Heuristic Approaches for Quadratic Unconstrained Binary Optimization

Jan-Erik Hein[1], Lukas Mehl[2] and Paul Matti Meinhold[3]
February 28, 2024

[1]TU Berlin, Student ID: 363120
[2]TU Berlin, Student ID: 500723
[3]TU Berlin, Student ID: 490796

Introduction

The Quadratic Unconstrained Binary Optimization (QUBO) problem is not only interesting from a theoretical point of view, but also arises in many applications to real world problems. Given a matrix $Q \in \mathbb{R}^{n\times n}$, the QUBO problem is defined as

$$\min_{x\in\{0,1\}^n} x^T Q x. \tag{1}$$

Rewriting the problem as

$$x^T Q x = \sum_{i,j=1}^{n} q_{ij} x_i x_j = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (q_{ij} + q_{ji}) x_i x_j + \sum_{i=1}^{n} q_{ii} x_i,$$

where q_{ij} is the entry of Q in row i and column j, we observe that the problem can be reformulated into an equivalent problem over an upper triangular matrix $\tilde{Q}$, given by $\tilde{q}_{ij} = q_{ij} + qji$ for $i < j$ and $\tilde{q}_{ii} = q_{ii}$ for $i = 1, \ldots, n$. Therefore, without loss of generality, we will from now on assume Q to be upper triangular. With this assumption, in order to simply notation throughout this paper we denote $q_{\{i,j\}}$ to be the upper triangular entry corresponding to the indices i and j, i.e.

$$q_{\{i,j\}} := \begin{cases} q_{ij} & \text{if } i \le j \\ q_{ji} & \text{if } j < i \end{cases}.$$

QUBO is strongly NP-hard, as shown in Chapter 3.2 of Punnen's book [1] and closely related to many other important problem classes. Specifically, Punnen highlights the equivalence to the Maximum Cut Problem, which allows solution techniques specialized on either to the two problem classes to be applied interchangeably after performing the respective problem transformations. Moreover, Punnen notes that any instance of the binary integer program can be formulated as a QUBO instance. In addition, QUBO plays an important role in physics and quantum computing, where the significance comes mainly from its equivalence to the Ising problem. For a more comprehensive overview of closely related problems and applications we again refer to Punnen [1].

Solution methods for QUBO can generally be split into two classes: *exact solution methods* and *heuristic solution methods*. In order to find exact solutions for the QUBO problem, it can be reformulated as a (mixed) integer linear program (MILP), as shown in Chapter 6.2 of [1], such that a general purpose solver, such as SCIP (see [2]), can be used. Exact algorithms specifically tailored for QUBO are mostly of enumerative type and mostly use branch-and-bound as e.g. in Li et al. [3], in some cases relying on semi-definite problem (SDP) relaxations rather than linear (LP) relaxations, as for example Helmberg and Rendl [4] do. For a more comprehensive survey on the different approaches we refer to Chapter 6 in Punnen's book [1] and Chapter 3 in Kochenberger et al. [5].

On the other hand, the lack of constraints present in QUBO problems motivates the use of metaheuristics for finding near-optimal solutions. For QUBO, ensemble search strategies that combine multiple metaheuristics, have been shown to perform well, as Kochenberger [5] point out. One common metaheursitic approach that has been applied successfully to QUBO is *local search*. Here, one iteratively tries to improve a given solution by small or 'local' changes also called moves. Alkhamis [6] present a local search approach based on simulated annealing, whereas Glover et al. [7] propose a tabu search approach, which we will adopt and modify in our work. Other metaheuristic approaches were also applied to QUBO, for example the scatter search approach described by Amini [8]. While Chapter 8 and 9 of Punnen's book [1] give more information on heuristic approaches and Kochenberger et al. [5] list prominent heuristics, a systematic evaluation can be found in Dunning et al. [9].

In our work, we aim to develop a heuristic approach for solving QUBO problems that allows finding solutions that are competitive with respect to upper bounds presented in the literature. Here our main contribution is the adaption of the Tabu Search method D^2TS proposed by Glover et al. [7], where we propose a custom diversification procedure based on a different long-term memory structure. In the following we first present the details of our approach and then discuss our experiments and the results.

Methodology

Our methodology employed is structured around a three-step approach. Firstly, we apply a set of simple preprocessing rules to identify an equivalent formulation of reduced size. Subsequently, we adopt a greedy rounding scheme to generate an initial solution, which then serves as the initial solution for a tabu search algorithm. Here the algorithm exploits a specific data structure, enabling an efficient neighbor selection in linear time. Moreover, supplementary data structures are integrated to facilitate diversification. In the following we provide a detailed description of these three steps.

Preprocessing

In our preprocessing, we aim to construct a matrix $\tilde{Q}$ of smaller size than Q such that

$$\min_{x \in \{0,1\}^m} x^T \tilde{Q} x + C = \min_{x \in \{0,1\}^n} x^T Q x, \tag{2}$$

for some constant C, where any optimal solution to the preprocessed problem can be extended into an optimal solution for the original problem. To this end, we apply a set of simple rules to identify a subset of variables that are guaranteed to be 0 or 1 in any optimal solution, as well as a subset of redundant variables that do not impact the value of an optimal solution and can thus be removed from the model. Here we basing our approach on the simple rules presented by Lewis and Glover [10]. For a more advanced preprocessing, we refer to Boros et al. [11].

