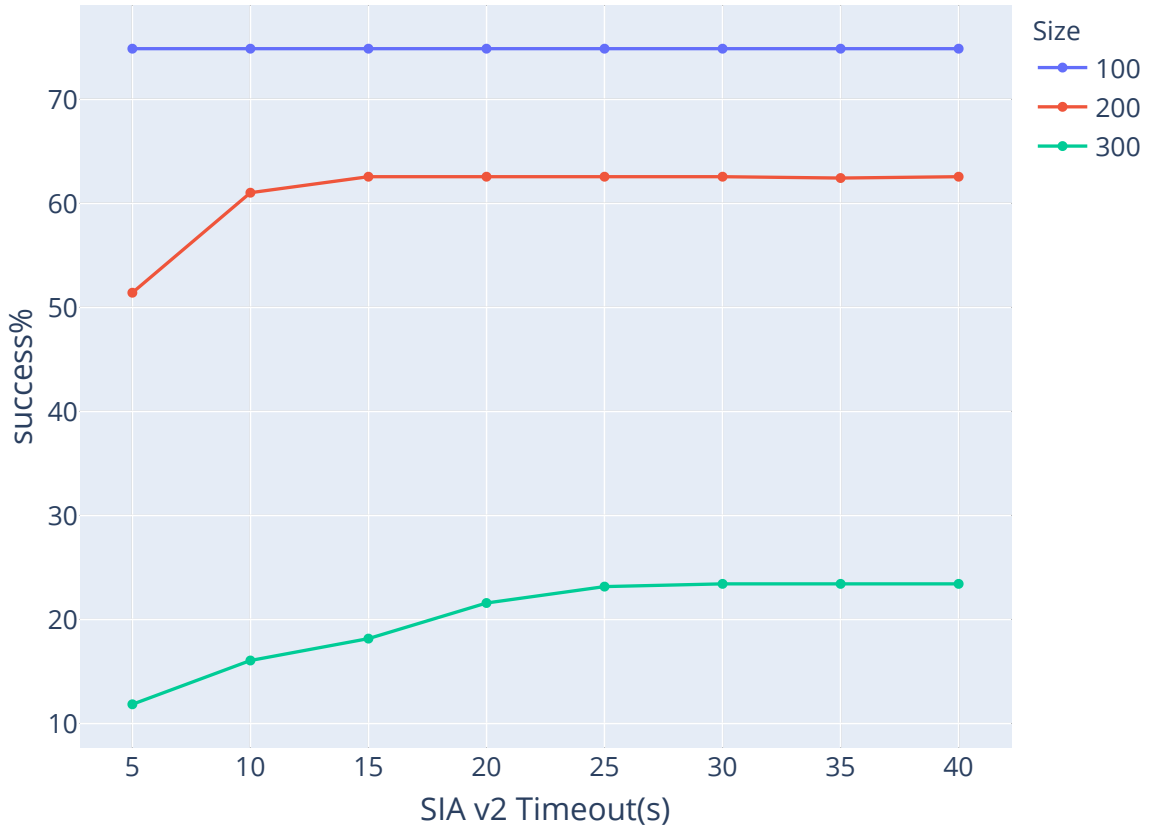

Fig. 2.6 Success% vs SIA v2 timeout.

2.6 Conclusions

In this chapter, a way to visualize the energy landscape between two states of Ising models is proposed. This visualization method is based on the Ising model itself and solutions found which represent states of the Ising model and thus can be applied when using other Ising machines. It helped us conceive our other proposal: a software solution improvement technique coupled with low α value which can compensate for the problem of a low auxiliary variable range giving too much priority to filling the knapsack over finding the solution which generates the most profit. Our software solution improvement algorithms make handling heavy items more efficient by considering swapping out the one with the lowest relative profit density in priority with other items. They operate on the original quadratic knapsack problem formulation, before being mapped to its corresponding Ising model, and thus can operate on solutions found by other methods than Ising machines.

Our experiments showed promising results summarized in Table 2.9. In the "mean" row, we compared the average success% over all instances from Tables 2.1, 2.2 and 2.3 for DA, DA+SIA v1, DA+SIAv2, respectively. We went from an overall success% of 6.7% to a

success% of 60.7%. We believe that our α adjustment policy is reasonable yet coarse grained and thus can potentially be improved. We also believe considering adding items without swapping could improve results. In future works we will also continue to develop novel ways to analyze Ising models and their behavior on Ising machines as well as test our current algorithm on other kind of problems to evaluate how much potential they have for a more general application, not limited to QKP.

Table 2.6 100 variables problem results comparison.

Instance name	Δ	DA only	DA + SIAv1	DA + SIAv2
r_100_25_1	25	5.0	0.0	0.0
r_100_25_10	25	30.0	0.0	0.0
r_100_25_2	**25**	**0.0**	**100.0**	**100.0**
r_100_25_3	**25**	**100.0**	**0.0**	**0.0**
r_100_25_4	**25**	**0.0**	**100.0**	**100.0**
r_100_25_5	**25**	**0.0**	**100.0**	**100.0**
r_100_25_6	25	0.0	0.0	0.0
r_100_25_7	**25**	**0.0**	**100.0**	**100.0**
r_100_25_8	25	20.0	0.0	0.0
r_100_25_9	**25**	**0.0**	**100.0**	**100.0**
r_100_50_1	**50**	**0.0**	**100.0**	**100.0**
r_100_50_10	**50**	**0.0**	**10.0**	**15.0**
r_100_50_2	**50**	**0.0**	**100.0**	**100.0**
r_100_50_3	50	55.0	0.0	0.0
r_100_50_4	**50**	**0.0**	**100.0**	**100.0**
r_100_50_5	**50**	**0.0**	**100.0**	**100.0**
r_100_50_6	50	100.0	100.0	100.0
r_100_50_7	**50**	**0.0**	**100.0**	**100.0**
r_100_50_8	**50**	**0.0**	**100.0**	**100.0**
r_100_50_9	**50**	**0.0**	**100.0**	**100.0**
r_100_75_1	**75**	**0.0**	**100.0**	**100.0**
r_100_75_10	**75**	**0.0**	**100.0**	**100.0**
r_100_75_2	**75**	**0.0**	**45.0**	**50.0**
r_100_75_3	**75**	**0.0**	**100.0**	**100.0**
r_100_75_4	**75**	**0.0**	**80.0**	**90.0**
r_100_75_5	**75**	**10.0**	**100.0**	**100.0**
r_100_75_6	**75**	**0.0**	**100.0**	**100.0**
r_100_75_7	**75**	**0.0**	**100.0**	**100.0**
r_100_75_8	**75**	**0.0**	**100.0**	**100.0**
r_100_75_9	**75**	**0.0**	**30.0**	**30.0**
r_100_100_1	100	0.0	0.0	0.0
r_100_100_10	100	0.0	0.0	0.0
r_100_100_2	**100**	**0.0**	**100.0**	**100.0**
r_100_100_3	**100**	**0.0**	**100.0**	**100.0**
r_100_100_5	**100**	**30.0**	**100.0**	**100.0**
r_100_100_6	**100**	**0.0**	**100.0**	**100.0**
r_100_100_7	100	100.0	0.0	0.0
r_100_100_8	**100**	**0.0**	**100.0**	**100.0**
r_100_100_9	100	0.0	0.0	0.0

Table 2.7 200 variables problem results comparison.

Instance name	Δ	DA only	DA + SIAv1	DA + SIAv2
r_200_25_1	**25**	**0.0**	**100.0**	**100.0**
r_200_25_10	**25**	**0.0**	**100.0**	**100.0**
r_200_25_2	**25**	**0.0**	**100.0**	**100.0**
r_200_25_4	**25**	**0.0**	**100.0**	**100.0**
r_200_25_5	**25**	**0.0**	**60.0**	**75.0**
r_200_25_6	**25**	**0.0**	**85.0**	**70.0**
r_200_25_7	**25**	**0.0**	**100.0**	**100.0**
r_200_25_8	**25**	**0.0**	**100.0**	**100.0**
r_200_25_9	**25**	**0.0**	**100.0**	**100.0**
r_200_50_1	**50**	**0.0**	**100.0**	**100.0**
r_200_50_10	**50**	**0.0**	**50.0**	**30.0**
r_200_50_2	**50**	**0.0**	**10.0**	**5.0**
r_200_50_3	**50**	**0.0**	**100.0**	**100.0**
r_200_50_4	**50**	**0.0**	**100.0**	**100.0**
r_200_50_5	**50**	**0.0**	**100.0**	**100.0**
r_200_50_6	**50**	**0.0**	**85.0**	**85.0**
r_200_50_7	**50**	**0.0**	**40.0**	**20.0**
r_200_50_8	**50**	**0.0**	**30.0**	**100.0**
r_200_50_9	**50**	**0.0**	**100.0**	**100.0**
r_200_75_1	75	0.0	0.0	0.0
r_200_75_10	75	0.0	0.0	0.0
r_200_75_2	**75**	**0.0**	**40.0**	**70.0**
r_200_75_3	75	0.0	0.0	0.0
r_200_75_4	**75**	**0.0**	**100.0**	**100.0**
r_200_75_5	75	0.0	0.0	0.0
r_200_75_6	**75**	**0.0**	**100.0**	**100.0**
r_200_75_7	**75**	**0.0**	**100.0**	**100.0**
r_200_75_8	**75**	**0.0**	**10.0**	**55.0**
r_200_75_9	**75**	**0.0**	**65.0**	**100.0**
r_200_100_1	**100**	**0.0**	**60.0**	**15.0**
r_200_100_10	**100**	**0.0**	**100.0**	**100.0**
r_200_100_2	100	0.0	0.0	0.0
r_200_100_3	100	100.0	0.0	0.0
r_200_100_4	**100**	**0.0**	**100.0**	**100.0**
r_200_100_5	**100**	**0.0**	**5.0**	**30.0**
r_200_100_6	**100**	**0.0**	**100.0**	**100.0**
r_200_100_7	**100**	**0.0**	**10.0**	**30.0**
r_200_100_8	**100**	**0.0**	**50.0**	**90.0**
r_200_100_9	**100**	**0.0**	**5.0**	**15.0**

Table 2.8 300 variables problem results comparison.

Instance name	Δ	DA only	DA + SIAv1	DA + SIAv2
r_300_25_1	**25**	**0.0**	**100.0**	**100.0**
r_300_25_10	**25**	**0.0**	**0.0**	**35.0**
r_300_25_2	**25**	**0.0**	**0.0**	**55.0**
r_300_25_4	**25**	**0.0**	**0.0**	**20.0**
r_300_25_5	25	0.0	0.0	0.0
r_300_25_6	**25**	**0.0**	**5.0**	**5.0**
r_300_25_7	**25**	**0.0**	**0.0**	**60.0**
r_300_25_8	25	100.0	0.0	0.0
r_300_25_9	**25**	**0.0**	**5.0**	**20.0**
r_300_50_1	**50**	**0.0**	**0.0**	**100.0**
r_300_50_10	**50**	**0.0**	**0.0**	**30.0**
r_300_50_2	**50**	**0.0**	**100.0**	**100.0**
r_300_50_3	50	0.0	0.0	0.0
r_300_50_4	**50**	**0.0**	**55.0**	**70.0**
r_300_50_5	**50**	**0.0**	**0.0**	**20.0**
r_300_50_6	50	0.0	0.0	0.0
r_300_50_7	50	0.0	0.0	0.0
r_300_50_8	50	0.0	0.0	0.0
r_300_50_9	50	0.0	0.0	0.0

Table 2.9 Result Summary.

size	DA	DA + SIA v1	DA + SIA v2
100	11.5	68.3	68.8
200	2.6	61.7	66.4
300	5.3	13.9	32.4
mean	6.7	55.0	60.7

Chapter 3

Cardinality Constrained Portfolio Optimization on an Ising Machine[1]

3.1 Introduction

The mean-variance portfolio selection [41], introduced by Markowitz, is a quantitative approach which aims at selecting assets which maximize return while minimizing their variance (also known as risk). From an operations research perspective, it is a simple quadratic problem from which optimal solution can easily be computed using quadratic programming [15]. However, it is often more practical to limit the number of assets to consider for portfolio selection, Chang et al. introduced the cardinality constrained mean-variance portfolio optimization problem (CCMVPOP) [15] which, as the name implies, introduces a cardinality constraint to limit the number of asset kinds to be considered. The introduction of this cardinality constraint has been proven to make the problem NP-hard [54] and thus challenging. Since CCMVPOP has been introduced, over the last 20 years metaheuristics based approaches have been successful when cardinality constraint limits to approximately 10 to 15 asset kinds and exact approaches have been successful for 5 asset kinds and below [29]. Well used metaheuristics for this problem are tabu search [24] and simulated annealing[37].

