Exact and Sequential Penalty Weights in Quadratic Unconstrained Binary Optimisation with a Digital Annealer

Preprint · May 2022

DOI: 10.1145/3520304.3528925

CITATIONS
6

READS
518

3 authors:

Marcos Diez García
Fujitsu Ltd.
12 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Mayowa Ayodele
Fujitsu Research of Europe
23 PUBLICATIONS **109** CITATIONS
SEE PROFILE

Alberto Moraglio
University of Exeter
92 PUBLICATIONS **1,745** CITATIONS
SEE PROFILE

Exact and Sequential Penalty Weights in Quadratic Unconstrained Binary Optimisation with a Digital Annealer

Marcos Diez García
University of Exeter
Exeter, United Kingdom
md518@exeter.ac.uk

Mayowa Ayodele
Fujitsu Research of Europe Ltd.
Slough, United Kingdom
mayowa.ayodele@fujitsu.com

Alberto Moraglio
University of Exeter
Exeter, United Kingdom
a.moraglio@exeter.ac.uk

ABSTRACT

Quadratic unconstrained binary optimisation (QUBO) problems is a general class of combinatorial optimisation problems, which regained popularity after recent advances in quantum computing. Quantum-inspired hardware like Fujitsu's Digital Annealer (DA), based on simulated annealing, can solve QUBO problems much faster than traditional computers. Penalty methods can convert constrained optimisation problems into QUBO problems. But some existing methods of adjusting penalty weights are limited to specific QUBO problems. Others become computationally prohibitive for large problem sizes, or require manual analysis by experts in QUBO. We address these issues. We present exact penalty methods that are efficient, applicable to *any* QUBO, and compute simple upper-bounds of the smallest penalty weight *guaranteeing* that unconstrained global optima are the same as the feasible global optima. These bounds can be iteratively improved by sequential penalty methods which we also present. Experimental results with the DA on minimum cut, travelling salesman and multi-dimensional knapsack problems show the viability of the novel methodology hybridising exact and sequential methods. This work contributes towards general, automatic and scalable penalty methods in QUBO.

CCS CONCEPTS

• **Mathematics of computing → Optimization with randomized search heuristics**; **Combinatorial optimization**.

KEYWORDS

Quadratic unconstrained binary optimisation, constraint handling, penalty weight, simulated annealing, Digital Annealer

ACM Reference Format:
Marcos Diez García, Mayowa Ayodele, and Alberto Moraglio. 2022. Exact and Sequential Penalty Weights in Quadratic Unconstrained Binary Optimisation with a Digital Annealer. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3520304.3528925

1 INTRODUCTION

Quadratic unconstrained binary optimisation (QUBO) is a widely used mathematical framework to formulate and solve unconstrained versions of many constrained combinatorial optimisation (CCO) problems [19], including all of Karp's NP-complete problems [17]. There is a growing interest in QUBO for its diverse range of applications: from detecting community structures in chemical or social networks [24], to tackling complex multi-period portfolio optimisation problems [27], or improving integer factorisation to break RSA cryptography [25], among others [14]. QUBO or its equivalent Ising spin-glass model [29, 31] also became the basis to develop specialised computer hardware, such as D-Wave's quantum annealer [22] and Fujitsu's Digital Annealer (DA) [21], towards more efficient and large-scale problem solving.

CCO problems can be reformulated as QUBO problems via penalty methods [5, 14, 19]. The underlying mechanism of penalty methods is to cast the problem's constraints as a non-negative term that is added to the objective function to increase the cost of any infeasible solution (i.e. penalise them). Thus cost and constraint violation can be minimised at once by searching for the global minima of the (now unconstrained) objective function. However, penalty methods pose a major challenge [5, 32]: it is not always clear how penalty terms should be weighted. Penalty weights impact how solutions of the unconstrained problem approximate those of the original constrained problem. Valid penalty weights should not be so small as to render infeasible the globally optimal solutions yet not so large as to harm search performance due to large jumps in fitness between feasible and infeasible solutions. The following are some of the most typically used penalty methods.

Exact penalty methods [8, 15] model a constrained optimisation problem as an unconstrained problem, where the unconstrained minima are *guaranteed* to be *exactly* the solutions of the original constrained problem. This is achieved by mathematical analysis of the original problem's objective and constraints. The solutions may even be provably characterised under customary assumptions of continuity and differentiability in non-linear programming [10]. This paper focuses on combinatorial optimisation problems where those assumptions are not available.

Penalty weights should be kept small while guaranteeing the unconstrained global minima are the same as the feasible global minima in the original CCO problem. One approach to find such weights is to derive a theoretical upper-bound of them by analysing properties of the problem class, the objective or penalty functions. Lucas [19] demonstrates this approach for Ising formulations of Karp's NP-complete problems [17]. The derived weights may not be smallest even if they preserve the feasible optima of the CCO problem. However, this is a manual and tedious approach that relies on the mathematical structure of each specific problem to obtain tight penalty weight upper-bounds. Thus limiting the applicability of the bounds down to a single problem class or instance. In fact, tightness of penalty weight upper-bounds can be a decisive performance factor as demonstrated, for example, by the EXPEDIS algorithm against state-of-the-art solvers [15] in linearly constrained quadratic optimisation problems over binary variables. Furthermore, such case-by-case analyses are often accessible only to experts in QUBO, and even so it becomes extraordinarily challenging for problems with complicated representations and constraints like capacitated vehicle routing problems [9].

Unlike exact penalty methods, the classical sequential penalty methods (or discrete variants [30]) model a constrained optimisation problem as a *sequence* of unconstrained minimisation problems [10]. That is, solving the problem starting with a very small penalty weight; then, gradually increasing the weight and solving the problem with the new weights until a *feasible* (near) optimum is obtained, which is a solution of the original constrained problem. In practice sequential penalty methods have been computationally limited to small problem sizes (often less than a thousand variables), and their success is sensitive to the starting penalty weight as well as the rate at which it gradually increases. Moving away from classic sequential methods made possible to tackle large-scale CCO problems reformulated as QUBO by means of fast greedy heuristics [13, 16], evolutionary algorithms [5] and other meta-heuristics like simulated annealing or tabu search [12, 14]. Yet penalty weights are often manually chosen ad hoc per problem instance or by trial and error without provably guaranteeing their validity [9, 24, 27].

Overall, correctly setting penalty weights raises two challenges. First, developing a general, automatic, way to find valid penalty weights which preserve feasible global optima and have relatively small magnitude. Second, understanding more deeply how different penalty weights and types of penalty methods can affect search performance in terms of solution quality, feasibility, or runtime.