The core idea behind the preprocessing rules discussed by Lewis and Glover is to define bounds on the *contribution* $f_i(x)$ each variable x_i has on the objective f, which is given as

$$f_i(x) = \begin{cases} \sum_{j=1}^{n} q_{\{i,j\}} x_j, & \text{if } x_i = 1 \\ 0 & \text{if } x_i = 0. \end{cases}$$

To that end they denote Q_i^+ to be the sum over all non-diagonal positive entries in column and row i of Q, i.e.

$$Q_i^+ = \sum_{\substack{j=1 \\ q_{\{i,j\}}>0, j \neq i}}^{n} q_{\{i,j\}}.$$

Analogously they define Q_i^- be the sum over all non-diagonal negative entries in column and row i of Q. Note that the largest possible contribution $f_i(x)$ to the objective is then obtained by setting

$$x_j = \begin{cases} 1 & \text{if } q_{\{i,j\}} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

which results in $f_i(x) = q_{ii} + Q_i^+$. This then leads to the following rule.

Rule 1 *If $q_{ii} + Q_i^+ \leq 0$, then setting $x_i = 1$ in any optimal solution also gives an optimal solution. More formally, for any optimal solution x^*, the solution x' with $x'_i = 1$ and $x'_j = x^*_j$ f.a. $j \neq i$, is also optimal.*

The second rule can be derived analogously to Rule 1. Here the idea is that, if the smallest possible contribution of a variable x_i is positive, then x_i can be set to 0.

Rule 2 *If $q_{ii} + Q_i^- \geq 0$, then setting $x_i = 0$ in any optimal solution also gives an optimal solution.*

The idea behind the third rule is similar to Rule 1, applied to not one but two variables. As argued above, the greatest possible contribution to the objective of setting $x_i = 1$ or $x_j = 1$ is $q_{ii} + Q_i^+$ or $q_{jj} + Q_j^+$, respectively. Hence, the greatest possible contribution of setting $x_i = x_j = 1$ can be upper-bounded by $q_{\{i,j\}} + q_{jj} + Q_j^+ + q_{ii} + Q_i^+$ and we can formulate the following rule.

Rule 3 *Assume $q_{ii} + Q_i^+ > 0$ and $q_{jj} + Q_j^+ > 0$. If $q_{\{i,j\}} + q_{jj} + Q_j^+ + q_{ii} + Q_i^+ \leq 0$, then setting $x_i = x_j = 1$ in any optimal solution also gives an optimal solution.*

Finally, it is apparent that given a row and column consisting only of zeros, the corresponding variable is redundant and can be removed from the model without affecting the objective of an optimal solution. As Lewis and Glover [10] state, even though one would not expect a given instance to have this property, it could occur after performing the previous three preprocessing rules.

Rule 4 *If both row and column i consist only of zeros, then x_i is redundant has no effect on the objective.*

After identifying a variable to be fixed we aim to remove the corresponding column and row of the matrix Q. However to ensure the equivalence in (2) we have to perform additional modifications to the matrix. Observe that if a variable x_i is fixed to 0, then $q_{\{i,j\}}$ does not contribute to the objective for any index j, independent of the other values in x. Thus, after removing the i-th row and column, no further modifying of Q is required. On the other hand if x_i is fixed to 1, observe that $q_{\{i,j\}}$ contributes to the objective if and only if $x_j = 1$. As, for any j, q_{jj} contributes to the objective if and only if $x_k = 1$, we can thus add x_{ii} to C and add $q_{\{i,j\}}$ to q_{jj} for all indices $j \neq i$. Then, after removing the i-th row and column of Q the equivalence in (2) still holds.

Lewis and Glover [10] formulate these rules into a simple preprocessing algorithm, which we give an overview of in Algorithm 1. In the following we provide more detail and analyze the runtime. The initial computation of Q_k^+ and Q_k^- for all $k \in \{1, \ldots, n\}$ in line 1 can be done in linear time. Having stored these values, checking for a given index i (and possibly j) whether one of the rules is applicable can be done in constant time. Remark that we refer to a second index j since Rule 3 refers to two indices. Hence, in line 2, finding an index i (and possibly j) such that one of the rules can be applied takes $\mathcal{O}(n^2)$. The updating of Q and the computation of C are explained above and take linear time in total. Remark that this can be done in place, i.e. the values of Q will not be copied for each update. Similarly, the updating of all values Q_k^+ and Q_k^- can be done in linear time, by simply subtracting $q_{\{i,k\}}$ from Q_k^+ if positive or Q_k^- if negative. In total, the preprocessing takes $\mathcal{O}(n^2)$ time for each fixation. Remark that if the number of fixations reaches the magnitude of n, the run time would be cubic. This could be avoided by omitting Rule 3. However we do not expect this to happen and even if so, the reduction achieved by preprocessing would be significant.