Over the past 10 years, quantum-based Ising machines [36] and non-quantum based Ising machines [42], [28], [62] have shown potential when solving certain kinds of combinatorial problems formalized as a quadratic unconstrained binary optimization (QUBO) form.

Yet their limits have been shown on problems having certain kinds of constraints [48], hybrid software-hardware technologies such as Fujitsu's third generation Digital Annealer (DA) [46], [19] have recently emerged to overcome such limitations. DA is now an Ising

[1]Technical contents in this chapter have been presented in the publication ⟨4⟩.

machine-software system which can handle a binary quadratic program (BQP), expressed using a limited set of constraints, in contrast to conventional Ising machines [36], [42], [28], [62] which can only handle QUBO.

CCMVPOP is a quadratic problem in nature, thus we want to evaluate how such an Ising machine-software system performs on this problem. Although researches have been done on using Ising machines for financial applications such as index tracking with cardinality constraints [1], [21] and trading trajectory problem [53], CCMVPOP on an Ising machine or system has not been studied before.

As we will show in detail in Section 3.2, some of CCMVPOP variables are real numbers. To encode those variables as binary variables, some established techniques exist such as unary encoding, binary encoding and onehot encoding and their impact on convergence when using Ising machines have been studied in [59] on the quadratic knapsack problem (QKP). However, no studies have been done for CCMVPOP from the viewpoint of integer encoding techniques.

In this chapter, we firstly propose a BQP formulation of the CCMVPOP based on well known integer to binary variable encoding. Secondly, we further propose a new base10 encoding where we have groups, composed of 10 binary variables for each power of 10, which allows faster convergence to optimal portfolio composition. Lastly, we compare each integer binary encoding formulations through experiments on the data used in [15] available at [7],[8].

The contributions of this chapter are summarized as follows:

1. We propose an effective formulation of the CCMVPOP for Ising machines.
2. In the formulation of the CCMVPOP, base10 integer to binary variable encoding for effectively solving the CCMVPOP by Ising machines is newly proposed.
3. Experimental evaluations demonstrate the effectiveness of the proposed CCMVPOP formulation. Particularly, the proposed base10 encoding gives the best results for large CCMVPOP solved.

The rest of this chapter is organized as follows: Section 3.2 explains the general CCMVPOP formulation. In Section 3.3 proposed BQP formulations, including our proposed base10 encoding, are exposed. In Section 3.4, the performance for the proposed BQP formulations on well-known instances from [15] is measured. Lastly, Section 3.5 gives several concluding remarks.

3.2 Cardinality Portfolio Optimization Problem With

Original Formulation In this section we will define the CCMVPOP original model from [15].

Let N be the number of assets to consider, K be the fixed number of assets which must compose the portfolio, μ_i be the expected return of asset i $(1 \leq i \leq N)$, σ_{ij} be the covariance between assets i and j $(1 \leq i, j \leq N)$, and ϵ_i and δ_i be respectively the minimum and maximum proportion of a chosen asset i. The decision variables are w_i which represent the proportion $(0 \leq w_i \leq 1)$ of held of asset i for a given solution. We also use a z_i binary variable to indicate if an asset i is chosen or not. The CCMVPOP is formulated as follows:

$$\text{minimize} \quad \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij} - (1-\lambda) \sum_{i=1}^{N} w_i \mu_i \tag{3.1}$$

subject to

$$\sum_{i=1}^{N} w_i = 1 \tag{3.2}$$

$$\sum_{i=1}^{N} z_i = K \tag{3.3}$$

$$\epsilon_i z_i \leq w_i \leq \delta_i z_i, \quad i = 1, \ldots, N \tag{3.4}$$

$$z_i \in \{0, 1\}, \quad i = 1, \ldots, N \tag{3.5}$$

Equation (3.1) represents the two objectives of CCMVPOP. The first objective is minimizing the risk of the chosen assets of the porfolion, where the risk is represented by the sums of covariance between all pairs i,j of chosen assets, $\sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij}$. The second objective is maximizing return of chosen assets, where return is represented by the sum of expected return of each asset i, $\sum_{i=1}^{N} w_i \mu_i$. Equation (3.2) states w_i is a proportion. Equation (3.3) is the cardinality constraint which forces the number of chosen assets to be equal to K. Equation (3.4) bounds the proportion of a chosen asset w_i to be within a given ϵ_i and δ_i. Equation (3.5) states z_i is a binary variable.

For K values close to N, this problem can easily be solved using quadratic programming, for $K \leq 5$ it can be solved using exact methods, finally most works involving metaheuristics are done using $K = 10$ which is believed to be the sweet spot for them [54]. Historically, Ising machines explore QUBO solution search space in order to lower its cost, using various strategies such as quantum annealing [36] or simulated annealing with parallel tempering [42]. Those machines fall within the category of metaheuristics. We thus for the rest of this chapter choose $K = 10$.

Although the problem is inherently a bi-objective optimization problem, current Ising machines or systems target a single objective only. Accordingly, we use the weighted

formulation in equation (3.1) which introduces a parameter λ to balance the importance given to each objective.

3.3 Binary Quadratic Program Formulation for Cardinality Mean-Variance Portfolio Optimization Problem Utilizing Ising Systems

As Ising machines or systems can only handle binary variables, we need to encode w_i in equations (3.1), (3.2) and (3.4) as a set of binary variables. The first step is to convert all w_i to integer variables w'_i. We thus propose to multiply equation (3.2) by a strictly positive integer "budget" number B. The larger B is, the larger the number of possible w_i can be represented. Thus equations (3.2) and (3.4) become:

$$\sum_{i=1}^{N} w'_i = B, \quad w'_i = Bw_i \quad B \in \mathbb{Z}, \quad B > 0 \tag{3.6}$$

$$\epsilon'_i z_i \leq w'_i \leq \delta'_i z_i, \quad \epsilon'_i = B\epsilon_i, \quad \delta'_i = B\delta_i \tag{3.7}$$

The next step is encoding w'_i as a set of binary variables. There are three main ways of doing so, described in [59]: binary encoding, unary encoding and onehot encoding. We adapt those encodings, which describe how to encode an integer between 0 and a constant C, to reflect the bounds described in equation (3.7). After that, we newly propose a base10 encoding for Ising systems, which must be superior to the existing encodings in terms of converging faster to dominating solutions risk or profit wise.

3.3.1 Binary encoding

Binary encoding is described as follows:

$$w'_i = \sum_{r \in V} r \times b_{i,r} + \epsilon'_i z_i \tag{3.8}$$

where $m = \lceil \log_2(\delta'_i - \epsilon'_i + 1) \rceil$ is the number of $b_{i,r}$ required per w'_i, $b_{i,r}$ is a binary variable, w'_i is an integer variable, and $V = \{1, 2, 4, ..., 2^{m-2}, \delta'_i - \epsilon'_i - (2^{m-1} - 1)\}$. Since we need an extra binary variable z_i to express when an asset is selected or not, we encode the minimum proportion ϵ'_i there. We thus only need to cover values from 0 to $C = \delta'_i - \epsilon'_i$ with each $b_{i,r}$.

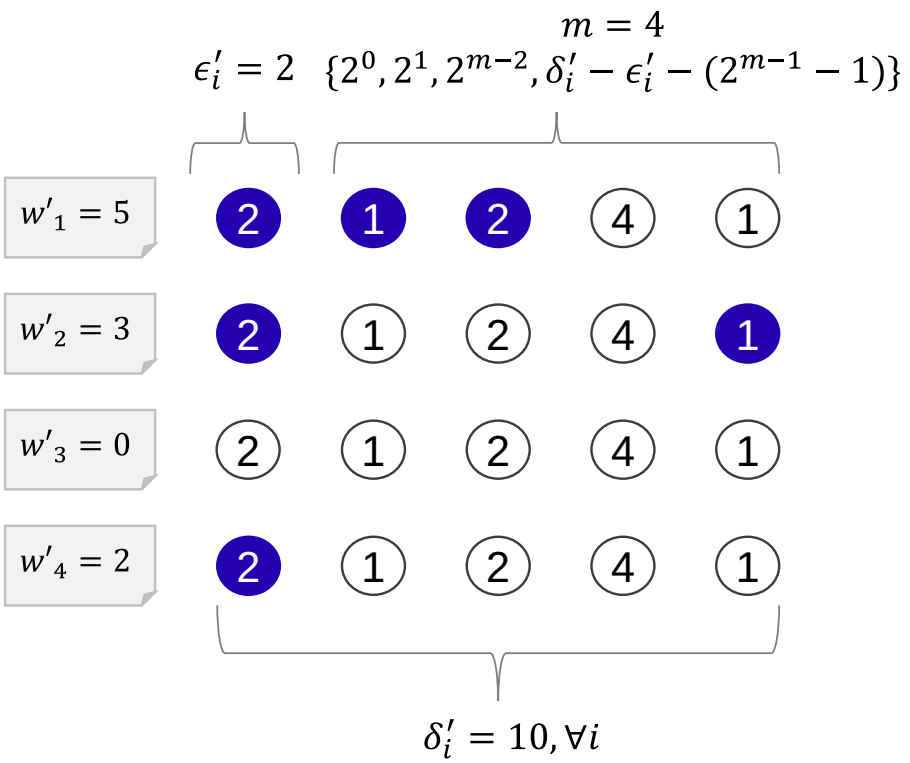

Fig. 3.1 $b_{i,r}$ configuration in regard to w'_i for binary encoding. Circled values represent r or ϵ'_i. A blue circle represents a variable at 1, a white circle 0.

We illustrate the binary variables $b_{i,r}$ configuration in regard to the integer variable w'_i they represent in Fig. 3.1. In this figure, $w'_1 = 5$ is represented by $2+1\times1+2\times1+4\times0+1\times0=5$, where $\epsilon'_1 = 2$. $w'_2 = 3$ is represented by $2+1\times0+2\times0+4\times0+1\times1=3$, where $\epsilon'_2 = 2$.