This paper addresses the above challenges by developing a novel framework for a general, automatic and efficient penalty method that can find upper-bounds of the smallest valid penalty weight given a CCO problem. Thus a framework that frees us from manually guessing penalty weights, scales well as problem size grows, and that is not limited to only one class of CCO problems nor instance. Our framework is based on exact penalty methods whereby such penalty weight upper-bounds can be efficiently derived using general, well-known, theoretical bounds on QUBO formulas. Since these bounds are general, the penalty weight upper-bounds may be loose. To improve on them, we also design new sequential penalty methods *informed* by such upper-bounds. We hypothesise this hybrid methodology is superior to the exact methods and standard sequential methods considered separately. Our experiments corroborate the hypothesis by comparing the performance of all these penalty methods in terms of penalty weight magnitude, solution quality and runtime using Fujitsu's third-generation DA [23] as solver. For benchmarking we choose instances of minimum cut, travelling salesman and multi-dimensional knapsack problems available from well-known test suites [2, 7, 26].

2 PRELIMINARIES

QUBO problems consist in minimising an unconstrained quadratic pseudo-Boolean function of $n \in \mathbb{N}$ binary variables $x_1, \dots, x_n$, which can always be expressed in the following polynomial form [3]:

$$q(x) = c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \leq i < j \leq n} c_{ij} x_i x_j \ , \qquad (1)$$

with fixed scalar coefficients c_0, c_i and $c_{i,j}$. We may call $q(x)$ the cost, energy or fitness of a candidate solution x. All penalty methods presented later consider unconstrained quadratic pseudo-Boolean functions $f(x) + w \cdot g(x)$, where w is a non-negative scalar, f is an objective function and g a penalty function associated with a given CCO problem. For simplicity, the penalty functions used in our selected benchmark problems are taken from the literature (see Section 5). So we do not address converting constraints into penalty functions, as we focus on adjusting penalty weights. We assume: (a) there always exists at least one feasible global optimum; (b) f and g can always be expressed in the quadratic polynomial form of Equation (1); (c) $g(x) = 0$ if x is feasible and $g(x) \geq 1$ whenever x is infeasible; and, (d) all constraints of a given CCO problem can be aggregated into g so being equally penalised by the same weight w.

2.1 Overview of the Digital Annealer

DA is a dedicated processor, introduced by Fujitsu in 2017, that is inspired in adiabatic quantum computation and uses massive parallelism to solve QUBO problems more efficiently [20, 28]. The scale of QUBO problems handled by DA has grown up to $n = 100{,}000$ for the current third generation of DA [20, 21, 23]. Besides, the DA is roughly four orders of magnitude faster than a CPLEX solver on conventional computers for quadratic assignment problems [21].

Algorithm 1 Digital Annealer (first generation) Algorithm

```
1:  generate an initial arbitrary solution $x$
2:  for each run do
3:      set the current solution to $x$
4:      $q_{\text{offset}} \leftarrow 0$
5:      for each iteration do
6:          update the temperature if due for temperature update
7:          for each binary variable $x_i$, in parallel do
8:              propose a bit-flip using $\Delta q_i - q_{\text{offset}}$
9:              if acceptance criteria is satisfied, record the flip
10:         if at least one bit-flip met the acceptance criteria then
11:             choose a flip uniformly at random from the record
12:             update the current solution by applying the flip
13:             $q_{\text{offset}} \leftarrow 0$
14:         else
15:             $q_{\text{offset}} \leftarrow q_{\text{offset}}$+ offset_increase_rate
16: return solution found
```

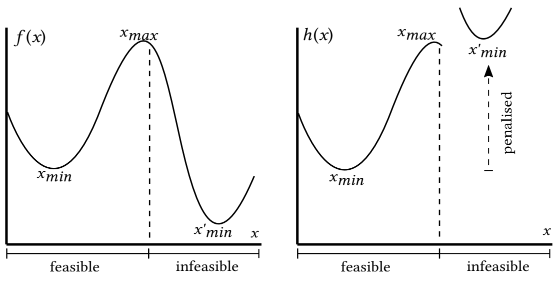
DA performs a randomised meta-heuristic search (see pseudocode in Algorithm 1 [1]) based on simulated annealing but differs from it in the following three aspects. First, the initialisation phase (lines 1–4), where the DA generates an initial candidate solution uniformly at random (or from a prescribed seed) that is reused for each run when the search restarts. Second, the parallel trial phase (lines 5–10), where the DA evaluates in parallel each change in solution cost (denoted Δq_i) between the current candidate solution and the i-th bit-flip neighbour in the search space $\{0, 1\}^n$. If at least one bit-flip is accepted based on the Metropolis-Hastings algorithm, then one of such bit-flips is chosen uniformly at random and applied to the current solution. Third, the dynamic offset phase (lines 10–15), where the DA gradually increases the cost of solutions with a positive offset (denoted q_{offset}) to help the search escape from local minima if no bit-flip move was accepted in the parallel-trial phase. Otherwise the offset is reset to zero. Regarding parameters in Algorithm 1, except for the number of runs and number of iterations per run, the DA is able to automatically set the initial temperature, the temperature update schedule as well as the offset increase rate to suitable default values. For further details about DA's hardware design, search behaviour and parameter settings, the reader is referred to [1, 20, 21, 23].

The penalty methods we present use the DA but do not require it, so other search algorithms may be used instead.

3 EXACT PENALTY METHODS

This section presents a novel, simple, general method to find upper-bounds of the smallest valid penalty weight by deriving bounds on QUBO polynomials (Equation (1)) with known methods. We consider unconstrained quadratic pseudo-Boolean functions of the form $h(x) = f(x) + w \cdot g(x)$, as in Section 2, with fixed penalty weight $w \in \mathbb{N}$ for a given CCO problem. Let us define first what it means for w to be valid.

In QUBO formulations that are equivalent to a given CCO problem, the global optimum value of h must be exactly the same as the feasible global optimum of the constrained problem. Assuming minimisation, if the penalty weight is "too small", the global minimum of h may become infeasible since infeasible solutions have not been penalised enough. Sufficiently large penalty weights avoid that. Figure 1 sketches how infeasible solutions can be penalised to have higher cost than all feasible solutions (Figure 1b) including the global maximum of f, so the global minimum of h is feasible and coincides with the feasible minimum in the constrained problem (Figure 1a).



(a) Constrained problem **(b) Unconstrained problem**

Figure 1: Validity of penalty weights.

We define a penalty weight $w \geq 0$ to be *valid* if and only if: for all pairs $x_{\min}, x' \in \{0,1\}^n$ where $x_{\min} = \operatorname{argmin}_{x \in \{0,1\}^n} h(x)$, $g(x_{\min}) = 0$ and $g(x') \geq 1$, it holds that

$$h_{\min} = h(x_{\min}) = f(x_{\min}) < h(x') = f(x') + w * g(x') \ . \quad (2)$$

What w would be sufficiently large to satisfy Equation (2)? Let $f_{\max}$ and $f_{\min}$ be respectively the (unconstrained) global maximum and minimum objective values. Note that feasible solutions cannot have worse objective value than $f_{\max}$, and all infeasible solutions have $g(x') \geq 1$ by assumption. Then, choosing any penalty weight

$$w > \Delta f = f_{\max} - f_{\min} \ , \quad (3)$$

as illustrated earlier in Figure 1b, would satisfy Equation (2). Thus any upper-bound of such w, derived by upper-bounding $f_{\max}$ and lower-bounding $f_{\min}$, would be valid as well. The following sections present known methods to find these upper and lower bounds.