Algorithm 1 Preprocessing (Overview)

```
Input: initial $Q \in \mathcal{Q}^{n\times n}$
Output: reduced $\tilde{Q} \in \mathcal{Q}^{m\times m}$ where $m \leq n$
and constant $C$ for objective
1: Compute $Q_k^+$, $Q_k^-$ f.a. indices $k$
2: while one of the rules 1-4 can be applied to an index $i$ (and possibly $j$) do
3:     Fix $x_i$ (and possibly $x_j$)
4:     Update $Q$ accordingly
5:     Store corresponding constant $C$ for objective
6:     Update the computed $Q_k^+$, $Q_k^-$ f.a. indices $k$
7: return (updated) $Q$ and $C$
```

Start Heuristic

After performing the initial preprocessing we employ a greedy rounding scheme to compute an initial solution for the Tabu Search method. Here, our approach in mainly based on the ideas formulated by Punnen in Chapter 8.2 of his book [1].

The basic idea is as follows. Starting from a fractional hint vector $x^{hint} \in (0,1)^n$, the heuristic iteratively rounds one of the fractional variables to either 0 or 1. Note that the resulting solution depends on the order in which the fractional variables are considered. Contrary to the randomized approach proposed by Punnen, we opted for a greedy approach for the order in which the variables are selected. That is, given a partially fractional $x \in [0,1]^n$ at some iteration, we select the fractional variable $x_k \in (0,1)$ and a rounding direction (i.e. up to 1 or down to 0) such that

$$\tilde{x}^T Q \tilde{x} - x^T Q x = \sum_{j=1}^{n} q_{\{k,j\}} \tilde{x}_j \tilde{x}_k - \sum_{j=1}^{n} q_{\{k,j\}} x_j x_k = (\tilde{x}_k - x_k)(q_{kk} + \sum_{j=1, j\neq k}^{n} q_{\{k,j\}} x_j)$$

is minimized, where $\tilde{x}$ is the new fractional solution. For a rounding direction $r \in \{\uparrow, \downarrow\}$, denoting rounding a variable up or down respectively, and an index $i \in \{1, \dots, n\}$, let

$$\Delta^r x_i := \begin{cases} (\lceil x_i \rceil - x_i)(q_{ii} + \sum_{j=1, j\neq i}^{n} q_{\{i,j\}} x_j) & \text{if } r = \uparrow, \\ (\lfloor x_i \rfloor - x_i)(q_{ii} + \sum_{j=1, j\neq i}^{n} q_{\{i,j\}} x_j) & \text{if } r = \downarrow. \end{cases} \tag{3}$$

Then the rounding operation that minimizes $\Delta^r x_i$ is performed. The whole procedure is displayed in Algorithm 2.

We now want to briefly analyze the runtime. The computation of the initial $\Delta^\uparrow x, \Delta^\downarrow x$ is done straightforward by the formula 3 in linear time. Choosing the arg min in line 5 takes $\mathcal{O}(n)$ as well. Now consider the update step in line 7. Note that we do not need to update any index that is not in N, as those will not be considered again. We turn to formula 3 again and calculate

$$\Delta^\uparrow \tilde{x}_i - \Delta^\uparrow x_i = (\lceil x_i \rceil - x_i)(\tilde{x}_k - x_k) q_{\{k,i\}},$$

where unlike as in the pseudo code, x is the old and $\tilde{x}$ is the new fractional solution. The equation for $\Delta^\downarrow$ is analogue. Hence this update step can be done in constant time for each index $i \in N$. We conclude that one iteration of the while loop takes $\mathcal{O}(n)$ time, so the total runtime is $\mathcal{O}(n^2)$.

Note, that the choice of the original fractional hint vector also impacts the quality of the resulting solution, where a hint vector that is close to the optimal binary solution will likely lead to a better rounding solution. This motivates the following three heuristic choices for constructing the intial hint vector, which we will anlalyze in more detail as part of our experiments.

Algorithm 2 Greedy Rounding heuristic

```
Input: matrix $Q \in \mathcal{Q}^{n \times n}$,
 hint vector $x^{hint} \in (0,1)^n$
Output: feasible solution $x \in \{0,1\}^n$
1: $x \leftarrow x^{hint}$
2: Compute initial $\Delta^{\uparrow} x, \Delta^{\downarrow} x$
3: Initialize set $N = \{1, \dots, n\}$        ▷ $N$ is the set of indices corresp. to not yet rounded variables
4: while $N \neq \emptyset$ do
5:     $(k, r) \leftarrow \arg\min_{j \in N, r \in \{\downarrow, \uparrow\}} \Delta^r x_j$
6:     $x_k \leftarrow \begin{cases} 1 & \text{if } r = \uparrow \\ 0 & \text{if } r = \downarrow \end{cases}$
7:     $N \leftarrow N \setminus \{k\}$
8:     Update $\Delta^{\uparrow} x_i$, $\Delta^{\downarrow} x_i$ accordingly f.a. $i \in N$
9: return $x$
```

Constant hint vector The simplest choice for a given hint vector is to consider an unbiased hint vector, where each value x_i is set to 0.5. We will use this naive hint vector as a baseline for evaluating the following two biased approaches.