3.3.2 Onehot encoding

Onehot encoding is described as follows:

$$w'_i = \sum_{r\in V} r \times b_{i,r} \tag{3.9}$$

$$\text{s.t:} \sum_{r\in V} b_{i,r} = 1 \tag{3.10}$$

where $V = \{0, \epsilon'_i, \epsilon'_i + 1, \ldots, \delta'_i\}$. As each $b_{i,r}$ covers a possible value of w'_i, it requires more binary variables than binary encoding. On the other hand, it does not need the extra z_i

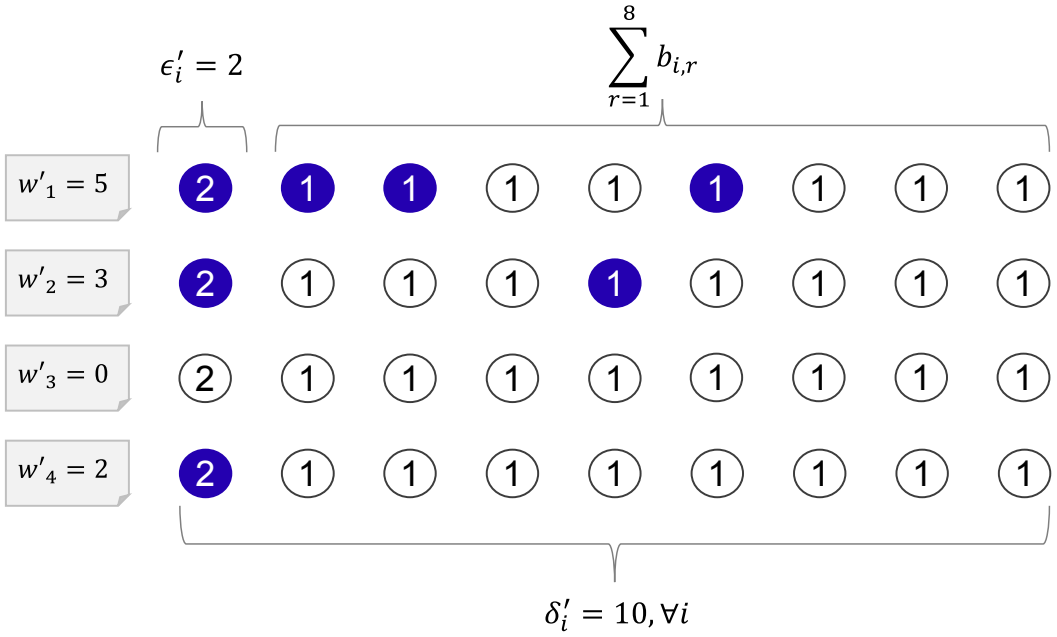

Fig. 3.2 $b_{i,r}$ configuration in regard to w'_i for unary encoding. Circled values represent r or ϵ'_i. A blue circle represents a binary variable at 1, a white circle 0.

variable, as we can use the $b_{i,0}$ variables to express the following constraint in replacement to equation (3.3):

$$\sum_{i=1}^{N} z_i = K \Leftrightarrow N - \sum_{i=1}^{N} b_{i,0} = K \tag{3.11}$$

3.3.3 Unary encoding

Unary encoding is described as follows:

$$w'_i = \sum_{r=1}^{m} b_{i,r} + \epsilon'_i z_i \tag{3.12}$$

where $m = \delta'_i - \epsilon'_i$ is the number of binary variables $b_{i,r}$ required per integer w'_i and $b_{i,r}$ is a binary variable. Note that as in the binary encoding, z_i is still required to express if an asset is chosen or not, we thus encode ϵ'_i value there as well. It requires N variables less than onehot encoding as there is no need to encode value 0 as it is represented by having all $b_{i,r}$ set to 0. We illustrate the same example as Fig. 3.1 for binary encoding with unary encoding in Fig. 3.2.

3.3.4 Base10 encoding proposal

Each encoding described before has a trade-off between the number of $b_{i,r}$'s needed to represent w'_i, the number of $b_{i,r}$'s which need to be flipped to adjust w'_i, and impact on the cost function when flipping every $b_{i,r}$. Binary encoding is the most compact but to adjust w'_i by a value 1 might require at worst $\log_2(\delta'_i - \epsilon_i + 1')$ of $b_{i,r}$ adjustments and $b_{i,r}$ with higher r index will strongly impact the cost function making them hard to adjust. Unary encoding is less compact, redundant as one integer value can be represented by different combination of $b_{i,r}$ set to 1, but on the other hand adjusting each $b_{i,r}$ has an impact of only 1 on the cost function. Unary encoding has been proven in [59] to be the most efficient for the quadratic knapsack problem. Onehot encoding, while being the less compact, can allow to go between any w'_i value within two $b_{i,r}$ flips. Since the Ising machine-software system [46] has a dedicated architecture to accelerate solving of problems which have onehot constraints, it makes onehot encoding relevant for comparison.

With the above in mind, we also wanted to evaluate if a better trade-off between compactness, number of binary variable adjustments required per integer value adjustment, cost function landscape smoothness could be achieved. We thus propose a base10 encoding where all variables corresponding to the value of a digit, will be expressed as a onehot group:

$$w'_i = \overbrace{\sum_{q=0}^{l-1}\sum_{r=0}^{9} r \times 10^q b_{i,r,q} + \sum_{r=0}^{k} r \times 10^l b_{i,r,l}}^{\text{general terms}} + \overbrace{(y-c)b_{i,k+1,l}}^{\text{remainder term}} + \epsilon'_i z_i \tag{3.13}$$

$$y = \delta'_i - \epsilon'_i \quad \text{and} \quad c = \sum_{q=0}^{l-1} 9 \times 10^q \tag{3.14}$$

$$\text{s.t.} \sum_{r=0}^{9} b_{i,r,q} = 1 (0 \le q < l) \quad \text{and} \quad \sum_{r=0}^{k+1} b_{i,r,l} = 1 \tag{3.15}$$

where $b_{i,r,q}$ is a binary variable and, l and k are the constants satisfying $c + k10^l \le y < c + (k+1)10^l$. y is the integer quantity to express as binary variables.

For instance, assume that $\delta'_i = 350$ and $\epsilon'_i = 0$. Then w'_i is expressed by:

$$w'_i = \sum_{q=0}^{1}\sum_{r=0}^{9} r \times 10^q b_{i,r,q} + \sum_{r=0}^{2} r \times 10^2 b_{i,r,2} + 251 b_{i,3,2}$$

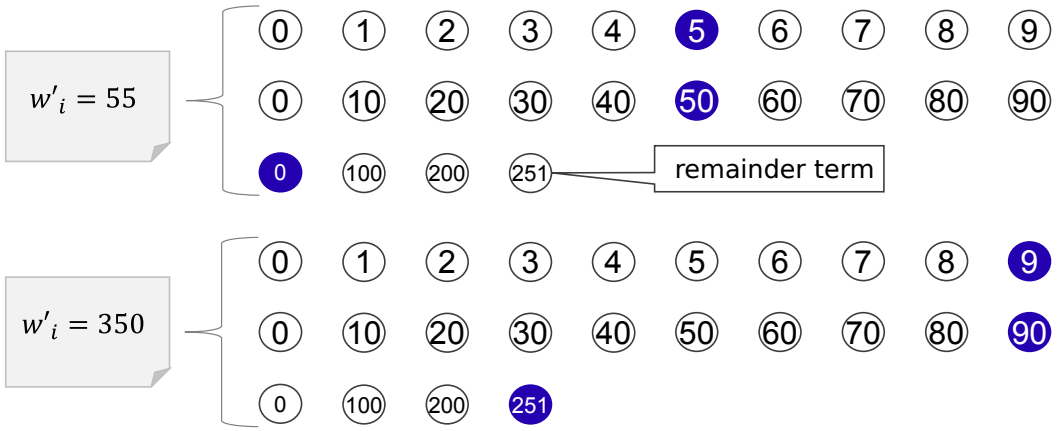

Fig. 3.3 $b_{i,r}$ configuration in regard to w'_i for the proposed base10 encoding. Circled values represent r or ϵ'_i. A blue circle represents a binary variable at 1, a white circle 0.

In this case, we have $l = 2$, $k = 2$, $c = 99$, and $y = 350$. We have the following three onehot groups for each asset i: $\{b_{i,0,0}, b_{i,1,0}, \ldots b_{i,9,0}\}$, $\{b_{i,0,1}, b_{i,1,1}, \ldots b_{i,9,1}\}$ and $\{b_{i,0,2}, b_{i,1,2}, b_{i,2,2}, b_{i,3,2}\}$ as we illustrate in Fig. 3.3.

The coefficient $y - c = 251$ of $b_{i,3,2}$ shows the remainder term. If $w'_i = 55$, then the three onehot groups become {0,0,0,0,1,0,0,0,0,0}, {0,0,0,0,1,0,0,0,0,0}, and {1,0,0,0}. If $w'_i = 350$, then the three onehot groups become {0,0,0,0,0,0,0,0,0,1}, {0,0,0,0,0,0,0,0,0,1}, and {0,0,0,1}. Note that, in this encoding, several integer values can be encoded in two ways, which causes no problems in the BQP formulation.

We will compare each encoding performance in Section 3.4.

3.3.5 Binary quadratic program input

In this section, we utilize an Ising machine-software system called the third generation Digital Annealer (DA) [46] for solving CCMVOP. The input of this system is composed of one objective function, H_{obj}, in the QUBO form as well as one constraint penalty function, H_{const}, also in QUBO form.[2].

[2]The Ising-machine-software system[46] used can also take up to 10,000 linear inequality constraints but we do not use inequality constraints in this dissertation

Objective function H_{obj}

The objective function described in equation (3.1) needs to reflect the binary variable encoding. Thus, the objective QUBO, or Hamiltonian becomes:

$$H_{obj} = \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} w'_i w'_j \sigma_{ij} - (1-\lambda) \sum_{i=1}^{N} w'_i \mu_i \tag{3.16}$$

Penalty

function H_{const} For the constraint QUBO, or constraint Hamiltonian H_{const}, solutions found need to evaluate to 0 for them to be judged feasible. If a problem is composed of several constraint Hamiltonians, we sum them as a feasible solution should evaluate to 0 for each of them, thus their sum evaluating to 0 as well. Now let us describe each constraint Hamiltonian needed for CCMVPOP.

The w'_i constraint described in equation (3.6), equivalent to the original constraint described in equation (3.2), is expressed within each corresponding encoding. Onehot encoding constraints described in equations (3.10), (3.11) and (3.15) need one extra step, expressed by the three following Hamiltonians:

$$H_{oh} = \sum_{i=1}^{N} \left(\sum_{r \in V} b_{i,r} - 1 \right)^2 \tag{3.17}$$

$$H_{ohK} = \left(\sum_{i=1}^{N} b_{i,0} - (N-K) \right)^2 \tag{3.18}$$

$$H_{oh10} = \sum_{i=1}^{N} \left(\sum_{q=0}^{l-1} \left(\sum_{r=0}^{9} b_{i,r,q} - 1 \right)^2 + \left(\sum_{r=0}^{k+1} b_{i,r,l} - 1 \right)^2 \right) \tag{3.19}$$

For all other encodings than onehot encoding, the cardinality constraint is expressed by the two following Hamiltonians:

$$H_{chosen} = \sum_{i=1}^{N} \left(\sum_{r \in V} b_{i,r} (1 - z_i) \right) \tag{3.20}$$

$$H_K = \left(\sum_{i=1}^{N} z_i - K \right)^2 \tag{3.21}$$

Table 3.1 Number of variables for each encoding and instance.

instance	N	K	onehot	unary	binary	proposed base10
port1	31	10	3131	3100	248	651
port2	85	10	8585	8500	680	1785
port3	89	10	8989	8900	712	1869
port4	98	10	9898	9800	784	2058
port5	225	10	22725	22500	1800	4725

Equation (3.20) ensures that for any binary variable $b_{i,r}$ from the corresponding w'_i to be chosen, its corresponding z_i has to be chosen or else H_{chosen} will be positive and thus constraint violated.

Finally, common to all encodings, budget equality constraint becomes:

$$H_{budget} = \left(\sum_{i=1}^{N} w'_i - B \right)^2 \tag{3.22}$$

Then the constraint term H_{const} is constructed by summing up equations (3.20)–(3.22) for binary encoding, equations (3.17), (3.18), and (3.22) for onehot encoding, equations (3.20)–(3.22) for unary encoding, and (3.19)–(3.22) for the proposed base10 encoding.

3.4 Encoding and Equality Formulation Experimental Evaluations

The goal of our experiments is to show the formulation in equations (3.16)–(3.22) is effective enough for an Ising machine and also to show which encoding allows to converge faster to the best solution for a given public instance of CCMVPOP published in [15]. We align ourselves with [15] and take K=10, ϵ_i=0.01 and δ_i=1 to generate our BQPs. Before explaining in details our experiment protocol, let us have a look at the number of assets, in each available instance as well as the number of binary variables it translates to in Table 3.1. In this table, the "instance" column refers to the instance's name according to [15]. N column represents the number of considered assets in the corresponding instance. K column represents the number of asset kinds which must be chosen. All other columns represent the number of binary variables required for each their corresponding integer binary encoding. We can already observe binary encoding is in all instances one order of magnitude more compact

than onehot and unary encodings. We can also observe binary encoding is 2.625 times more compact than the proposed base10 encoding.

To run our experiments, we use DA[46] with its default settings, except for base10 and onehot encodings, which thus possess onehot constraints. For those we use the DA onehot constraint feature which allows acceleration when solving problems which have constraints similar to equation (3.10) as well as changing "gs_level" from 5 to 0 as suggested in [60]. We set the run time for each instance solving to 150 seconds. We compare results for λ=0 (maximizing profit only), λ=0.5 and λ=1 (minimizing risk only) in equation (3.1). We measure the lowest value of H_{obj} achieved among found feasible solutions. We use 20 different random seeds.