3.1 Sum of Coefficients Absolute Values

Given that objective functions f are expressible as QUBO polynomials [3, 4], an upper-bound of $f_{\max}$ and lower-bound of $f_{\min}$ can be simply derived from the coefficient values in the constant, linear and quadratic terms as follows:

$$f_{\max} \leq c_0 + \sum_{\substack{i=1 \\ c_i > 0}}^{n} c_i + \sum_{\substack{1 \leq i < j \leq n \\ c_{i,j} > 0}}^{n} c_{i,j} \ , \quad (4)$$

$$f_{\min} \geq c_0 + \sum_{\substack{i=1 \\ c_i < 0}}^{n} c_i + \sum_{\substack{1 \leq i < j \leq n \\ c_{i,j} < 0}}^{n} c_{i,j} \ . \quad (5)$$

Subtracting the lower-bound in Equation (5) from the upper-bound in Equation (4), which is the same as summing the absolute values of coefficients in all linear and quadratic terms, yields an upper-bound of Δf. That is, an upper-bound of the smallest valid penalty weight according to Equation (2).

Example 1. Let the QUBO polynomial $f(x) = 13 - 5x_1 + 9x_2 + x_3 + 12x_4 + 7x_5 - 12x_1x_2 + 8x_1x_4 + 4x_2x_3 - 10x_2x_4 - 6x_3x_4 - 8x_4x_5$. Using Equations (4)-(5), we have $f_{\max} \leq 13 + 9 + 1 + 12 + 7 + 8 + 4 = 54$ and $f_{\min} \geq 13 - 5 - 12 - 10 - 6 - 8 = -28$. Thus, based on Equation (3), a penalty weight greater than $54 - (-28) = 82$ is valid.

Although this method is simple and effective in estimating upper-bounds of valid penalty weights, it can lead to a significantly loose upper-bound compared with best-known penalty weights for a particular problem instance as shown in Section 7. By applying relatively simple transformations to QUBO polynomials, the following method can attain a penalty weight upper-bound which is tighter than the method above or (in the worst case) just as loose.

3.2 Posiform-negaform

Introducing complemented binary variables $\bar{x}_i = 1 - x_i$ allows every QUBO polynomial defined by Equation (1), in fact every pseudo-Boolean function, to be expressed as an equivalent quadratic polynomial called *posiform* such that all linear and quadratic terms have positive coefficients (or zero otherwise) [3]:

$$\phi(x) = a_0 + \sum_{u \in \mathbf{L}} a_u u + \sum_{u,v \in \mathbf{L}} a_{u,v} \, u \, v \quad (6)$$

where $a_u u \geq 0$ and $a_{u,v} \geq 0$ for any pair of literals $u, v \in \mathbf{L} = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. *Negaforms* are the analogous of Equation (6) where $a_u u \leq 0$ and $a_{u,v} \leq 0$. To transform a QUBO polynomial into a posiform (or negaform) is simple: it mainly involves successive substitutions of the form $x_i = 1 - \bar{x}_i$ applied only to negative (or only positive) quadratic and linear terms, with time complexity linear in the number of non-zero terms [4]. Boros and Hammer [3] prove that the constant term a_0 of any posiform for any given QUBO polynomial f is a lower-bound of $f_{\min}$; analogously, a_0 is an upper-bound of $f_{\max}$ for negaforms. From these bounds one can easily derive a valid penalty weight based on Equation (3).

Example 2. Let the QUBO polynomial $f(x) = 13 - 5x_1 + 9x_2 + x_3 + 12x_4 + 7x_5 - 12x_1x_2 + 8x_1x_4 + 4x_2x_3 - 10x_2x_4 - 6x_3x_4 - 8x_4x_5$. A posiform of $f(x)$ is

$$\begin{aligned}\phi(x) &= 0 + 5\bar{x}_1 + 3\bar{x}_2 + x_3 + 4\bar{x}_4 + \bar{x}_5 + 12\bar{x}_1x_2 \\ &\quad + 8x_1x_4 + 4x_2x_3 + 10\bar{x}_2x_4 + 6\bar{x}_3x_4 + 8\bar{x}_4x_5 \ ,\end{aligned}$$

and a negaform of $f(x)$ is

$$\begin{aligned}\phi'(x) &= 49 - 3\bar{x}_1 - 9\bar{x}_2 - 5\bar{x}_3 - 12\bar{x}_4 - 7\bar{x}_5 - 12x_1x_2 \\ &\quad - 8x_1\bar{x}_4 - 4\bar{x}_2x_3 - 10x_2x_4 - 6x_3x_4 - 8x_4x_5 \ .\end{aligned}$$

Therefore, $f_{\min} \geq 0$ and $f_{\max} \leq 49$. So based on Equation (3) a penalty weight greater than 49 is valid. This is significantly tighter than the penalty weight upper-bound 82 obtained in Example 1 by summing absolute coefficient values.

3.3 Verma-Lewis

If c_i and $c_{i,j}$ respectively denote linear and quadratic QUBO coefficients for a given objective function, then the penalty weight proposed by Verma and Lewis [33]

$$w = \max\left\{ c_i + \sum_{\substack{c_{i,j}>0 \\ j\neq i}} c_{i,j} \quad \forall\, i \in \{1,\dots,n\}, \right. \\ \left. -c_i - \sum_{\substack{c_{i,j}<0 \\ j\neq i}} c_{i,j} \quad \forall\, i \in \{1,\dots,n\} \right\} \tag{7}$$

is valid according to Lemma 1 [33]. This validity is a particular case of our validity notion in Equation (2), provided that inequality constraints are converted to equalities and that local search based on bit-flip moves is used, which we respect throughout the experiments in Sections 6–7. Calculating w in Equation (7) has quadratic computational complexity [33], though it can lead to smaller penalty weights as the experiments in Section 7 show later.