Bias with respect to number of negative entries Let n_i be the number of non-zero entries and n_i^- be the number of negative entries in the i-th column and row of Q, respectively. Then we set $x_i = n_i^- / n_i$. Here the motivation is that setting a variable with a high number of negative entries will likely be set to 1 in an optimal solution, whereas a variable with a high number of positive entries will likely be set to 0.

Bias with respect to sum over negative entries Let c_i be the sum over all absolute values of entries and c_i^- be the sum over absolute values of negative entries in the i-th row and column of Q respectively. Then we set $x_i = c_i^- / c_i$. Here the motivation is similar as above, however we introduce an bias based on not only the number of negative entries but also the actual values of the negative entries, as variables with a small number of highly negative entries might still be likely to be set to 1 in an optimal solution.

In our start heuristic we perform Algorithm 2 for each of the three hint vectors above and select the best solution as the start solution for the tabu search, which we will discuss next.

Tabu Search

In a similar fashion to the D^2TS algorithm proposed by Glover et al. [7], our approach alternates between a simple version of Tabu Search and a diversification phase utilizing a long-term frequency measure.

Starting from an initial solution obtained by the rounding heuristic discussed in the previous section, the algorithm performs standard Tabu Search iterations until no globally improving solution was found for some fixed number of iterations. Then the search state of the best solution found throughout the search is restored and a short diversification phase is initiated to steer the search towards previously unvisited areas of the search space. To aid the neighbor selection process during diversification, a special frequency-based memory structure is used to add additional penalties for solutions features that where commonly observed throughout the search.

In the following we give a detailed explanation of the neighborhood relation and neighbor selection process, as well as the diversification procedure. Furthermore the diagram in Figure 1 provides a general overview of the search process.

1-flip neighborhood Following the approach of the D^2TS algorithm we base our algorithm on the well-known *1-flip neighborhood* structure, where a neighboring solution can be obtained by replacing a single binary solution entry x_i by its complementary value $1 - x_i$. Evidently this neighborhood definition ensures that the underlying search space is connected, as any two binary vector of length n can be transformed into another by performing at most n consecutive 1-flips.

It is apparent that the main prerequisite for an efficient implementation of the tabu search method is a fast evaluation method for determining the impact a given 1-flip has on the objective f. Note, that a naive evaluation of all moves requires iterating over all ρn^2 non-zero elements of the matrix Q, where ρ is the density of Q. Assuming a constant density, this leads to a time complexity of $O(n^2)$ for a single iteration of the tabu search method and greatly reduces the number of iterations one could expect to perform on larger instances in a reasonable amount of time. To circumvent this, the D^2TS algorithm maintains a list storing the objective changes of all moves throughout the search and provides an efficient method for updating this list in linear time after performing a given swap. We will refer to the objective change of applying a given move x_i as the *flip delta*, which we denote by Δx_i.

Specifically Glover et al. observe that for a given solution x the flip deltas are given by

$$\Delta x_i = (1 - 2x_i)\left(q_{ii} + \sum_{j=1, j\neq i}^{n} q_{\{i,j\}} x_j\right) \tag{4}$$

Observe that $1 - 2x_i$ defines a sign on the sum based on the value of x_i. Thus after swapping x_i the flip value Δx_i gets negated. Furthermore note that for any $j \neq i$ we have to either add or subtract $q_{\{i,j\}}$ from Δx_j, depending on the original binary variable of x_i and x_j. More precisely let $\tilde{x}$ be the solution obtained after swapping x_i in x. Then, for $k \neq i$ we have

$$\Delta \tilde{x}_j = \begin{cases} \Delta x_j + q_{\{j,i\}} & \text{if } x_i = x_k \\ \Delta x_j - q_{\{j,i\}} & \text{if } x_i \neq x_k. \end{cases}$$

As updating any given flip value can thus be performed in constant time, we can formulate a linear time algorithm for updating all flip values after each performed move, which we describe in Algorithm 3.

Algorithm 3 Updating Flip Deltas

```
1: $\Delta x_i \leftarrow -\Delta x_i$
2: for $k \in \{1, \ldots, n\} \setminus \{i\}$ do
3:     if $x_i = x_k$ then
4:         $\Delta x_k \leftarrow \Delta x_k + q_{\{k,i\}}$
5:     else
6:         $\Delta x_k \leftarrow \Delta x_k - q_{\{k,i\}}$
```

As a side remark, we want to mention the similarity between the flip deltas introduced here and the notation rounding operations declared in section Start Heuristic. In both concepts, we look at the change of the objective caused by changing one variable. Unlike as in the rounding heuristic, there is no need to specify the direction of the change for the flip deltas as we only look at binary solutions in this section.

Neighbor Selection Our neighbor selection process follows the standard procedure popularized by Glover and Laguna [12]. As with most local search methods, Tabu Search allows the search to perform non-improving moves in order to escape non-optimal local optima of the objective function.