We separate the results for each instance. For each (instance, λ) pair, we take the lowest value of H_{obj} found across all encodings as reference value H_{objRef}. We measure the gap GAP defined as follows for each run:

$$GAP=\frac{H_{obj}(W')-H_{objRef}}{H_{objRef}} \tag{3.23}$$

where $H_{obj}(W')$ is the value of H_{obj} for the best solution W' found for a given encoding for a random seed, for an instance, for a given λ. We then aggregate the mean of the gap for each given encoding across all seeds, named "gap" for each instance in Table 3.2. For encodings which have reached H_{objRef} we measure the ratio of seeds which have reached it as "success". Finally, for encodings which have a non-zero success ratio, we aggregate on all random seeds the mean and standard deviation of the time needed to reach H_{objRef} as "TTB_mean" and "TTB_std", where TTB stands for "time to best".

For instance port1, we observe that the unary encoding, contrary to the trend observed in [59], is performing the worst with TTB one order of magnitude higher than other encodings for all λ values. For λ=1.0, unary encoding is the only encoding needing more than 10 seconds to converge to H_{objRef}. Although binary encoding is also performing poorly for λ=0 and λ=0.5, with TTB_mean almost 20 times worse than the best performing encoding, which is the proposed base10, it is the best performing encoding for λ=1, converging to H_{objRef} 1.5 times faster than the second best, base10 encoding. Both onehot and base10 encodings are strong performers for all λ values, always converging to H_{objRef} in less than 10 seconds. When comparing those two encodings, the proposed base10 encoding converges more than 17 times faster to H_{objRef} for λ=0 and λ=0.5 and 1.5 times slower for λ=1. Thus we can make the claim for port1 instance, base10 encoding is the better performing encoding overall. We can also make the assumption that the difference of performance for encoding such as binary encoding depending on λ value, could be explained by how the search space

landscape changes when we adjust λ. We also note that for binary encoding, 1 seed could not reach H_{objRef} solution for λ=0.5.

For instance port2, trends overall remain similar with base10 encoding performing the best for λ=0 and λ=0.5. For TTB_mean, "NaN" values implies the corresponding encoding could not reach the best found solution H_{objRef} by other encodings and thus TTB could not be evaluated. "NaN" only present in TTB_std column means only 1 seed could reach H_{objRef} thus standard deviation could not be computed. We also note that binary encoding cannot converge to H_{objRef} for λ=1 (minimizing risk only). Binary encoding could only reach H_{objRef} for 1/20 seed for λ=0.0 and 3/20 seeds for λ=0.5. We also note base10 success ratio is 35% for λ=1. Only onehot encoding can consistently reach H_{objRef} for all λ values. Port2 results thus suggest its landscape is a better fit for onehot encoding solution quality wise. Note that proposed base10 encoding is the fastest for λ=0, being one or two order of magnitude faster than other encodings, and λ=0.5 where it's 1.3 times faster than onehot encoding.

Instance port3 has a similar number of assets to port2 (89 vs 85). We can observe similar trend to port2 for binary encoding as it cannot converge at all to H_{objRef} solution for λ=0 and λ=0.5 and has a low success ratio on λ=1. Only onehot encoding achieves 100% convergence to H_{objRef} solution for all λ values. Unary encoding is close when looking at success ratio as it achieves 100%, 100% and 80% respectively for λ=0, λ=0.5 and λ=1. TTB wise onehot is approximately three times faster than unary encoding for all λ values making it the superior encoding overall. We note that the proposed base10 encoding performs the best for λ=0 being 16.9 times faster than onehot encoding, the second best.

Instance port4 is larger than port3, with 98 assets. The main difference is unary encoding's poor solution quality as its success ratio is lower than 40% across all λ values. We observe the same trend of onehot encoding being the best all around with a success ratio higher than 80% across all λ values. Again, the proposed base10 encoding is the best for λ=0, being 8.7 times faster than onehot but has success ratio below 50% for other λ values. Binary performs poorly overall again with success ratio below 25% for all λ values.

Instance port5 is more than double the size of previous instances with N=225 and shows drastically different trends. The large number of variables required for unary and onehot encoding largely impedes their performance as they can never reach the H_{objRef} value nor even lead to feasible solutions, which explains the "NaN" values for their gap. Base10 encoding has an advantage over binary encoding for all λ values as binary encoding can never reach H_{objRef}.

Overall, the experimental evaluations demonstrate the effectiveness of the proposed CCMPOP formulation. Particularly, the proposed base10 encoding gives the best results when the large-sized CCMVPOP is solved with N=225 as shown in Table 3.2.

3.5 Conclusion

In this chapter, we first proposed a BQP formulation for the CCMVPOP and furthermore we proposed a new base10 integer binary encoding. In the case of CCMVPOP we showed that among the well-known encodings, onehot encoding leads most of the time to faster TTB than unary or binary encodings for the smallest 4 instances. Furthermore, we showed there is room to come up with new encoding schemes. In fact, our proposed base10 encoding could lead to TTB between same level and 10 times faster than onehot encoding in some cases. The proposed base10 encoding is also unequivocally the best for the largest instance, port5.

In the future, we will extend our proposed base10 encoding to other bases and study the impact of the used base for convergence in regard to CCMVPOP as well as other combinatorial problem involving integer variables.

Table 3.2 Encoding performance comparison

instance	λ	encoding	gap [%]	success [%]	TTB_mean [sec]	TTB_std [sec]
port1	0.0	onehot	0.0	100.0	6.5	0.0
		unary	0.0	100.0	30.5	2.1
		binary	0.0	100.0	23.6	20.9
		base10	0.0	100.0	**0.6**	0.1
	0.5	onehot	0.0	100.0	6.5	0.1
		unary	0.0	100.0	32.2	0.3
		binary	0.0	95.0	23.3	11.5
		base10	0.0	100.0	**1.3**	0.6
	1.0	onehot	0.0	100.0	7.0	0.1
		unary	0.0	100.0	34.6	0.3
		binary	0.0	100.0	**1.5**	0.7
		base10	0.0	100.0	2.3	1.6
port2	0.0	onehot	0.0	100.0	55.9	1.0
		unary	0.0	100.0	123.5	0.5
		binary	0.5	5.0	141.4	NaN
		base10	0.0	100.0	**3.9**	1.2
	0.5	onehot	0.0	100.0	40.2	1.5
		unary	0.0	100.0	99.0	5.2
		binary	0.1	15.0	81.0	11.1
		base10	0.0	100.0	**32.9**	19.0
	1.0	**onehot**	0.0	100.0	**52.4**	2.2
		unary	0.1	30.0	121.8	7.4
		binary	0.1	0.0	NaN	NaN
		base10	0.0	35.0	82.8	29.3
port3	0.0	onehot	0.0	100.0	62.4	1.0
		unary	0.0	100.0	139.1	3.0
		binary	1.2	0.0	NaN	NaN
		base10	0.0	100.0	**3.7**	0.4
	0.5	**onehot**	0.0	100.0	**34.0**	3.4
		unary	0.0	100.0	104.6	3.0
		binary	0.0	0.0	NaN	NaN
		base10	0.0	45.0	83.3	42.5
	1.0	**onehot**	0.0	100.0	**47.4**	1.2
		unary	0.1	80.0	131.1	7.2
		binary	0.0	25.0	93.3	41.6
		base10	0.2	25.0	62.3	63.8
port4	0.0	onehot	0.0	100.0	78.1	1.6
		unary	47.4	0.0	NaN	NaN
		binary	1.0	0.0	NaN	NaN
		base10	0.0	100.0	**9.1**	12.6
	0.5	**onehot**	0.0	80.0	**74.8**	26.6
		unary	0.1	25.0	131.3	18.1
		binary	0.2	5.0	19.6	NaN
		base10	0.2	35.0	101.2	44.8
	1.0	**onehot**	0.0	**90.0**	**73.0**	16.1
		unary	0.4	40.0	138.7	2.7
		binary	0.1	25.0	49.5	22.9
		base10	0.2	50.0	74.9	42.2
port5	0.0	onehot	NaN	0.0	NaN	NaN
		unary	NaN	0.0	NaN	NaN
		binary	2.2	0.0	NaN	NaN
		base10	0.0	**100.0**	14.3	5.3
	0.5	onehot	NaN	0.0	NaN	NaN
		unary	NaN	0.0	NaN	NaN
		binary	0.3	0.0	NaN	NaN
		base10	0.0	**100.0**	30.7	18.3
	1.0	onehot	NaN	0.0	NaN	NaN
		unary	NaN	0.0	NaN	NaN
		binary	0.4	0.0	NaN	NaN
		base10	0.0	**100.0**	43.8	18.2

Chapter 4

Fast Hyperparameter Tuning for Ising Machines[1]

4.1 Introduction

Ising machines such as [36], [42], [28], [62] or Ising machine-software system such as [46], have a varying number of hyperparameters and tuning them is a time and money consuming task as most machines' usage price are significant. Popular automated tuning techniques are random sampling where hyperparameters values are picked randomly, grid-search where predefined combinations of hyperparameters are explored, and manual search done by experts.

A more sophisticated tuning approach which takes into account past results to find the best hyperparameter values is Tree-structured Parzen Estimator (TPE), which is a sequential model-based optimization approach originally created and shown to be highly efficient for neural network hyperparameter tuning [9]. TPE has recently been shown to be efficient for QUBO penalty coefficient tuning [57, 56, 33, 27, 38] but not for Ising machine hyperparameters themselves. Tuning of QUBO penalty coefficient is an extensive topic and area of research. The reason for its popularity is QUBO penalty coefficient is often the most critical parameter to improve performance. In fact, this coefficient tuning will result in higher performance across all Ising machines [61, 4, 20]. However tuning of hyperparameters of Ising machine themselves is not covered as extensively, thus our motivation to research black-box tuning approach such as random sampling or TPE.

In this chapter, after analyzing tuning performance of random sampling and TPE alone, a new convergence acceleration method for TPE called FastConvergence is proposed. It aims at limiting the number of required TPE trials to reach best performing hyperparameter

[1]Technical contents in this chapter have been presented in the publication ⟨3⟩ and have been applied to the patent office in the patent ⟨11⟩.

values combination. We compare FastConvergence to previously mentioned well-known hyperparameter tuning techniques to show its effectiveness. Our main contributions are:

- We show that TPE converges to better parameters than random sampling and how much improvement it can yield compared to default parameters.
- We propose a method called FastConvergence, which allows TPE to converge to better parameters faster than TPE alone.

In Section 4.2, we will define Ising machine performance and how to evaluate Ising machines hyperparameters in regard to said performance. In Section 4.3, we explain what tuning techniques can be used as baselines and what kind of objective they should target. In Section 4.4, we describe our main contribution which aims at accelerating TPE convergence. In Section 4.5, we show the efficiency of our proposed method using well-known combinatorial optimization problems: Travel Salesman Problem(TSP) and Quadratic Assignment Problem (QAP). Finally Section 4.6, gives several concluding remarks.

4.2 Evaluating Ising Machine Hyperparameters

To illustrate the importance of hyperparameter tuning, we first define Ising machines performance and what problem they solve. BQP solved by DA are defined as follows:

$$\text{minimize} \quad E(x)=x^T Q_{obj} x \tag{4.1}$$

$$\text{subject to} \quad x^T Q_{pen} x=0 \tag{4.2}$$

$$W_i x \leq C_i, \quad \forall i \tag{4.3}$$

where $x=(x_1,\ldots,x_m)$ is an m-dimensional vector of binary variables. Q_{obj}, an $m\times m$ matrix called a QUBO matrix, represents the objective function we want to minimize. Q_{pen} is an $m\times m$ QUBO constraint to respect, if any. W_i the linear inequality constraint i weights for each binary variable, and C_i the constant for linear inequality constraint i, if any. An example where W_i and C_i are inputs is when solving a combinatorial problem which has one or more linear constraints such as the binary knapsack problem where W_0 would represent the weight of each item and C_0 the capacity of the knapsack.

DA's objective is thus to find the vector x which gives the lowest value for $E(x)$ described in (4.1) which respects (4.2) and (4.3) if (4.2) and/or (4.3) are present.