4 SEQUENTIAL PENALTY METHODS

This section proposes scaled-sequential and binary search penalty methods as new variants of a traditional form of sequential penalty method described in Section 4.1. These variants differ from traditional sequential methods [10, 30] because they *exploit knowledge from upper-bounds of valid penalty weights*, which one may compute via the exact penalty methods seen before. We hypothesise this knowledge guides our new sequential methods by narrowing the sequence of penalty weights that need to be generated until a valid one is found, and thus obtain better penalty weights in fewer trials. Any solver that can find optimal or suboptimal solutions (e.g. DA [1, 21]) can be used together with the three sequential methods to find feasible minima and small valid penalty weights. Actually, all three methods generate weights on a power scale because: previous research [14] suggests that ballpark estimates of a target penalty weight in practice are not that bad for search performance, and the magnitude order of smallest valid penalty weights can be reached in exponentially fewer trials than a linear scale. Besides, none of the three sequential penalty methods need to start from an initial feasible solution, which is an advantage over certain classic sequential penalty methods that require it [5, 10].

4.1 Sequential Penalty Method

Algorithm 2 is a standard form of known sequential penalty methods [10]. It geometrically increases an initial weight to do a sequence of unconstrained minimisations using DA for a finite maximum number of iterations. The number of iterations is a key parameter that can make all the difference between DA eventually finding feasible solutions or not since we have no upper-bound of a valid penalty weight and thus no indication if the weights generated are sufficiently large. The geometric scaling helps when a problem's best known weight that we target has a relatively high order of magnitude such as 10^6 or greater. It only takes 7 iterations to reach 10^6 compared with the 100 thousand iterations it takes using a linear scaling of 10, 20, 30, etc.

Algorithm 2 Sequential Penalty Algorithm

```
1: $w \leftarrow 1$                                    ▷ initial penalty weight
2: for each iteration do
3:     minimise $h(x) = f(x) + w \cdot g(x)$ with one DA run
4:     if solution found is feasible then
5:         record solution found and $w$
6:     $w \leftarrow w * 10$
7: return minimum feasible solution found and corresponding $w$
```

4.2 Scaled-sequential Penalty Method

The scaled-sequential penalty method in Algorithm 3 works essentially the same as the previous method, but differs from it as follows. Algorithm 3 adapts the scale factor so the generated sequence of weights stays within a given upper-bound w_U of a valid penalty weight; an upper-bound that can be quickly computed by the exact methods in Section 3. We hypothesise this knowledge can lead to fewer iterations until a feasible minimum and valid weight are found. Effectively, the scaled-sequential penalty method can stop iterating once it hits the upper-bound w_U since it is a valid weight.

Algorithm 3 Scaled-sequential Penalty Algorithm

```
Input: $w_U \leftarrow$ upper-bound of a valid penalty weight
Input: $t$, maximum number of iterations
1: $w \leftarrow 1$                                    ▷ initial penalty weight
2: scale_factor $\leftarrow \left\lceil {w_U}^{1/t} \right\rceil$
3: for each iteration do
4:     minimise $h(x) = f(x) + w \cdot g(x)$ with one DA run
5:     if solution found is feasible then
6:         record solution found and $w$
7:     $w \leftarrow w \cdot$ scale_factor
8: return minimum feasible solution found and corresponding $w$
```

4.3 Binary Search Penalty Method

Similar to the scaled-sequential method, the binary search method in Algorithm 4 uses valid penalty weight upper-bounds computed by the exact penalty methods. In fact, both the scaled-sequential and binary search methods are free to use *any* penalty weight upper-bound (regardless of which method produced it) as long as it preserves feasibility of global minima. What makes Algorithm 4 superior to the previous sequential methods is that, at each iteration, it *prunes* by half the range of penalty weights that need to be tried before potentially arriving at a feasible minimum and corresponding valid penalty weight. This translates into a possible exponential speedup over the baseline sequential method in Algorithm 2. Binary search is a well-known algorithm, the novelty here is applying it to find penalty weights (made possible by the availability of upper-bounds).

Algorithm 4 Binary Search Penalty Algorithm

```
Input: $w_U \leftarrow$ upper-bound of a valid penalty weight
 1: interval endpoint $a \leftarrow 1$; interval endpoint $b \leftarrow w_U$
 2: for each iteration do
 3:     penalty weight $w \leftarrow \lceil \sqrt{a \cdot b} \rceil$        ▷ power scale midpoint
 4:     minimise $h(x) = f(x) + w \cdot g(x)$ with one DA run
 5:     if solution found is feasible then
 6:         record solution found and $w$
 7:         $b \leftarrow w$                                ▷ update interval endpoint
 8:     else
 9:         $a \leftarrow w$                                ▷ update interval endpoint
10: return minimum feasible solution found and corresponding $w$
```

5 FORMULATION OF QUBO PROBLEMS

This section summarises the QUBO formulations for a set of well-known CCO problems against which we later benchmark the penalty methods seen in Sections 3–4. We consider the following. First, the minimum cut (Mincut) problem [14], with a binary representation and single logical constraint. Second, the travelling salesman problem (TSP) [19], with a permutation representation and multiple logical constraints. Third, the multi-dimensional 0-1 knapsack problem (MKP) [18], with a binary representation as well as equality and logical constraints. Converting these problems into QUBO is possible via standard reformulation techniques [14]. Here the penalty functions meet the assumption $g(x) \geq 1$ for infeasible solutions x.

5.1 Minimum Cut Problem

Consider the Mincut problem where the vertices of a graph must be partitioned into two equally-sized and disjoint subsets so the number of edges connecting the subsets is minimised:

$$\text{minimise } f(x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} e_{i,j} \cdot a_{i,j} \ , \quad a_{i,j} = \begin{cases} 0 & \text{if } x_i = x_j \\ 1 & \text{if } x_i \neq x_j \end{cases} \tag{8}$$

$$\text{subject to } \sum_{i=1}^{n} x_i = \frac{n}{2} \ . \tag{9}$$

We use the QUBO reformulation of Mincut problems by Matsubara and others [21], where f' corresponds to the objective function f and the penalty function g to the constraint in Equation (9):

$$f'(x) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} \left(x_i + x_j - 2x_i x_j \right) \ , \tag{10}$$

$$g(x) = \left(\frac{n}{2} - \sum_{i=1}^{n} x_i \right)^2 \ . \tag{11}$$

In Equation (10), $e_{i,j}$ is the weight of the edge between vertices i and j; x_i is a binary variable where $x_i = 1$ if the vertex belongs to the first subset or $x_i = 0$ if it belongs to the second.