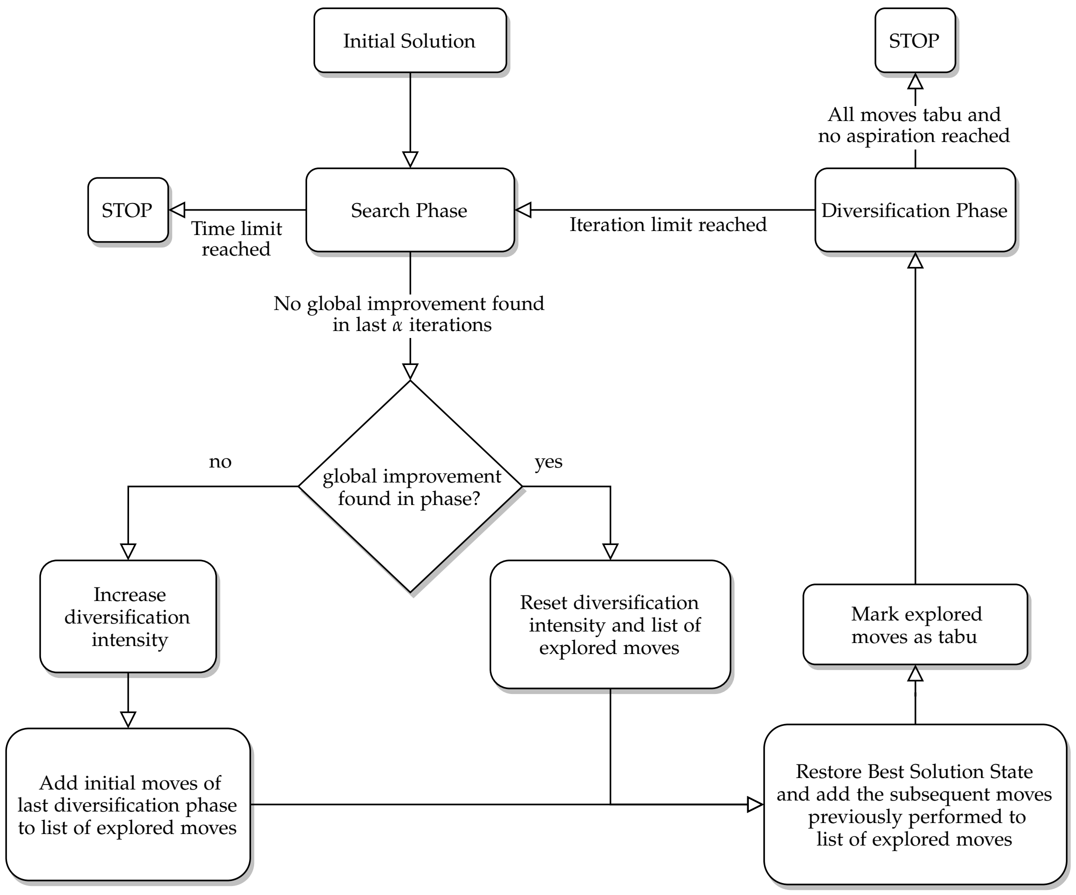
Initial Solution
STOP
All moves tabu and no aspiration reached
STOP
Time limit reached
Search Phase
Iteration limit reached
Diversification Phase
No global improvement found in last α iterations
no
yes
global improvement found in phase?
Increase diversification intensity
Reset diversification intensity and list of explored moves
Mark explored moves as tabu
Add initial moves of last diversification phase to list of explored moves
Restore Best Solution State and add the subsequent moves previously performed to list of explored moves

Figure 1: *Overview of the Tabu Search process*

However, the main difficulty of allowing non-improving moves is that it permits cycles in the search path. Typically, Tabu Search methods prevent cycling by incorporating a short-term memory structure that is used to restrict the neighborhood to solutions that do not reintroduce certain features that have been changed in recent iterations. For neighborhood relations based on self-inverse moves, such as the 1-flip neighborhood, the standard implementation is to define a suitable *tabu tenure* for each applied move. The tabu tenure defines for how many search iterations a move is excluded from consideration after it is performed, where the iteration number at which the move becomes available again is typically stored in a *tabu list*.

Similarly to the D^2TS algorithm our algorithm uses a tabu tenure that is scaled linearly to the problem size n, that is we have a tenure length of n, where λ is a model parameter we refer to as the *tenure ratio*.

Limiting the search to non-tabu moves can prove to be too restrictive and may exclude good solutions that do not lead to cycles. Therefore Glover et al. highlights the importance of incorporating so-called *aspiration criteria* into the search, which are sufficient criteria for a neighboring solution to not lead to a cycle. If an aspiration criterion is fulfilled the tabu status of the respective move can thus be disregarded. We use the standard approach for such an aspiration criterion where a tabu move gets accepted only if it leads to a solution that is a global improvement with respect to the objective f, that is its objective value is better than that of all other solutions found throughout the search.

Diversification After each standard search phase, the state of the best solution found so far is restored and a short diversification phase with a fixed number of iterations is performed, for which a modified objective is used. Contrary to the D^2TS algorithm, where the modified objective relies on a simple frequency vector counting the number of times each 1-flip was performed throughout the search, we base our diversification procedure on a matrix structure which we describe in the following.