In this chapter, we will focus on BQPs difficult enough so that their optimal solutions cannot be found within the set experiment DA run time T. Thus hyperparameters performance means the set of hyperparameters which give the lowest E found at T respecting the

constraints, which we will call E_{min} hereinafter. In this chapter, we do not discuss trade-off of hyperparameters leading to a better $E(x)$ within a smallest time $t_1 < T$ but have a worse $E(x)$ than others past t_2 where $t_1 < t_2 \leq T$.

We define GAP between E_{min} and another hyperparameter set p tried, leading to worse E_p as follows:

$$GAP = \frac{E_p - E_{min}}{|E_{min}|} \tag{4.4}$$

where E_p is the lowest E found for given hyperparameter set p. Thus, the best hyperparameter tuning method will lead to the best p with a GAP_p value of 0, which corresponds to E_{min}, and other methods will lead to worse p which will lead to GAP_p values strictly positive.

4.3 Black-box Hyperparameter Tuning Techniques

Black-box hyperparameter tuning techniques consist in picking a set of hyperparameters, input this set in an objective black-box function which should be minimized or maximized, record said objective value for the tried hyperparameter set and repeat the process until satisfied. In our case, the black-box is the DA, which is fed DA hyperparameters values and a given fixed BQP for which the hyperparameters are tuned for. What we measure are metrics related to finding the lowest $E(x)$ for a given BQP for a given hyperparameter values set. Metrics are described in Section. 4.3.3. This tuning flow is illustrated in Fig. 4.1.

The hyperparameter selection techniques (Step 1 in Fig. 4.1) we use as baselines for comparison are random sampling and TPE.

4.3.1 Baseline1: Random Sampling

Random sampling consists in randomly picking combination of hyperparameters within a given range r for each hyperparameter following a set distribution. r_{min} is the minimum value allowed for a given parameter and r_{max} its maximum. We thus have $r = r_{max} - r_{min}$ with $r_{max} > r_{min} \geq 0$. Every time a random combination of hyperparameters is chosen, it is evaluated using DA on the given BQP (Step 2) then the objective from DA solving metrics is computed (Step 3). Finally, the hyperparameter set and its corresponding objective value are recorded during Step 4. After doing n trials, the set of parameters p giving the lowest E_p is chosen as the best parameter set for the hyperparameter selection technique.

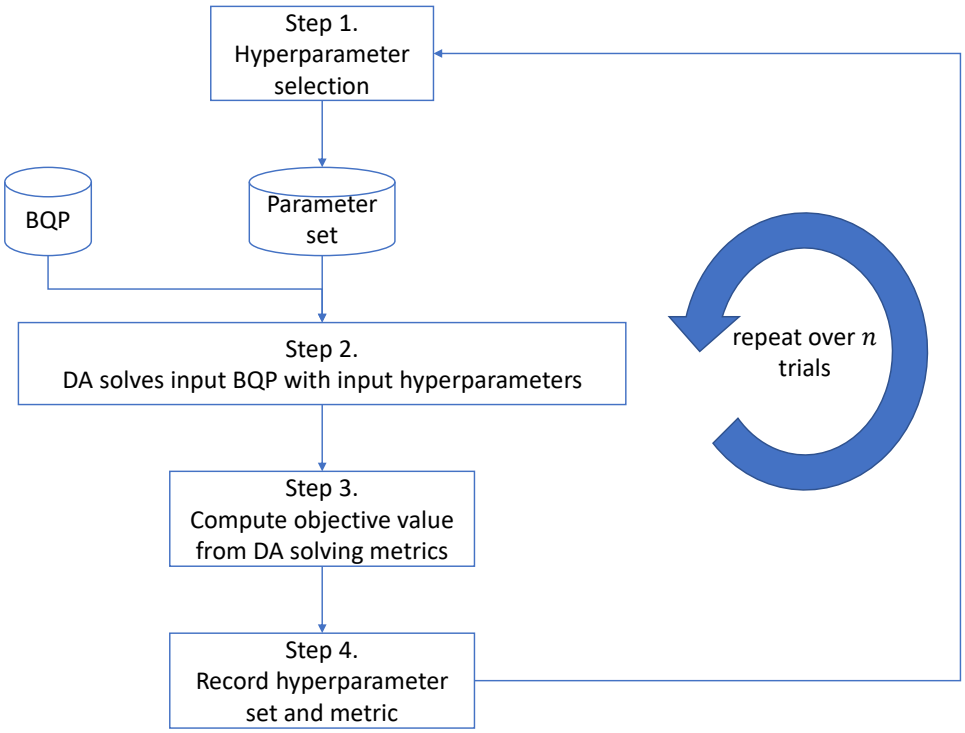

Fig. 4.1 DA Black-box tuning method

4.3.2 Baseline2: Tree-structured Parzen Estimator

Tree-structured Parzen Estimator (TPE) is a Bayesian method whose main difference with random sampling is that, in order to determine the next hyperparameter values to try (Step 1), it will consider past hyperparameter trials recorded during Step 4. Those past trials are used to constitute a surrogate model to the objective to minimize which is differentiable and thus easier to minimize than the original objective function. This surrogate model represents prior probability distributions $p(x|y)$ and $p(y)$ where y is the expected objective value given hyperparameter value x using tree-structured adaptive Parzen estimators. $p(x|y)$ is modeled using one Gaussian mixture model $l(x)$ to the set of hyperparameter values corresponding to the best objective values and another Gaussian mixture model $g(x)$ for the remaining parameter values. TPE selects the hyperparameter value x which maximizes the ratio $l(x)/g(x)$. TPE is used in hyperparameter optimization frameworks such as Hyperopt [10] and Optuna [2]. In this dissertation we use Optuna for our experiments.

4.3.3 Objective Computation

For objective computation (Step 3 in Fig. 4.1), we propose to calculate it as follows:

$$O(p)=t\times E_p+T_{E_p} \tag{4.5}$$

where $O(p)$ is the objective value for given hyperparameter set p, E_p is the lowest E found using p during the DA experiment run time T, T_{E_p} is the time at which solution corresponding to E_p was found, and t is a coefficient we set to T. The reason we do not simply record E_p is to differentiate two sets of parameters leading to the same E_p. By adding T_{E_p}, the hyperparameter set which leads to finding a solution corresponding to E_p the fastest is prioritized. Differentiating parameter set leading to same E_p is only useful for TPE based methods as it models the relation between hyperparameters and objective value to gradually converge to the best possible p. It is also only useful for BQP where E_p ties happen often.

4.4 Tree-structured Parzen Estimator acceleration for Ising Machines

Our main contribution is a “fast convergence" method to accelerate TPE when using TPE for optimizing hyperparameter values of Ising machines. We describe the method in Fig. 4.2.

The differences with the method described in Fig. 4.1 are twofold as we introduce:

- **Range narrowing**: a tuning warm-up phase which lasts m trials after which we update each hyperparameter range to explore (Step 5 and 6 in Fig. 4.2).
- **Convergence judgment**: if E_{min} was not updated for l number of trials (Step 7 in Fig. 4.2) after warm-up phase has been done, we terminate the tuning regardless of how many trials were left to be run in regard to set n.

4.4.1 Range Narrowing

In Step 6 in Fig. 4.2, for each parameter x we update its search range r by narrowing it and centering on current best parameter value. Thus $r_{new}=r_{old}/\gamma$ with r_{new} the new range of a given hyperparameter, r_{old} its previous range until m trials were completed and γ an introduced hyperparameter for our method. We note m is also a newly introduced hyperparameter representing how long the warm-up phase will be. The new min and max of each parameter range is centered on their bast values after m trials, they become $r_{new_{min}}=x_b-\frac{r_{new}}{2}$ and $r_{new_{max}}=x_b+\frac{r_{new}}{2}$ where x_b is the parameter value leading to E_{min} during

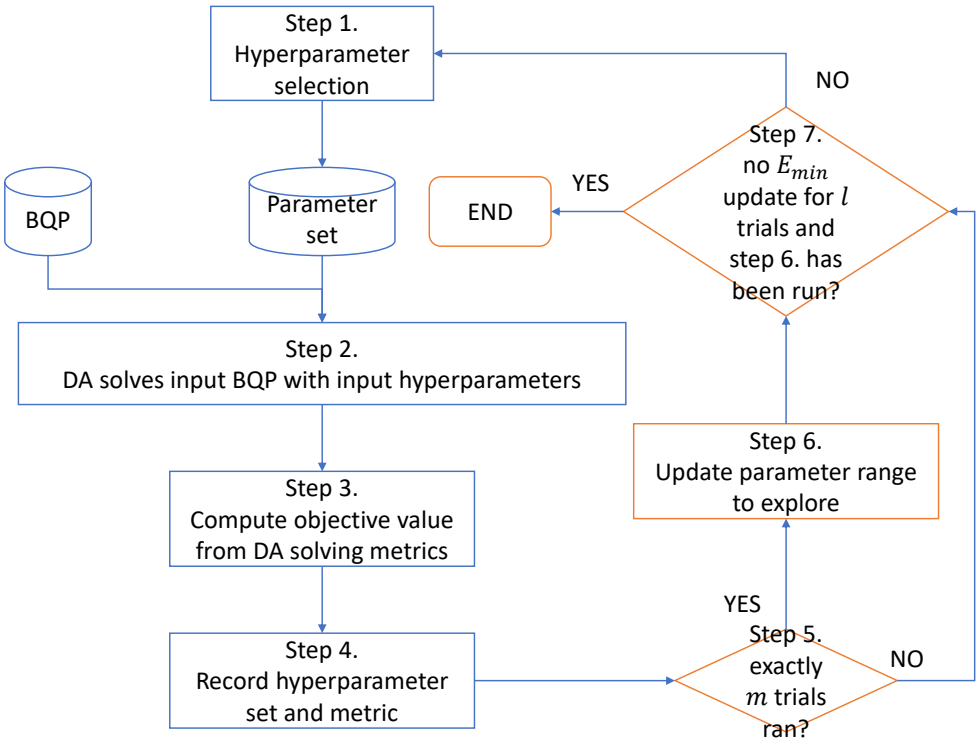

Fig. 4.2 DA fast-convergence tuning method

the warm-up phase. If $r_{new_{min}}$ would become lower than the lowest value allowable for that parameter, it is clipped to this lowest allowable value. If $r_{new_{max}}$ would become higher than the highest value allowable for that parameter, it is clipped to to this highest allowable value.

The intuition behind our concept of range narrowing is that hyperparameter range allowed by an Ising machine such as DA as its input can be significantly wide and the actual “effective" range can be much narrower. Thus, even when using intelligent method such as TPE, it can be challenging and our proposed range narrowing method can be effective although we introduce two new hyperparameters in our method: m and γ.

4.4.2 Convergence Judgment

Step 7 in Fig. 4.2 avoids doing hyperparameter trials which have a low probability of finding better hyperparameter values and thus lowering E_{min} when a significant number of trials have already been completed and E_{min} has not been updated for a set number of trials l. To implement this part, after warm-up phase is over, we simply count how many trials have been done without updating E_{min}. If more than l trials have been completed without updating E_{min}, the tuning is terminated.

4.4.3 Newly Introduced Hyperparameters

As we we have described in Section 4.4.1 and 4.4.2, we have introduced three new hyperparameters: l, m and γ. In this sub-section we discuss the trade-off of introducing new hyperparameters, with their related complexity, against expected benefit of finding better parameters, faster.

First, we make the claim that l virtually replaces n and is easier to tune. Our reason is, now instead of setting a fixed number of trials to run n, hoping it will be large enough to reach low enough E_{min} but not so large that a significant amount will be ran without any improvement, and thus wasting Ising machine time, we have l where the user consciously decides after how many trials without E_{min} updates he judges the hyperparameter tuning should end. The user should set l proportionally to the BQP difficulty he wants to solve, with a larger l value for more difficult problems. Thus, we think our proposed convergence judgment should not just benefit TPE but any black-box tuning method where a certain number of trials to execute has to be input.

For m, the length of our warm-up phase for hyperparameter range narrowing described in Section 4.4.1, like for l, we suggest to set it at a value proportional to BQP difficulty to solve. We also note it has to be inferior to n. We suggest to have $m{=}l$, as we used in Section 4.5.