5.2 Travelling Salesman Problem

Given n locations and a square matrix d for the distances between any two locations, the TSP consists in minimising the distance travelled while visiting each location exactly once and returning to the origin location:

$$\text{minimise } f(\sigma) = \sum_{i=2}^{n} d_{\sigma_{i-1},\sigma_i} + d_{\sigma_n,\sigma_1} \ . \tag{12}$$

In Equation (12), the permutation $\sigma = (\sigma_1, \dots, \sigma_n)$ represents a solution to the TSP, each σ_i with $i \in \{1, \dots, n\}$ represents the i-th location to visit, and $d_{\sigma_{i-1},\sigma_i}$ is the distance between σ_{i-1} and σ_i. We use the equivalent QUBO formulation of the TSP presented by Lucas [19], where f' corresponds to the objective function f and the penalty function g to the additional logical constraints needed for encoding permutations as binary sequences x:

$$f'(x) = \sum_{(i,j) \in \{1,\dots,n\}} d_{i,j} \sum_{k=1}^{n} x_{i,k} \cdot x_{j,k+1} \ , \tag{13}$$

$$g(x) = \sum_{i=1}^{n} \left(1 - \sum_{k=1}^{n} x_{i,k} \right)^2 + \sum_{k=1}^{n} \left(1 - \sum_{i=1}^{n} x_{i,k} \right)^2 \ . \tag{14}$$

Here the two-way one-hot representation is used to encode solutions to the TSP: each location i is represented by a substring of n zeros with the i-th bit set to 1. The encoded solution x can also be described as a 0-1 matrix where each row represents a location. In Equation (14), g penalises any row or column that does not sum to 1, ensuring that x can be decoded to a valid permutation σ.

5.3 Multi-dimensional 0-1 Knapsack Problem

In the MKP we are given a set of m knapsacks with positive capacities $W_k, \forall k \in \{1, \dots, m\}$, and a set of n items with non-negative profit values p_i and weights $w_{i,k}, \forall i \in \{1, \dots, n\}$. The goal is to find a subset of items that maximises the total profit without exceeding the capacities:

$$\text{maximise} \quad f(x) = \sum_{i=1}^{n} p_i \, x_i \ , \tag{15}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_{i,k} \cdot x_i \leq W_k, \quad x_i \in \{0, 1\} \ . \tag{16}$$

We adapt the QUBO formulation for MKP from Coffey [6], converting the maximisation problem into a minimisation problem by changing the sign of the objective function f as $f'(x) = -f(x)$:

$$f'(x) = -\sum_{i=1}^{n} p_i x_i \ , \tag{17}$$

$$g(x) = \sum_{k=0}^{m} \left(\sum_{j=0}^{M_k - 1} 2^j y_j + \left(W_k + 1 - 2^{M_k} \right) y_{M_k} - \sum_{i=1}^{n} w_{i,k} \ x_i \right)^2 \ . \tag{18}$$

We use slack variables y represented in powers of two. So the number of slack variables used for each capacity constraint is limited to $\log_2 W_k + 1$ for the slack range $\{0, \dots, W_k\}$. Therefore, the number of binary variables to represent this problem is $n + \sum_{k=0}^{m} (\log_2 W_k + 1)$. In Equation (18), $2^{M_k} \leq W_k < 2^{M_k+1}$ where $M_k = \log_2 W_k$.

6 EXPERIMENTAL SETTINGS

In our experiments we use the DA and the QUBO formulations seen in Section 5 for the problem instances given in Table 1 [2, 7, 26]. For reproducibility, the used QUBO formulas are available as separate Python's Numpy NPZ files in an online repository [11].

Table 1: Problem sets

Problem	Instance	Original Problem		QUBO Formulation		
		Size	No of Constraints	QUBO Size	No of constraints	Representation
Mincut (C. Walshaw[a])	add20	2395	1	2395	1	Binary
	data	2851	1	2851	1	Binary
	3elt	4720	1	4720	1	Binary
	uk	4824	1	4824	1	Binary
	add32	4960	1	4960	1	Binary
MKP (ORLIB[b])	weing1	28	2	50	2	Binary
	weing3	28	2	50	2	Binary
	weing5	28	2	50	2	Binary
	weing7	105	2	131	2	Binary
	weing8	105	2	131	2	Binary
	weish01	30	5	85	5	Binary
	weish06	50	5	100	5	Binary
	weish12	50	5	110	5	Binary
	weish18	70	5	135	5	Binary
	weish30	90	5	155	5	Binary
TSP (TSPLIB[c])	fri26	26	0	676	2	2-way 1-hot
	bays29	29	0	841	2	2-way 1-hot
	dantzig42	42	0	1764	2	2-way 1-hot
	brazil58	58	0	3364	2	2-way 1-hot
	st70	70	0	4900	2	2-way 1-hot

[a]Source: https://chriswalshaw.co.uk/partition/
[b]Source: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknap2.txt
[c]Source: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

6.1 Parameter Setting

The DA can automatically set the most suitable parameters (e.g. initial temperature, temperature update schedule, etc.) for any given QUBO. We use these default settings when running experiments. The DA has two stopping criteria: *target energy*, which corresponds to the cost to be minimised in the QUBO problems, and *time limit*. We set *target energy* to a known optimal for the problem sets and *time limit* to 20 seconds. So the DA will stop its run if it reaches the known optimal before 20 seconds, otherwise it will stop once 20 seconds of solve time elapse. For each set of experiments the DA is executed independently 20 times. Regarding the sequential penalty methods in Section 4, we allow a maximum of 10 iterations and 20 runs of the DA for each iteration. The experiments stop before 10 iterations if: (1) feasibility is reached using the standard or scaled sequential methods; or, (2) the interval endpoints in the binary search method narrow down to a single penalty weight.

6.2 Performance Measures

To measure solution quality with respect to a known optimal solution, we use the *average relative percentage deviation* (ARPD):

$$\text{ARPD} = \left(\frac{\left| \frac{\sum_{i=1}^{r} \text{Alg}_i}{r} - \text{Optimal} \right|}{\text{Optimal}} \right) \times 100 \ . \tag{19}$$

In Equation (19), r is the total number of runs allowed. We use $r = 20$. The ARPD expresses as a percentage the absolute difference between the known optimal and the solution quality found by the DA averaged over 20 runs. ARPD will be calculated based on feasible solutions. *Time to solution* (TTS) will measure, for a given maximum number of seconds, how long it takes the DA to achieve the best quality of solution found within the allowed time. We allow a maximum of 20 seconds solve time; in practice, the observed TTS may go slightly over 20 seconds because sometimes may be unsafe for DA to stop exactly at 20 seconds.

7 RESULTS

This section benchmarks the performance of the penalty methods seen in Sections 3–4 using the previous experimental settings.

7.1 Results for Exact Penalty Methods

Table 2 shows the penalty weights computed by the exact penalty methods as well as the valid, best known, penalty weights for the problem sets. We have empirically calculated these best known weights, and verified they indeed preserve feasibility of optima, but they are not necessarily the smallest valid penalty weights for the problem instances. Columns "Posi-Nega", "Sum" and "Verma-Lewis" refer respectively to the posiform-negaform, sum of coefficients' absolute values and Verma-Lewis penalty methods seen in Section 3.