For a given solution x we say an entry $\{i, j\}$ is *active*, if and only if $x_i = x_j = 1$ and *inactive* otherwise. That is, $\{i, j\}$ is active if and only if $q_{\{i,j\}}$ is part of the objective $f(x)$. Then for a neighboring solution $\tilde{x}$ to x let

$$\begin{aligned} M(x, \tilde{x}) =& \{\{i, j\} : \tilde{x}_i = \tilde{x}_j = 1 \land (x_i = 0 \lor x_j = 0)\} \\ & \cup \{\{i, j\} : (\tilde{x}_i = 0 \lor \tilde{x}_j = 0) \land x_i = x_j = 1\} \end{aligned}$$

be the set of entries that are activated or deactivated by the given move. We then define a *frequency matrix* $F \in \mathbb{Z}^{n \times n}$ that stores the overall count of activations and deactivations of entries throughout the search. Specifically, for a given solution path $(x^{(0)}, x^{(1)}, \ldots, x^{(m)})$ the frequency matrix is an upper triangular matrix $F = (F_{\{i,j\}})$ defined by

$$F_{\{i,j\}} = \left| \left\{ k \in \{1, \ldots, m\} : \{i, j\} \in M\left(x^{(k-1)}, x^{(k)}\right) \right\} \right|,$$

Here $F_{\{i,j\}}$ again denotes the entry in row i and column j of F for $i \leq j$, and the entry in row j and column i of F for $i > j$.

The frequency matrix is then used to incentivize moves towards solutions that activate or deactivate entries which were rarely activated or deactivated throughout the search. To that end we consider the modified objective

$$f_D(\tilde{x}) = \tilde{x}^T Q \tilde{x} + \gamma \sum_{\{i,j\} \in M(x, \tilde{x})} F_{\{i,j\}}, \tag{5}$$

where we refer to $\gamma > 0$ as the *diversification intensity*.

Note that the impact of the frequency penalties during diversification not only depends on the intensity parameter γ but also on the distribution of values in the two matrices Q and F. Therefore we define a *base value* $\gamma_0(Q, F)$ of the diversification intensity as

$$\gamma_0(Q, F) = \mu \cdot \frac{\|Q\|_1}{\|F\|_1},$$

where μ is a model parameter we call the *diversification base factor*. For sake of notation we will abbreviate $\gamma_0(Q, F)$ in the following to γ_0. The diversification intensity is then obtained by scaling the base value in an adaptive way throughout the search, which we discuss in more detail in the next subsection.

To ensure an efficient move evaluation during diversification, special care has to be taken in evaluating the sum over the frequency values in equation 5. Note that a naive implementation would lead to a linear time for evaluating a single move, ultimately leading to a quadratic runtime for each iteration of the diversification. Similarly to the data structure used for storing and maintaining flip values throughout the search we maintain a list storing the frequency penalties incurred by any given move which then allows evaluating a given move in constant time. Furthermore, we show that the list can be updated in linear time after performing a given move and thus allows for an overall linear runtime for each iteration of the diversification procedure.

In order to formulate the data structure used for storing frequency penalties we first introduce the notions of *partially* and *fully inactive* entries. Given an entry $\{i, j\}$ and a solution x, we say $\{i, j\}$ is partially inactive if precisely one of the values x_i and x_j is 0, and fully inactive if both values are 0. We consider inactive diagonal elements to always be partially inactive.

Let $\tilde{x}$ be obtained by swapping the value x_i in x. We can observe that if x_i is flipped to 1, all partially inactive entries $\{i,j\}$ are activated, whereas if x_i is flipped to 0, all active entries $\{i,j\}$ are deactivated. Thus we can reformulate the frequency sum in (5) to

$$\sum_{\{i,j\}\in M(x,\tilde{x})} F_{\{i,j\}} = \begin{cases} \sum\limits_{j:\{i,j\}\text{ active}} F_{\{i,j\}}, & x_i = 1, \\ \sum\limits_{j:\{i,j\}\text{ part. inactive}} F_{\{i,j\}}, & x_i = 0. \end{cases} \tag{6}$$

Furthermore note that in case x_i gets flipped to 1, all fully inactive entries $\{i,j\}$ become partially inactive and in case x_i gets flipped to 0, all partially inactive entries $\{i,j\}$ become fully inactive.