For γ, we suggest to use a value of 4, as reducing the r to a fourth of its width worked well in our experiments in Seciton 4.5. A sidenote on BQP difficulty, BQP size can be used as a proxy as well as the number of linear inequalities, but other more sophisticated approaches can also be considered.

4.5 Experimental Results

As we stated in Section 4.2, we focus on BQP difficult enough so that we will never reach its optimal solution when we try to solve them using DA within run time T, no matter what parameter was used. We chose two well-known permutation problems Travelling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP). Within their publicly available benchmark, respectively “tsplib"[51] and “qaplib"[13], we chose 2 instances from each. First, we will describe TSP and QAP formulations as BQP.

4.5.1 Travelling

Salesman Problem as Binary Quadratic Program A well known formulation of Travel Salesman Problem as QUBO is described in [40], the cost of the QUBO is:

$$E(x)=\sum_{i=1}^{N}\sum_{j=1}^{N}D_{i,j}x_{i,t}x_{j,t+1} \tag{4.6}$$

where D is the TSP distance matrix, N is the number of nodes of the TSP, $x_{i,t}$ is the i visited node at time slot t, and $x_{j,t+1}$ is the j visited node at next time slot $t+1$. We thus have N binary variables per time slot which represent for each time slot, which node was visited, for a total of N^2 binary variables. The goal is to find the order in which to travel each node so that (4.6) is minimized. The coefficients between each binary variable constitute Q_{obj}, the objective input of the DA. The constraints of the TSP are that each node should be visited once and only once and that at each time slot, only one node should be visited. This can be expressed as:

$$H_{pen}(x)=\sum_{t=1}^{N}\left(1-\sum_{i=1}^{N}x_{i,t}\right)^2+\sum_{i=1}^{N}\left(1-\sum_{t=1}^{N}x_{i,t}\right)^2 \tag{4.7}$$

The coefficients between each binary variable in $H_{pen}(x)$ constitute Q_{pen}, the constraint QUBO input of the DA. If both constraints are satisfied, $H_{pen}(x)$ value will be 0.

4.5.2 Quadratic Assignment Problem as Binary Quadratic Program

A well-known formulation of QAP as a QUBO has been established in [26]. From it, we derive its formulation as BQP:

$$E(x)=\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N}F_{i,j}D_{k,l}x_{i,k}x_{j,l} \tag{4.8}$$

where $x_{i,k}$ is a binary variable which represents factory i assigned at location k, F is known as the flow matrix representing the amount of exchange between each factory, and D the distance matrix between each location. We thus have N binary variables per location which represent for each location, which factory should be assigned, for a total of N^2 binary variables. The goal is to assign each factories at locations such that (4.8) is minimized. The coefficients between each binary variable constitute Q_{obj}, the objective input of the DA.

The constraints are similar to TSP, each factory should be assigned once and only once, and at each location only one factory should be assigned:

$$H_{pen}(x)=\sum_{i=1}^{N}\left(1-\sum_{j=1}^{N}x_{i,j}\right)^2+\sum_{j=1}^{N}\left(1-\sum_{i=1}^{N}x_{i,j}\right)^2 \tag{4.9}$$

Likewise the coefficients between each binary variable in $H_{pen}(x)$ constitute Q_{pen}, the constraint QUBO input of the DA. If both constraints are satisfied, $H_{pen}(x)$ value will be 0.

4.5.3 Experimental Settings

We chose two difficult instances from tsplib and qaplib: respectively kroA100, gr120 for TSP and tai80a, tai100a for QAP. We run one experiment per problem instance. We describe the common settings between all experiment (and thus all problem instances BQP) below.

We used Optuna which allows to choose between both random and TPE sampler. Among the hyperparameters available for the DA, we chose to tune “gs_level", “gs_cutoff", “num_run" and “num_group" as they are the only parameters related to the search engine performance. Their description available in [60] is:

num_run: The number of parallel attempts of each group (int64 type). (num_run × num_group) specifies the number of parallel attempts.

num_group: The number of groups of parallel attempts (int64 type). (num_run × num_group) specifies the number of parallel attempts.

gs_level: Level of the global search (int64 type). In the global search, the search starting point with local solution group escape is determined, and the constrained search combining various search methods is repeatedly executed as a processing unit. The higher the value, the longer the constraint exploitation search. Specifies the level of the global search. Lower level is weak on Global Search.

gs_cutoff: Global search cutoff level (int64 type). Specifies the convergence judgment level for global search constraint usage search. The higher the value, the longer the period during which the constraint-based search energy on which convergence is based is not updated. Convergence assessment is turned off at 0.

We tuned the above parameters using their full parameter range, “gs_level": $[0,100]$, “gs_cutoff": $[0,10^6]$, “num_run": $[1,16]$, “num_group": $[1,16]$.

Those parameters are integers, thus the total search space is approximately 25.9×10^9 combinations of hyperparameter values. For each trial, we use a DA run time T of 30 seconds,

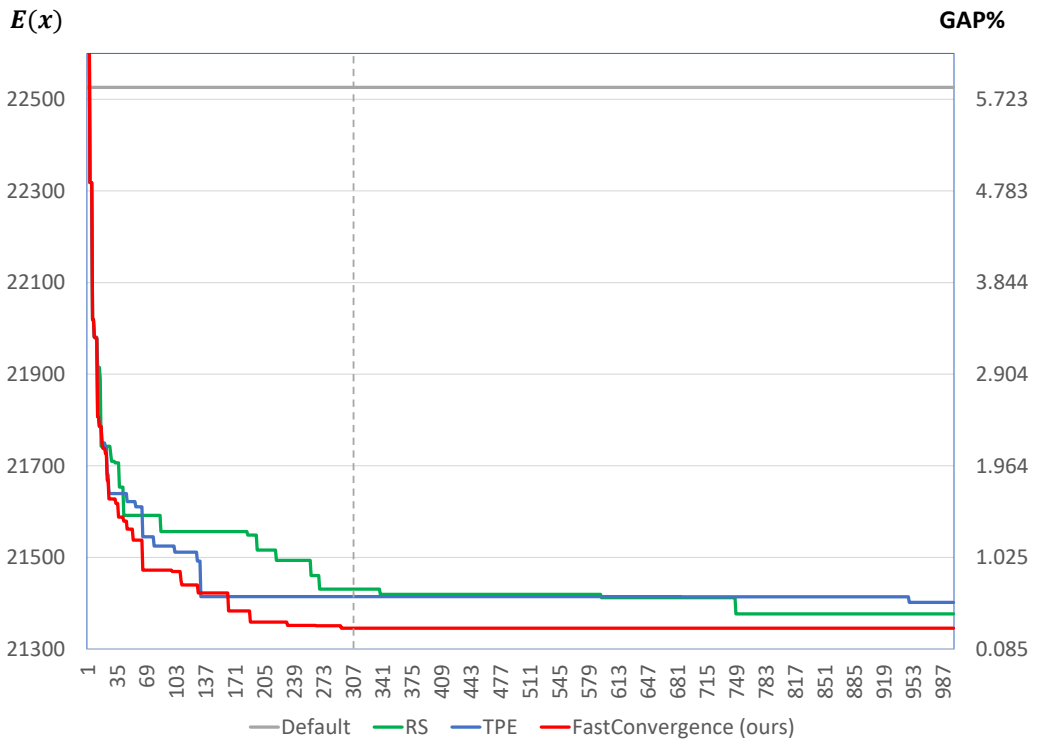

Fig. 4.3 TSP kroA100 Results

exploring the full space would require approximately twenty-five thousand years. We run n=1000 trials. We use γ=4 to reduce the range of parameters to a fourth of their original range after the warm-up phase is completed, as described in Section 4.4.1. We compare, after each trial, each baseline method and the proposed fast-convergence method average cost value of their best-found feasible solution, for their best-found hyperparameter set since the start of the experiment over 5 different Optuna random seeds. We plot them respectively on x and y axis in Fig. 4.3, 4.4, 4.5 and 4.6. In addition, we also plot as a horizontal line the cost obtained after T=30s using default hyperparameter values.

4.5.4 Results

For KroA100 instance, which is composed of 100 nodes, we use l=m=150 as settings for Step 5 and 7 of our FastConvergence method in Fig. 4.2. We then observe the results in Fig. 4.3. Gray line represents the cost value when using default hyperparameters values for T, green line Random Sampling, blue line TPE and red line the proposed FastConvergence method.

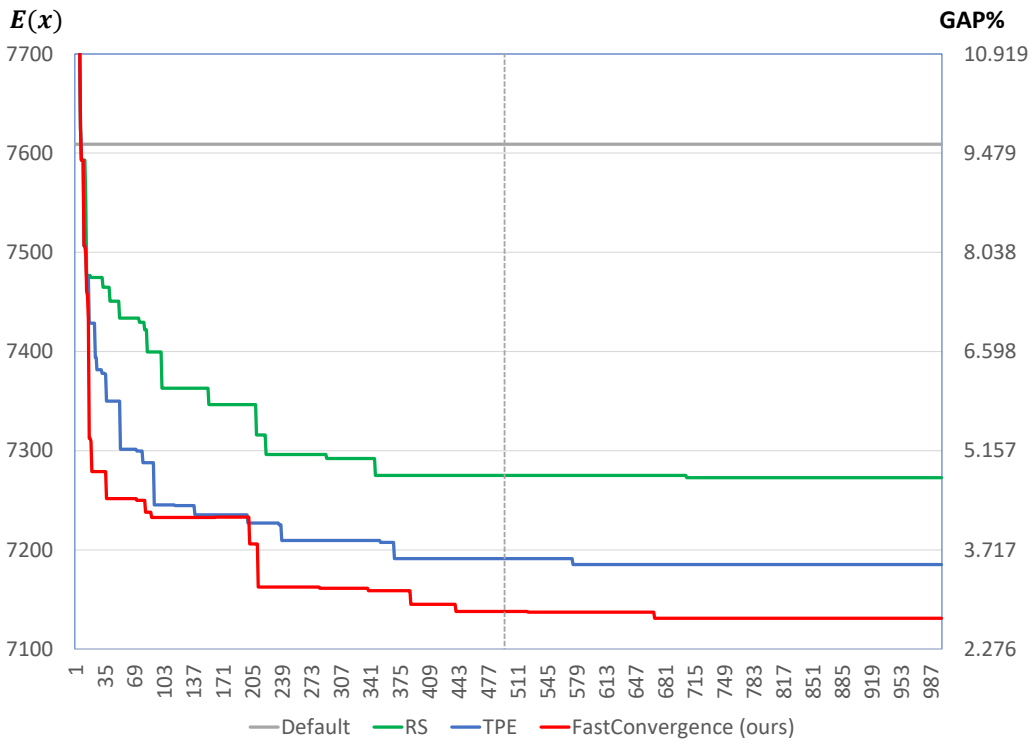

Fig. 4.4 TSP gr120 Results

We observe FastConvergence performs better than random sampling and TPE. It converges in average after 304 trials, referred to as "END" on Fig. 4.2 and is symbolized by the grey dashed line in Fig .4.3. Whereas for other methods, n=1000 trials are ran. At the 304 trials point, there is a GAP of 0.3% in favor of FastConvergence against TPE, where GAP means the relative difference in $E(x)$ between FastConvergence and TPE. This instance illustrates how if the total number of trials n to complete tuning was smaller than 150, performance would have been significantly worse and how going beyond 150 trials yields only marginal improvements. Thus, FastConvergence yields top performing parameters for approximately a third of the tuning time of standalone TPE and random sampling.

For TSP instance gr120, which is composed of 120 nodes, we use FastConvergence settings m=l=200 and observe the results in Fig. 4.4. We setup a higher value for l and m since the number of TSP nodes is larger. In this instance, compared to kroA100, random sampling perform significantly worse than the other two methods. We note FastConvergence converges after 496 trials in average, at which point the GAP with TPE is of 0.7% in favor FastConvergence which is larger than for kroA100.