Looking at Table 2, all the exact penalty methods produce valid penalty weights across all problem instances. This is expected since all these methods are designed to produce upper-bounds of the smallest valid weights, so respecting our validity condition in Equation (2). Verma-Lewis produces penalty weights that are closest to the best known across all instances and, in particular, for certain Mincut instances identical to the best known. By contrast, Posi-Nega and Sum produce penalty weights that are between three and five orders of magnitude larger than the best known across the problem instances. Therefore, Verma-Lewis is the preferred penalty method if smaller valid penalty weights are preferred. However, Verma-Lewis tends to be the slowest: we noted it can take up to 20 seconds to compute a penalty weight on the larger Mincut and TSP instances, whereas Posi-Nega never needed more than 4 seconds and Sum approximately one second across the problem sets. This is also expected since Verma-Lewis has quadratic time complexity, whereas Sum and Posi-Nega have linear time complexity growing in the number of QUBO coefficients (see Section 3).

Table 2: Penalty weights found by exact penalty methods

Problem Category	Instance Name	Best Known Penalty Weight	Penalty Weights		
			Posi-Nega	Sum	Verma-Lewis
Mincut	add20	93	2,221	29,848	**123**
	data	17	2,823	60,372	**17**
	3elt	9	1,365	54,888	**9**
	uk	3	4,405	27,348	**3**
	add32	31	4,011	37,848	**31**
MKP	weing1	119	164,046	164,045	**30,800**
	weing3	119	164,046	164,045	**30,800**
	weing5	119	164,046	164,045	**30,800**
	weing7	992	1,123,048	1,123,047	**107,200**
	weing8	992	1,123,048	1,123,047	**107,200**
	weish01	1	5,830	5,829	**892**
	weish06	1	6,975	6,974	**892**
	weish12	1	8,605	8,604	**892**
	weish18	1	11,526	11,525	**892**
	weish30	1	13,418	13,417	**892**
TSP	fri26	275	1,750,581	1,750,580	**9,666**
	bays29	473	4,852,049	4,852,048	**17,186**
	dantzig42	190	5,356,261	5,356,260	**10,058**
	brazil58	6,677	408,742,937	408,742,936	**577,104**
	st70	122	17,667,301	17,667,300	**10,110**

Table 3: Average relative percentage deviation achieved by the DA with the exact penalty methods

Problem Category	Instance Name	Optimal	ARPD		
			Posi-Nega	Sum	Verma-Lewis
Mincut	add20	596	0.00	0.00	0.00
	data	189	0.00	5.29	0.00
	3elt	90	26.67	26.67	0.00
	uk	20	190.00	190.00	0.00
	add32	11	809.09	809.09	100.00
		Average	205.15	206.21	**20.00**
MKP	weing1	141,278	0.00	0.00	0.00
	weing3	95,677	0.00	0.00	0.00
	weing5	98,796	0.23	0.26	0.00
	weing7	1,095,445	0.24	0.27	0.25
	weing8	624,319	1.00	1.00	1.02
	weish01	4,554	3.14	3.14	0.64
	weish06	5,557	1.69	1.69	2.50
	weish12	6,339	6.42	3.37	5.36
	weish18	9,580	6.20	6.20	5.91
	weish30	11,191	5.58	4.63	3.08
		Average	2.45	2.05	**1.88**
TSP	fri26	937	0.00	0.00	0.00
	bays29	2020	0.00	0.00	0.00
	dantzig42	699	0.72	0.72	0.72
	brazil58	25395	3.16	3.16	3.16
	st70	675	3.66	3.66	3.67
		Average	**1.51**	**1.51**	**1.51**

Regarding solution quality, Table 3 shows that running DA with weights by Verma-Lewis leads (on average) to the smallest ARPD for Mincut and MKP instances. This agrees with the fact that Verma-Lewis obtained penalty weights significantly smaller than Sum and Posi-Nega (Table 2). For TSP instances, all methods led to the same solution quality. DA's total TTS, however, was slightly smaller with Verma-Lewis for all TSP instances (as shown in Table 4). Similarly, for MKP instances the total TTS was slightly smaller with Verma-Lewis. By contrast, on Mincut instances the method Sum led to the smallest total TTS. Overall, this indicates larger penalty weights can cause premature convergence to suboptimal solutions. Therefore, we conclude the Verma-Lewis method will often lead DA to better solution quality or TTS, which is not surprising given that Sum and Posi-Nega are simpler methods. For all these experiments, DA was able to find feasible solutions (i.e. 100 percent feasibility rate).

Table 4: Average time to solution achieved by the DA with the exact penalty methods

Problem Category	Instance Name	TTS (seconds)		
		Posi-Nega	Sum	Verma-Lewis
Mincut	add20	7.12	7.14	11.27
	data	14.55	3.78	10.12
	3elt	12.32	11.63	15.54
	uk	20.06	20.07	20.17
	add32	20.06	20.05	20.01
	Total Time	74.11	**62.66**	77.11
MKP	weing1	6.60	6.22	6.04
	weing3	1.78	1.88	3.55
	weing5	6.47	2.23	8.36
	weing7	2.85	2.11	6.36
	weing8	8.67	7.93	7.69
	weish01	7.16	7.31	4.22
	weish06	6.69	6.55	3.25
	weish12	9.20	9.24	5.83
	weish18	7.20	7.61	5.10
	weish30	7.90	8.11	8.02
	Total Time	64.52	59.20	**58.42**
TSP	fri26	0.41	0.40	0.41
	bays29	1.47	1.46	1.47
	dantzig42	12.48	12.48	12.49
	brazil58	20.10	20.05	20.11
	st70	15.94	16.01	15.39
	Total Time	50.39	50.40	**49.87**

7.2 Results for Sequential Penalty Methods

This section compares the performance of the sequential methods (Section 4), allowing up to 10 iterations. Except for the standard sequential method, the scaled-sequential and binary search penalty methods are informed by an upper-bound of a valid penalty weight. Here we choose Sum, the exact penalty method in Section 3.1, because it is usually faster than Verma-Lewis and to show that DA performs well even with a loose bound. For all these experiments we obtained 100 percent feasibility rate across all 20 DA runs.