Algorithm 4 Maintaining frequency penalties

```
    Input: $\tilde{x}$ obtained by flipping $x_i$ in $x$
 1: $F_{\{i,i\}} \leftarrow F_{\{i,i\}} + 1$                    ▷ Since $x_i$ was flipped, $q_{\{i,i\}}$ is (de)activated
 2: if $x_i = 0$ then
 3:     active[$i$] ← partial[$i$]+1                ▷ '+1' is due to the update of $F_{\{i,i\}}$ in line 1
 4:     partial[$i$] ← inactive[$i$]
 5:     inactive[$i$] ← 0
 6: else
 7:     inactive[$i$] ← partial[$i$]
 8:     partial[$i$] ← active[$i$]+1                ▷ '+1' is due to the update of $F_{\{i,i\}}$ in line 1
 9:     active[$i$] ← 0
10: for $j \in \{1,\dots,n\}\setminus\{i\}$ do
11:     $\sigma \leftarrow 1$ if $x_i = 0$ else $-1$                       ▷ $\sigma$ is used below in lines 20-24
12:     if $x_j = 1$ then
13:         $F_{\{i,j\}} \leftarrow F_{\{i,j\}} + 1$     ▷ Since $x_i$ was flipped and $x_j = 1$, $q_{\{i,j\}}$ is (de)activated
14:         if $x_i = 1$ then
15:             active[$j$] ← active[$j$]+1
16:             active[$i$] ← active[$i$]+1
17:         else
18:             partial[$j$] ← partial[$j$]+1
19:             partial[$i$] ← partial[$i$]+1
20:         active[$j$] ← active[$j$]+$\sigma F_{\{i,j\}}$
21:         partial[$j$] ← partial[$j$]−$\sigma F_{\{i,j\}}$
22:     else
23:         partial[$j$] ← partial[$j$]+$\sigma F_{\{i,j\}}$
24:         inactive[$j$] ← inactive[$j$]−$\sigma F_{\{i,j\}}$
```

Our method for maintaining the frequency penalties utilizes these observations and is outlined in Algorithm 4. Here the algorithm stores the frequency sums over active, partially and fully inactive entries throughout the search. Specifically, for each of these three entry types, we maintain one array of length n. In an entry of such an array which corresponds to an index i, we store the frequency sums over all entries $\{i,j\}$ that have the respective entry type. Then after flipping a given variable x_i all (de)activations are registered in the frequency matrix F. This modification of F happens in line 1 and line 13, if reached. For the arrays storing the sums over the frequency penalties, two updates have to be performed: They need to be modified to reflect the changed state of F as well as the changes in the entry types with respect to the new solution $\tilde{x}$. For i, the update of the entries in the arrays are executed in lines 2-9. For all other indices, this is done in the for loop. The steps in lines

14-19 perform the update to reflect the changed state of F. The last 5 lines ensure that the arrays comply with the changes in the entry types with respect to the new solution $\tilde{x}$.

Note that our algorithm requires a constant number of operations for updating the frequency penalties of each index, thus the overall procedure performs in linear time. Furthermore, as shown in (6), when evaluating a given move during the diversification the frequency penalties can be obtained in constant time by a simple lookup operation on either the `active` or the `partial` array. This ensures that the diversification procedure does not slow down the overall search process, as both search and diversification iterations have the same asymptotic linear runtime.

Phase Transition A search phase ends when no new best solution was found in the last αn iterations, where α is a model parameter we refer to as the *improvement threshold ratio*. We say the phase was *successful* if one or more global improvements were found and *unsuccessful* otherwise. After a search phase ends the best solution as well as its accompanying state of the tabu list is restored and a diversification phase is initiated. Note that, in order to prevent cycling, it is imperative to prevent the search from taking the same subsequent moves away from the restored solution as performed previously when visiting that solution. Furthermore, note that after one or more consecutive unsuccessful search phases the same best solution is restored multiple times, potentially requiring to block the subsequent moves performed each for each previous time the given solution was restored.

We implement two measures to prevent cycling in those scenarios. First, we further alter the objective by modifying the diversification intensity based on whether the previous search phase was successful or not. Specifically, after k consecutive unsuccessful search phases we set the diversification intensity γ to

$$\gamma = \theta^k \gamma_0,$$

where $\theta > 1$ is a model parameter, which we refer to as the *diversification scaling factor*. After a successful phase γ is then reset back to the base value γ_0. Secondly, we maintain a list of *explored moves*, which are moves that where previously performed after visiting the restored solution and are explicitly excluded by being marked tabu at the start of a diversification phase. This list includes the first τn moves performed after initially visiting the best solution during a search phase as well as the first τn moves performed in each diversification phase since the last successful search phase. τ is again a model parameter, which we call the *explored move ratio*. We note that during diversification the same aspiration criterion is used as during the search phases, that is a tabu move only gets accepted if it leads to a new best solution with respect to the original objective f.

The diversification phase ends after a fixed number of iteration, which we again scale linearly to the problem size n. Specifically, each diversification phase, we perform δn iterations, where δ is again a model parameter we refer to as the *diversification length ratio*. Afterwards, a new search phase starts with its initial solution being the last solution of the diversification phase. In case, that all moves are marked tabu due to the explored moves being marked tabu at the start of the diversification phase, and no move fulfills the aspiration criterion, the algorithm terminates. Otherwise the algorithm terminates after a set time limit.

Results

In this section we give an overview over our experiments as well as present our results on the performance and solution quality obtained by each of the three steps in our approach.