For QAP instance tai80a, which is composed of 80 factories and locations, we use FastConvergence settings m=l=200 and observe the results in Fig. 4.5. Although the number

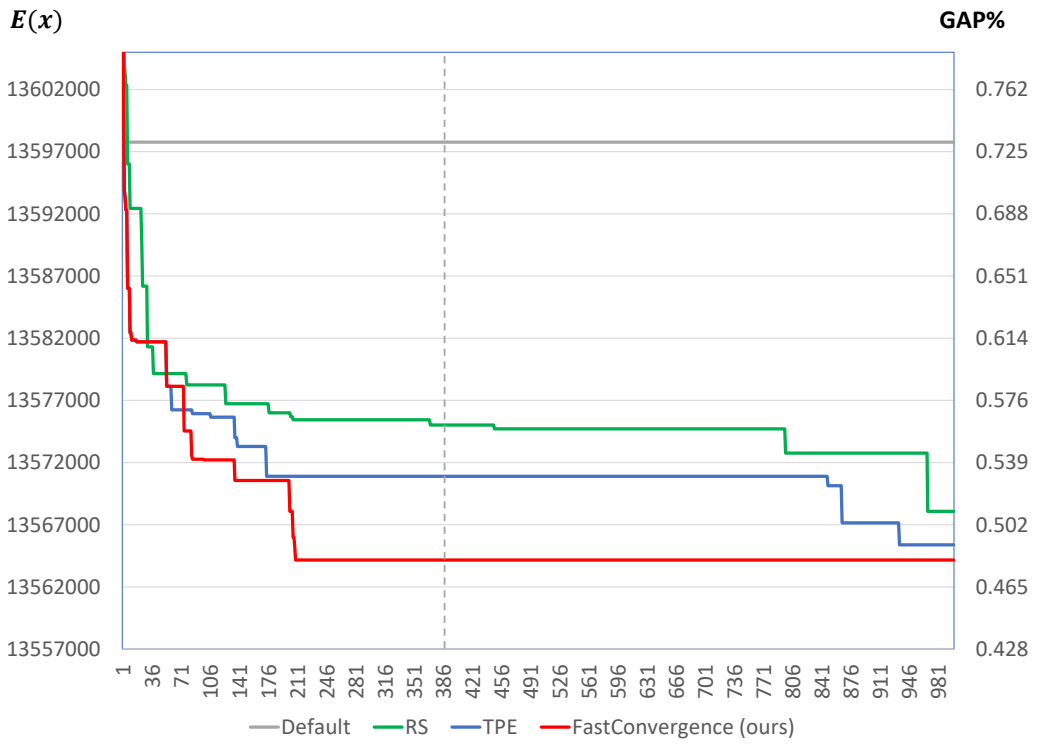

Fig. 4.5 QAP tai80a Results

of variables is smaller than for gr120, we setup a similar value to gr120 for l and m since QAP is a more complex problem than TSP. We still have FastConvergence, TPE, random sampling in order of most well performing tuning technique as for TSP. FastConvergence converges in average after 388 trials at which point the GAP with TPE is 0.05%.

For QAP instance tai100a, which is composed of 100 factories and locations, we use FastConvergence settings m=l=250 and observe the results in Fig. 4.6. FastConvergence converges after 422 trials in average at which point the GAP with TPE is 0.01% and is the best performing method overall.

Across all tested instances, FastConvergence was able to find better parameters than TPE alone, within between 304 and 496 trials. Compared to the n=1000 trials, it represents approximately between a third and a half of the learning time to obtain similar performing hyperparameter values.

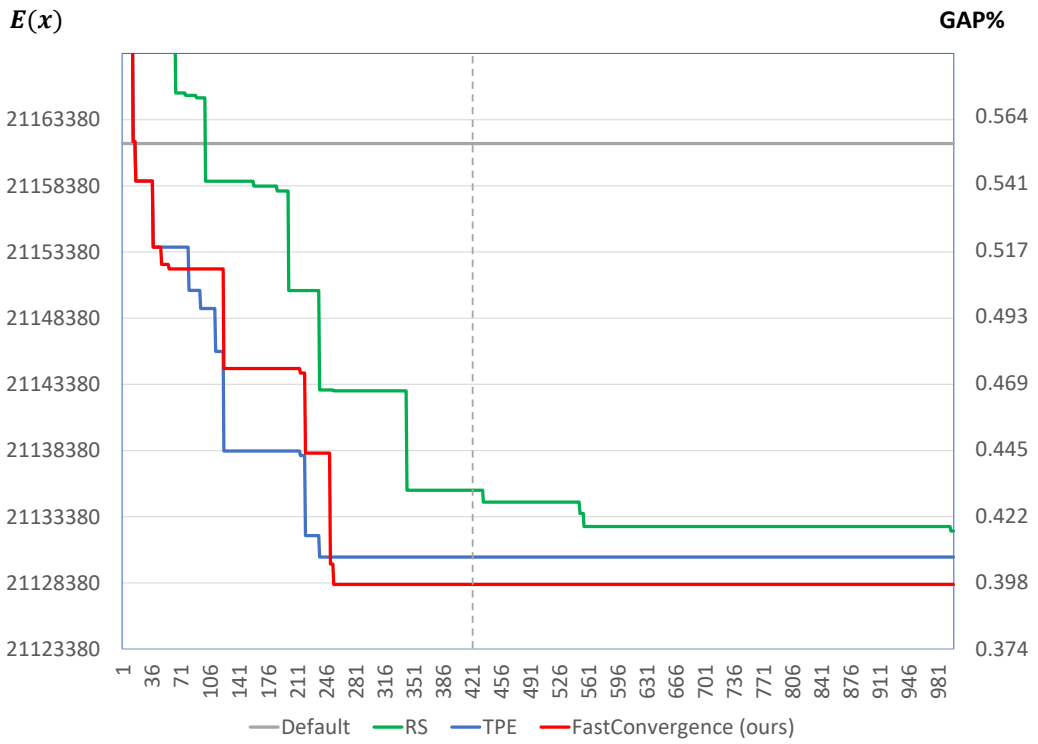

Fig. 4.6 QAP tai100a Results

4.6 Conclusion

In this chapter, we showed popular black-box tuning techniques such as random sampling or tree-structured Parzen estimator can be used effectively to tune Ising machines. We proposed to improve on the state of the art, TPE, to allow it to converge faster while still being able to obtain similar parameter quality. The proposed FastConvergence method as well as the objective we defined for TPE are the first steppingstones in our hyperparameter tuning framework for Ising machines.

In our futures works, we would like to try FastConvergence on other problem categories and Ising machines to demonstrate how general it is. We also would like to study how hyperparameter values in general can be re-used across different problem instance and how we could do that efficiently to accelerate tuning time.

Chapter 5

Conclusions

This dissertation goal was to maximize Ising Machines' combinatorial problem-solving performance. To that end, it proposed a visualization method to find bottlenecks in QUBO modeling and illustrated how to solve bottlenecks by using a solution mending algorithm and a different integer to binary variables encoding. It also proposed an efficient Ising machine hyperparameter tuning framework which help improve performance regardless of the type of QUBO input. Overall, this dissertation demonstrated what are the most important aspects to consider to maximize Ising machine performance, and that the proposed methods are efficient to this end. In this regard, this dissertation has successfully achieved its goal.

Chapter 2 [Analysis and Acceleration of the Quadratic Knapsack Problem on an Ising Machine] proposed a QUBO search space landscape visualization technique which uses two local minima solutions found to represent the multi-dimensional space in between and understand what makes a QUBO hard to solve. This technique was applied to the Quadratic Knapsack Problem (QKP) where the goal was to maximize the value of items inserted in a knapsack. Chapter 2 showed that moving in and out heavy items from the knapsack is difficult due to the nature of representing linear inequality constraints as QUBO. With this insight, a solution mending method was proposed to help Ising machines stuck in local minima which raises the chances of finding the optimal solution with an Ising machine from only 6.7% to 60.7% for an Ising machine used with the proposed solution mending method. Chapter 2 also compared proposed Ising machine used with solution mending against SA. Results showed that SA never reaches optimal solution for all problem instances evaluated on.

Chapter 3 [Cardinality Constrained Portfolio Optimization on an Ising Machine] proposed a novel integer variable to binary variables modeling technique. Proposed landscape technique from Chapter 2 helped us to identify another bottleneck when solving problems which include integer variables. Classic encoding methods either create large differences in the cost function when changing a binary variable value representing a large integer

value, which makes those moves unlikely to happen, or require a large quantity of binary variables, creating both computing complexity and a larger memory footprint. A “base10” encoding was proposed as a tradeoff between having binary variables representing large integer values and having a high number of binary variables. Proposed encoding was applied to the Cardinality Constrained Mean Variance Portfolio Optimization Problem (CCMVPOP), an NP-hard problem which uses real number variables. An efficient QUBO model for the CCMVPOP was thus proposed by first converting real number variables to integer variables using coefficient multiplication with rounding combined with integer to binary variable encoding. Solving performance using the proposed encoding was compared to classic encodings. Results showed the time to the best-known solutions can be improved by a factor of up to 10x for several CCMVPOP instances. Results also showed the proposed encoding is the only one which allows to reach the best-known solutions for a large CCMVPOP instance.

Chapter 4 [Fast Hyperparameter Tuning for Ising Machines] proposed a novel Ising machine hyperparameter tuning framework. It is based on machine learning state of the art hyperparameter tuning method called Tree-structured Parzen Estimator (TPE), which is a kind of Bayesian optimization technique. After showing TPE effectiveness extends to Ising machines, as well comparing TPE to random parameter sampling, it proposed an enhanced TPE called "FastConvergence". "FastConvergence" reduces the time required to find parameters which enable the same level of performance as TPE. Random sampling, TPE, and "FastConvergence" are compared using DA to solve Travel Salesman Problem (TSP) and Quadratic Assignment Problem (QAP), two well-known NP-hard problems often used for Ising machine benchmarking. Results showed that the proposed “FastConvergence” can find parameters which give solving performance equivalent or better than TPE with two to three times faster tuning time.

In conclusion, several methods to improve Ising machines’ performance have been proposed. However, several tasks remain to achieve peak performance. Our future works are as follows:

- Further improve our hyperparameter tuning framework to reduce tuning time.
- Continue to develop QUBO solving visualization techniques to identify bottlenecks, to give hints for future QUBO modeling techniques and Ising machines architecture improvements.
- Perform in-depth benchmarking to compare Ising machines peak performance with other solvers, which is a complex task as, depending on the considered problem, reference solving techniques can vary greatly.

- Combine bottleneck analysis, hyperparameter tuning and benchmarking into an automated evaluation system to accelerate on Ising machines.

First, since current hyperparameter tuning is still a long process, we aim at reducing the necessary tuning time further, so that tuning could be an integrated part in the solving process. Second, we will develop more QUBO visualization techniques to quickly understand when an Ising machine is under-performing. Third, we will do more benchmarking to understand in which applications Ising machines are viable to replace other solving technologies. Finally, we will develop a system to automate hyperparameter tuning, benchmarking and bottleneck analysis to accelerate experiment turnover time and allow researchers to focus on the most promising applications for Ising machines.

Acknowledgements

First and foremost, I would like to express my deepest thanks and my gratitude to Prof.Nozomu Togawa (戸川望教授) at the Department of Computer Science and Communications Engineering of Waseda University for his support and valuable expertise given to me for three years.

Secondly, I appreciate the valuable advices of Prof. Keiji Kimura (木村啓二教授) at the Department of Computer Science and Communications Engineering of Waseda University and Prof. Shinji Kimura (木村晋二教授) at the Graduate School of Information, Production and Systems of Waseda University.

I would like to offer my special thanks to Mr. Keisuke Fukada (深田佳佑氏) for working together on this project. I also thank all the students in the Togawa Laboratory for their support.

Finally I thank again all my family, my friends and my colleagues. Without their support, this work would not have been possible. My manager, Dr. Kazuya Takemoto (竹本一矢氏), for being an inspirational leader and supporter in this endeavor. My colleague, Dr. Norihiro Kakuko (覚幸典弘氏), for his invaluable technical contribution on chapter 4. My former team members, Mr. Hidetoshi Matsuoka (松岡英俊氏), Mr. Hiroshi Ikeda (池田弘氏) and Mr. Tatsuya Yamamoto (山本達也氏), for always mentoring me, pushing me and motivating me. My friend and colleague Mr. Amir Haderbache, for all the great discussions and the time we enjoy together daily.

References

[1] A quadratic unconstrained binary optimization problem formulation for single-period index tracking with cardinality constraints (2021). A quadratic unconstrained binary optimization problem formulation for single-period index tracking with cardinality constraints. https://qcware.com/wp-content/uploads/2019/09/index-tracking-white-paper.pdf. Accessed: 2021-08-18.