Table 5 shows all sequential methods find the same minimum penalty weight (i.e. $w = 1$) for Mincut instances. Although this weight falls below the best known weights in Table 2, we recall they are not necessarily the smallest valid weights. In fact, since DA consistently finds the feasible optimal solutions for Mincut (see

Table 5: Minimum penalty weights found (and number of iterations to reach the weight) by the sequential methods within 10 iterations, which led to feasible solutions

Problem Category	Instance Name	Minimum Weight (Number of Iterations)		
		Standard Sequential	Scaled Sequential	Binary Search
Mincut	add20	1 (1)	1 (1)	1 (5)
	data	1 (1)	1 (1)	1 (5)
	3elt	1 (1)	1 (1)	1 (5)
	uk	1 (1)	1 (1)	1 (5)
	add32	1 (1)	1 (1)	1 (5)
Average		1 (1)	1 (1)	1 (5)
MKP	weing1	10 (2)	4 (2)	2 (4)
	weing3	10 (2)	14 (3)	5 (6)
	weing5	10 (2)	4 (2)	3 (5)
	weing7	10 (2)	5 (2)	3 (5)
	weing8	1,000 (3)	22 (3)	44 (9)
	weish01	1 (1)	1 (1)	1 (5)
	weish06	1 (1)	1 (1)	1 (5)
	weish12	1 (1)	1 (1)	1 (5)
	weish18	1 (1)	1 (1)	1 (5)
	weish30	1 (1)	1 (1)	1 (5)
Average		105 (2)	5(2)	6 (5)
TSP	fri26	100 (3)	121 (4)	82 (10)
	bays29	1,000 (4)	169 (4)	129 (10)
	dantzig42	100 (3)	175 (4)	36 (9)
	brazil58	10,000 (5)	6,719 (5)	2,017 (10)
	st70	100 (3)	41 (3)	30 (6)
Average		2,260 (4)	1,445 (4)	459 (9)

Table 6), we believe $w = 1$ is valid. Also, when optimal penalty weights are this small, we note that standard and scaled sequential penalty methods need fewer iterations than the binary search method to find weights close optimal. So the Mincut instances and the *weish* MKP instances seem not the best choice to show the potential performance benefits of binary search.

Except for the *weing8* MKP instance, the binary search method achieves the smallest penalty weights across the MKP and TSP instances as shown in Table 5. The scaled-sequential and standard sequential methods were respectively the second and third best method in terms of penalty weight magnitude. But the standard and scaled sequential methods also needed fewer iterations even for the instances where the best known weights are much larger than $w = 1$. We think this is mainly due to our stopping condition being 10 iterations, which could be limited to 3 or 4 iterations without negatively affecting solution quality nor feasibility. In fact, Table 6 shows that the smallest penalty weights do not always lead to the best feasible solutions, by comparing the ARPD with smallest weights (in Table 5) and the best ARPD across the 10 iterations allowed. That is, the final iteration of the sequential methods may not lead to the best results, so the best solution found per iteration should be kept in memory. Still, binary search and scaled sequential achieved optimal solutions on all Mincut instances and, respectively, yield feasible solutions within 9.25% and 11.83% of optimal across all MKP and TSP instances. Although standard sequential performed

Table 6: DA results (20 runs per iteration): Best ARPD derived using the sequential methods of updating penalty weights

Problem Category	Instance Name	Optimal	ARPD using the minimum penalty weight			Best ARPD in 10 iterations		
			Standard	Scaled	Binary Search	Standard	Scaled	Binary Search
Mincut	add20	596	0.50	0.50	0.50	0.00	0.00	0.00
	data	189	0.00	0.00	0.00	0.00	0.00	0.00
	3elt	90	0.00	0.00	0.00	0.00	0.00	0.00
	uk	20	0.00	0.00	0.00	0.00	0.00	0.00
	add32	11	54.55	54.55	54.55	54.55	0.00	0.00
		Average	11.01	11.01	11.01	10.91	0.00	0.00
MKP	Weing1	141,278	1.61	9.79	14.02	0.00	0.00	0.00
	Weing3	95,677	17.24	10.14	29.09	0.00	0.00	0.00
	Weing5	98,796	5.96	31.12	31.12	0.00	0.00	0.00
	Weing7	1,095,445	5.21	3.80	3.35	0.09	0.02	0.06
	Weing8	624,319	0.09	23.00	22.71	0.06	0.39	0.70
	Weish01	4,554	3.93	3.93	3.93	1.62	0.00	0.00
	Weish06	5,557	14.54	14.54	14.54	0.00	0.27	0.50
	Weish12	6,339	39.53	39.53	39.53	0.48	0.02	0.00
	Weish18	9,580	8.04	8.04	8.04	0.72	0.37	0.51
	Weish30	11,191	32.97	32.97	33.10	0.47	0.29	0.29
		Average	12.91	17.69	19.94	0.34	0.14	0.21
TSP	fri26	937	8.00	0.00	13.77	1.54	0.00	0.00
	bays29	2020	0.40	0.79	2.77	0.40	0.40	0.00
	dantzig42	699	1.29	3.15	8.73	1.29	3.15	0.00
	brazil58	25395	13.20	11.83	19.06	13.20	11.83	8.03
	st70	675	6.21	2.87	9.48	6.21	2.87	9.25
		Average	5.82	3.73	10.76	4.53	3.65	3.46

poorly on the *add32* Mincut instance, it led to feasible solutions within 13.20% of optimal across all MKP and TSP instances.

Overall, binary search will often achieve similar or smaller penalty weights compared with the standard and scaled sequential penalty methods. But, from the experiments presented, the scaled sequential method gives the best compromise between penalty weight magnitude, solution quality and number of iterations needed to get there. Nevertheless, we expect the superiority of binary search to stand out in experiments that are less biased towards favouring standard or scaled sequential methods, where the stopping criterion is fairer and the best penalty weights of problems are not as small.

8 CONCLUSIONS

Solving CCO problems with quantum-based or quantum-inspired computers often involves converting these problems into QUBO form. This is possible via penalty methods, where solutions violating problem constraints are penalised. So creating a search bias towards feasible solutions. It is not obvious, however, how to adjust penalty weights: optimal solutions may be displaced by infeasible solutions if the penalty weight is too small. We present exact and sequential penalty methods, which are automatic, scalable and generally applicable to *any* QUBO problem. We analyse experimentally how penalty weights by these methods affect DA performance based on QUBO formulations of Mincut, MKP and TSP. We show the exact methods can produce general penalty weight upper-bounds *guaranteeing* that the unconstrained global optimum is the same as the feasible global optimum, even if such bounds may be loose. For this reason we propose a hybrid approach where sequential penalty methods are informed by the penalty weight upper-bounds from the exact methods. Our experiments corroborated our hypothesis that the hybrid approach can lead to better search performance.

In future work, we plan to benchmark the penalty methods presented on a more diverse set of CCO problems, including portfolio or vehicle routing. Also, we aim to improve exact penalty methods via tighter upper or lower bounds on the global optima of the objective function. Or to explore other penalty methods that exploit information from the constraint function. Some possible approaches may be to refine our notion of valid penalty weights, or deriving tighter bounds via roof duality and fractional programming [3].

REFERENCES

[1] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. 2019. Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer. *Frontiers in Physics* 7, 48 (2019), 1–14. Frontiers. https://doi.org/10.3389/fphy.2019.00048.

[2] John E. Beasley. 1990. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society* 41, 11 (1990), 1069–1072. Springer. https://doi.org/10.1057/jors.1990.166.

[3] Endre Boros and Peter L. Hammer. 2002. Pseudo-Boolean optimization. *Discrete Applied Mathematics* 123, 1–3 (2002), 155–225. Elsevier. https://doi.org/10.1016/S0166-218X(01)00341-9.

[4] Endre Boros, Peter L. Hammer, and Gabriel Tavares. 2006. *Preprocessing of Unconstrained Quadratic Binary Optimization.* Technical Report RUTCOR Research Report RRR 10-2006. Rutgers University, New Jersey, USA.

[5] Carlos A. Coello Coello. 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering* 191, 11-12 (2002), 1245–1287. Elsevier. https://doi.org/10.1016/S0045-7825(01)00323-1.

[6] Mark W Coffey. 2017. Adiabatic quantum computing solution of the knapsack problem. *arXiv preprint arXiv:1701.05584* (2017).

[7] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software* 38, 1 (2011), 1–25. ACM. https://doi.org/10.1145/2049662.2049663.

[8] G. Di Pillo and L. Grippo. 1989. Exact Penalty Functions in Constrained Optimization. *Journal on Control and Optimization* 27, 6 (1989), 1333–1360. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/0327068.

[9] Sebastian Feld et al. 2019. A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer. *Frontiers in ICT* 6, 13 (2019), 1–13. Frontiers. https://doi.org/10.3389/fict.2019.00013.

[10] Anthony V. Fiacco and Garth P. McCormick. 1990. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Classics in Applied Mathematics, Vol. 4. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611971316.

[11] Marcos Diez García. 2022. *FLE-UoE GitHub repository.* https://github.com/marcosdg/FLE-UoE Last accessed on 30 January 2022.

[12] Michel Gendreau and Jean-Yves Potvin. 2019. *Handbook of Metaheuristics.* International Series in Operations Research & Management Science, Vol. 272. Springer, Cham. https://doi.org/10.1007/978-3-319-91086-4.

[13] Fred Glover, Bahram Alidaee, César Rego, and Gary Kochenberger. 2002. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research* 137, 2 (2002), 272–287. Elsevier. https://doi.org/10.1016/S0377-2217(01)00209-0.

[14] Fred Glover, Gary Kochenberger, and Yu Du. 2019. Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models. *4OR–A Quarterly Journal of Operations Research* 17, 4 (2019), 335–371. Springer. https://doi.org/10.1007/s10288-019-00424-y.

[15] Nicolò Gusmeroli and Angelika Wiegele. 2021, in press. EXPEDIS: An exact penalty method over discrete sets. *Discrete Optimization* (2021, in press). Elsevier. https://doi.org/10.1016/j.disopt.2021.100622.

[16] Saïd Hanafi, Ahmed-Riadh Rebai, and Michel Vasquez. 2013. Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems. *Journal of Heuristics* 19, 4 (2013), 645–677. Springer. https://doi.org/10.1007/s10732-011-9169-z.

[17] Richard Manning Karp. 1972. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger (Eds.). Springer, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9.

[18] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Multiple Knapsack Problems.* Springer, Berlin, Heidelberg, 285–316. https://doi.org/10.1007/978-3-540-24777-7_10.

[19] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2, 5 (2014), 1–15. Frontiers. https://doi.org/10.3389/fphy.2014.00005.

[20] Satoshi Matsubara et al. 2017. Ising-Model Optimizer with Parallel-Trial Bit-Sieve Engine. In *Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2017*, Leonard Barolli and Olivier Terzo (Eds.). Advances in Intelligent Systems and Computing, Vol. 611. Springer, Cham, 432–438. https://doi.org/10.1007/978-3-319-61566-0_39.

[21] Satoshi Matsubara et al. 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).* IEEE, Beijing, China, 667–672. https://doi.org/10.1109/ASP-DAC47756.2020.9045100.

[22] Catherine C. McGeoch, Richard Harris, Steven P. Reinhardt, and Paul I. Bunyk. 2019. Practical Annealing-Based Quantum Computing. *Computer* 52, 6 (2019), 38–46. IEEE. https://doi.org/10.1109/MC.2019.2908836.

[23] Hiroshi Nakayama et al. 2021. *Third Generation Digital Annealer Technology.* Technical Report. Fujitsu Ltd.

[24] Christian F. A. Negre, Hayato Ushijima-Mwesigwa, and Susan M. Mniszewski. 2020. Detecting multiple communities using quantum annealing on the D-Wave system. *PLOS ONE* 15, 2 (2020), 1–14. Public Library of Science. https://doi.org/10.1371/journal.pone.0227538.

[25] WangChun Peng et al. 2019. Factoring larger integers with fewer qubits via quantum annealing with optimized parameters. *Science China Physics, Mechanics & Astronomy* 62, 6 (2019), 1869–1927. Springer. https://doi.org/10.1007/s11433-018-9307-1.

[26] Gerhard Reinelt. 1991. TSPLIB—A Traveling Salesman Problem Library. *INFORMS Journal on Computing* 3, 4 (1991). Springer. https://doi.org/10.1287/ijoc.3.4.376.

[27] Gili Rosenberg, Poya Haghnegahdar, Phil Goddard, Peter Carr, Kesheng Wu, and Marcos López de Prado. 2016. Solving the Optimal Trading Trajectory Problem Using a Quantum Annealer. *IEEE Journal of Selected Topics in Signal Processing* 10, 6 (2016), 1053–1060. IEEE. https://doi.org/10.1109/JSTSP.2016.2574703.

[28] Masataka Sao et al. 2019. Application of Digital Annealer for Faster Combinatorial Optimization. *Fujitsu Scientific & Technical Journal* 55, 2 (2019), 45–51. Fujitsu Ltd.

[29] David Sherrington and Scott Kirkpatrick. 1975. Solvable Model of a Spin-Glass. *Physical Review Letters* 35, 26 (1975), 1792–1796. American Physical Society. https://doi.org/10.1103/PhysRevLett.35.1792.

[30] Dong Ku Shin, Z. Gürdal, and O. H. Griffin Jr. 1990. A Penalty Approach for Nonlinear Optimization with Discrete Design Variables. *Engineering Optimization* 16, 1 (1990), 29–42. Taylor & Francis. https://doi.org/10.1080/03052159008941163.

[31] Peter F. Stadler. 1995. Towards a Theory of Landscapes. In *Complex Systems and Binary Networks*, Ramón López-Peña et al. (Eds.). Lecture Notes in Physics, Vol. 461. Springer, Berlin, Heidelberg, 78–163. https://doi.org/10.1007/BFb0103571.

[32] Amit Verma and Mark Lewis. 2020. Optimal quadratic reformulations of fourth degree pseudo-Boolean functions. *Optimization Letters* 14, 6 (2020), 1557–1569. Springer. https://doi.org/10.1007/s11590-019-01460-7.

[33] Amit Verma and Mark Lewis. 2020. Penalty and partitioning techniques to improve performance of QUBO solvers. *Discrete Optimization* (2020). Elsevier. https://doi.org/10.1016/j.disopt.2020.100594.