[2] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework.

[3] Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., and Katzgraber, H. G. (2019). Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7:48.

[4] Ayodele, M. (2022). Penalty weights in qubo formulations: Permutation problems. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 159–174. Springer.

[5] Babaeian, M., Nguyen, D. T., Demir, V., Akbulut, M., Blanche, P.-A., Kaneda, Y., Guha, S., Neifeld, M. A., and Peyghambarian, N. (2019). A single shot coherent ising machine based on a network of injection-locked multicore fiber lasers. *Nature Communications*, 10(1):3516.

[6] Barahona, F. (1982). On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241.

[7] Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

[8] Beasley, J. E. (1996). Obtaining test problems via internet. *Journal of Global Optimization*, 8(4):429–433.

[9] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

[10] Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org.

[11] Billionnet, A. and Calmels, F. (1996). Linear programming for the 0–1 quadratic knapsack problem. *European Journal of Operational Research*, 92(2):310 – 325.

[12] Bretzke, W.-R. and Barkawi, K. (2012). *Sustainable logistics: responses to a global challenge*. Springer Science & Business Media.

[13] Burkard, R. E., Karisch, S. E., and Rendl, F. (1997). Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403.

[14] Caprara, A., Pisinger, D., and Toth, P. (1999). Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137.

[15] Chang, T.-J., Meade, N., Beasley, J., and Sharaiha, Y. (2000). Heuristics for cardinality constrained portfolio optimisation. *Computers and Operations Research*, 27(13):1271–1302.

[16] Chen, Y. and Hao, J. (2016). Memetic search for the generalized quadratic multiple knapsack problem. *IEEE Transactions on Evolutionary Computation*, 20(6):908–923.

[17] Chen, Y. and Hao, J.-K. (2015). Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1):101–131.

[18] D-Wave Simulated Annealing Sampler (2021). D-wave simulated annealing sampler. https://docs.ocean.dwavesys.com/projects/neal. Accessed: 2021-03-10.

[19] DA Portal (2022). Fujitsu Quantum-inspired Computing Digital Annealer Portal. https://www.fujitsu.com/global/services/business-services/digital-annealer/. Accessed: 2022-01-31.

[20] Diez García, M., Ayodele, M., and Moraglio, A. (2022). Exact and sequential penalty weights in quadratic unconstrained binary optimisation with a digital annealer. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, New York, NY, USA. Association for Computing Machinery.

[21] Fernández-Lorenzo, S., Porras, D., and García-Ripoll, J. J. (2021). Hybrid quantum–classical optimization with cardinality constraints and applications to finance. *Quantum Science and Technology*, 6(3):034010.

[22] Ferreira, C. E., Martin, A., de Souza, C. C., Weismantel, R., and Wolsey, L. A. (1996). Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, 74(3):247–266.

[23] Gallo, G., Hammer, P. L., and Simeone, B. (1980). *Quadratic knapsack problems*, pages 132–149. Springer Berlin Heidelberg, Berlin, Heidelberg.

[24] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549. Applications of Integer Programming.

[25] Glover, F., Kochenberger, G., and Alidaee, B. (2002). Solving quadratic knapsack problems by reformulation and tabu search: Single constraint case. *Combinatorial and Global Optimization*, 14.

[26] Glover, F., Kochenberger, G., and Du, Y. (2018). A tutorial on formulating and using qubo models.

[27] Goh, S. T., Gopalakrishnan, S., Bo, J., and Lau, H. C. (2020). A hybrid framework using a qubo solver for permutation-based combinatorial optimization.

[28] Goto, H., Tatsumura, K., and Dixon, A. (2019). Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. *Science Advances*, 5:eaav2372.

[29] Graham, D. I. and Craven, M. J. (2021). An exact algorithm for small-cardinality constrained portfolio optimisation. *Journal of the Operational Research Society*, 72(6):1415–1431.

[30] Hamerly, R., Inagaki, T., McMahon, P. L., Venturelli, D., Marandi, A., Onodera, T., Ng, E., Langrock, C., Inaba, K., Honjo, T., Enbutsu, K., Umeki, T., Kasahara, R., Utsunomiya, S., Kako, S., Kawarabayashi, K.-i., Byer, R. L., Fejer, M. M., Mabuchi, H., Englund, D., Rieffel, E., Takesue, H., and Yamamoto, Y. (2019). Experimental investigation of performance differences between coherent ising machines and a quantum annealer. *Science Advances*, 5(5).

[31] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[32] Hen, I. (2019). Equation planting: a tool for benchmarking ising machines. *Physical Review Applied*, 12(1):011003.

[33] Huang, T., Goh, S. T., Gopalakrishnan, S., Luo, T., Li, Q., and Lau, H. C. (2021). Qross: Qubo relaxation parameter optimisation via learning solver surrogates. In *2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 35–40.

[34] Hukushima, K. and Nemoto, K. (1996). Exchange monte carlo method and application to spin glass simulations. *Journal of the Physical Society of Japan*, 65(6):1604–1608.

[35] Johnson, E. L., Mehrotra, A., and Nemhauser, G. L. (1993). Min-cut clustering. *Mathematical programming*, 62(1-3):133–151.

[36] Johnson, M. W., Amin, M. H. S., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., Chapple, E. M., Enderud, C., Hilton, J. P., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M. C., Tolkacheva, E., Truncik, C. J. S., Uchaikin, S., Wang, J., Wilson, B., and Rose, G. (2011). Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198.

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

[38] Komiyama, T. and Suzuki, T. (2021). Sparse matrix ordering method with a quantum annealing approach and its parameter tuning. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 258–264.

[39] Laughhunn, D. (1970). Quadratic binary programming with application to capital-budgeting problems. *Operations research*, 18(3):454–461.

[40] Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2.

[41] Markowitz, H. M. (1968). *Portfolio selection*. Yale university press.

[42] Matsubara, S., Takatsu, M., Miyazawa, T., Shibasaki, T., Watanabe, Y., Takemoto, K., and Tamura, H. (2020). Digital annealer for high-speed solving of combinatorial optimization problems and its applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 667–672.

[43] McGeoch, C. C. (2014). Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5(2):1–93.

[44] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092.

[45] Montreuil, B. (2011). Toward a physical internet: meeting the global logistics sustainability grand challenge. *Logistics Research*, 3(2):71–87.

[46] Nakayama, H., Koyama, J., Yoneoka, N., and Miyazawa, T. (2021). Third Generation Digital Annealer Technology. White paper, Fujitsu LTD. Accessed: 2022-05-05.

[47] Padberg, M. and Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100.

[48] Parizy, M. and Togawa, N. (2021). Analysis and acceleration of the quadratic knapsack problem on an ising machine. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E104.A(11):1526–1535.

[49] Patvardhan, C., Bansal, S., and Srivastav, A. (2015). Solving the 0–1 quadratic knapsack problem with a competitive quantum inspired evolutionary algorithm. *Journal of Computational and Applied Mathematics*, 285:86 – 99.

[50] Pisinger, D. (2007). The quadratic knapsack problem—a survey. *Discrete Applied Mathematics*, 155(5):623 – 648.

[51] Reinelt, G. (1991). TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.

[52] Rhys, J. M. (1970). A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207.

[53] Rosenberg, G., Haghnegahdar, P., Goddard, P., Carr, P., Wu, K., and de Prado, M. L. (2016). Solving the optimal trading trajectory problem using a quantum annealer. *IEEE Journal of Selected Topics in Signal Processing*, 10(6):1053–1060.

[54] Shaw, D. X., Liu, S., and Kopman, L. (2008). Lagrangian relaxation procedure for cardinality-constrained portfolio optimization. *Optimization Methods and Software*, 23(3):411–420.

[55] Shirai, T. and Togawa, N. (2022). Multi-spin-flip engineering in an ising machine. *IEEE Transactions on Computers*.

[56] Suen, W. Y., Yat Lee, C., and Lau, H. C. (2021). Quantum-inspired algorithm for vehicle sharing problem. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 17–23.

[57] Takabatake, K., Yanagisawa, K., and Akiyama, Y. (2022). Solving generalized polyomino puzzles using the ising model. *Entropy*, 24(3).

[58] Tamura, K., Shirai, T., Katsura, H., Tanaka, S., and Togawa, N. (2021a). Performance comparison of typical binary-integer encodings in an ising machine. *IEEE Access*, 9:81032–81039.

[59] Tamura, K., Shirai, T., Katsura, H., Tanaka, S., and Togawa, N. (2021b). Performance comparison of typical binary-integer encodings in an ising machine. *IEEE Access*, 9:81032–81039.

[60] Third Generation Digital Annealer API (2022). Digital Annealer API Reference (QUBO API V3). https://portal.aispf.global.fujitsu.com/apidoc/da/jp/api-ref/da-qubo-v3-en.html/. Accessed: 2022-01-23.

[61] Verma, A. and Lewis, M. (2020). Penalty and partitioning techniques to improve performance of qubo solvers. *Discrete Optimization*, page 100594.

[62] Yamaoka, M., Yoshimura, C., Hayashi, M., Okuyama, T., Aoki, H., and Mizuno, H. (2015). A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing. *IEEE Journal of Solid-State Circuits*, 51(1):303–309.

List of Publications

Peer reviewed journal articles

⟨1⟩ ○ **M. Parizy**, and N. Togawa, "Analysis and Acceleration of the Quadratic Knapsack Problem on an Ising Machine," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, vol. E104.A, no. 11, pp. 1526-1535, Dec. 2021.

⟨2⟩ K.Fukada, M. Parizy, and N. Togawa, "A Three-Stage Annealing Method Solving Slot-Placement Problems Using an Ising Machine," *IEEE Access*, vol. 9, pp. 134413-134426, 2021.

Peer reviewed conference papers

⟨3⟩ ○ **M. Parizy**, N. Kakuko, and N. Togawa, "Fast Hyperparameter Tuning for Ising Machines," in *2023 IEEE International Conference on Consumer Electronics (ICCE) (2023 ICCE)*, Las Vegas, Nevada, Jan. 2023.

⟨4⟩ ○ **M. Parizy**, P. Sadowski, and N. Togawa, "Cardinality Constrained Portfolio Optimization on an Ising Machine," in *2022 IEEE 35th International System-on-Chip Conference (SOCC) (SOCC 2022)*, Belfast, United Kingdom (Great Britain), Sep. 2022.

⟨5⟩ W.Y. Suen, M. Parizy, and H.C. Lau, "Enhancing a QUBO solver via data driven multi-start and its application to vehicle routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22)*, pp. 2251—2257, Boston, Massachusetts, Jul. 2022.

⟨6⟩ M. Ayodele, R. Allmendinger, M. López-Ibáñez, and M. Parizy, "Multi-objective QUBO solver: bi-objective quadratic assignment problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22)*, pp. 467—475, Boston, Massachusetts, Jul. 2022.

Non-peer reviewed conference participation

⟨7⟩ **M. Parizy**, and N. Togawa, “Analysis and Acceleration of Combinatorial Optimization Problems Including Inequality Constraints on Ising Machines,” in *Adiabatic Quantum Computing Conference 2021 (AQC 2021)*, Tokyo, Japan, Jun. 2021.

国内学会

⟨8⟩ 深田佳佑, パリジマチュー, 富田憲範, 戸川望, “さまざまなイジング計算機による組合せ最適化問題の解法と比較,” 情報処理学会　量子ソフトウェア（QS）, vol. 2022-QS-5, no. 3, pp. 1–12, オンライン, Mar. 2022.

招待論文

⟨9⟩ 宮澤俊之，小山純平，Matthieu Parizy, “第三世代デジタルアニーラ—ハイブリッドソルバ技術とその性能—,” 日本オペレーションズ・リサーチ学会　機関誌　特集イジングマシンとOR, vol. 67, no. 6, pp. 312–319, Jun. 2022.

特許

⟨10⟩ 覚幸　典弘，パリジ　マチュー, “情報処理装置、情報処理方法およびプログラム,”　出願番号2021-192401

⟨11⟩ 覚幸　典弘，パリジ　マチュー, “情報処理方法、情報処理プログラム、および情報処理装置,”　出願番号2022-106092