To ensure a fair analysis of our approach in the context of providing a general purpose heuristic for solving QUBO, we perform our experiments on three classes of problem instances, each with their own inherent problem structure. The instances named 'bqpx.x' and introduced by Beasley

[13] are randomly generated. More precisely, all non-zero entries are integers uniformly drawn from $[-100, 100]$ with an expected density of 10% and their sizes vary from $n = 50$ up to $n = 2500$. The second set of instances introduced by Palubeckis [14] and thus named 'px.x' are instances with densities of up to 100% and contain large instances with sizes between $n = 3000$ and $n = 7000$. Ultimately, the third class consists of instances named 'Gxx' reduced from the Maximum Cut problem. Here, we find a great variety of sizes, ranging from 800 to 1400, and rather low densities, ranging from around 6% to less than 0.05%.

All instances where retrieved using an freely available script [15] which also provides upper bounds from the literature that we base our analysis on. We note that we were not able to verify the optimality of these bounds and in fact found improvements on two instances using our method as well as the freely available QUBO solver Qbsolv [16]. Table 5 gives an overview of the types and their characteristics of all instances considered as part of our experiments as well as their reported bounds.

All experiments in this section where implemented in the Rust programming language and performed single-threaded on a AMD Ryzen 5 2600 (3.4 GHz) processor with a memory of 16GB. The implementation is also freely available on GitHub [17].

Instance	**Size**	**Density** [%]	**Size Reduction** [%]	**Time** [ms]
G55	5000	0.12	0.62	398
G56	5000	0.12	0.62	400
G60	7000	0.08	0.61	783
G61	7000	0.08	0.61	774
G70	10000	0.03	13.54	1639
bqp100.1	100	9.39	4.00	0
bqp100.10	100	9.82	3.00	0
bqp100.2	100	9.76	1.00	0
bqp100.4	100	9.62	1.00	0
bqp100.5	100	9.30	4.00	0
bqp100.6	100	10.38	5.00	0
bqp100.7	100	9.47	2.00	0
bqp100.8	100	9.91	3.00	0
bqp100.9	100	10.13	2.00	0
bqp50.1	50	8.76	10.00	0
bqp50.10	50	8.76	50.00	0
bqp50.2	50	9.72	26.00	0
bqp50.3	50	10.72	36.00	0
bqp50.4	50	9.04	66.00	0
bqp50.5	50	10.76	62.00	0
bqp50.6	50	8.64	32.00	0
bqp50.7	50	10.04	74.00	0
bqp50.8	50	11.36	28.00	0
bqp50.9	50	10.04	42.00	0

Table 1: *Overview of the problem size reductions achieved by the preprocessing procedure. 84% of instances for which no reduction was achieved are omitted.*

Preprocessing

The goal behind our preprocessing procedure was to reduce the size of the original problem to facilitate a speedup for the tabu search iterations. To analyze the effectiveness of the simple rules we apply, we perform the preprocessing routine on all 151 instances in our data set and evaluate the relative reduction in problem size achieved.

We observe that the preprocessing rules are largely ineffective in reducing the problem size, where for 84% of our instances we fail in achieving any reduction. Table 1 displays the size reductions for the remaining instances. We observe that, with the exception of instance G70, the only meaningful reductions in problem size of over 5% can be achieved on the smallest instances of size $n = 50$. However we note, that performing the preprocessing procedure does not lead to a significant execution overhead. Even considering the largest instance G70 with $n = 10000$, where the method was able to make significant reductions to the size of the problem, the procedure finished in less than 1.8 seconds. Furthermore, we can not identify a clear correlation between the density and the achieved reduction in problem size, as even though G70 has the smallest density in our data set, other instances of similar density did not allow for any size reduction.

Start heuristic

For the start heuristic we analyze the solution quality obtained with respect to the upper bounds from the literature as well as the runtime. To that end we perform the start heuristic on the partial dataset also used for evaluating the Tabu Search and evaluate the achieved relative gap with respect to the bounds from the literature. Additionally, we compare the performances of the three hint vector choices discussed previously. Our results are presented in table 2.

We observe that the simple rounding scheme is able to find good solutions with respect to the provided bounds for a majority of instances, where on average the rounding heuristic achieves a gap of 1.6%. However we observe a noticeably higher gap for the maximum cut instances compared to the randomly generated instances, where we obtain an average gaps of 3.4% and 0.8% resepctively. Finally we note that the method performs efficently, where we report a maximum computation time of less than 1.7 seconds on the largest instance p7000.2 of size $n = 7000$.

We observe a positive impact of the biased choices for the hint vector in our heuristic, where for 80% of instances the objective obtained by the constant hint vector is dominated by one of the objectives of the biased hint vectors. However no clear trend can be identified on which biased hint vector performs best, which supports our decision to perform the start heuristic for all three hint vectors and selecting the best solution.

Tabu search

In the experiments on our Tabu search method we first analyze the impact of the model parameters on the quality of the solution, for which we perform a grid search over empirically chosen test values on a small dataset of instances ranging in size $n = 500$ to $n = 1000$. Then we run the tabu search with the best performing model parameters on a different randomly selected subset of our instances, which include large instances from the 3 instance classes in our dataset.

Model Parameter Tuning For our parameter tuning we opted for a simple grid search, where we evaluate all combinations over empirically chosen test values on a small subset of instances. Table 3 shows the empirical values tested as well as the best performing parameter combination, which was then applied in the rest of our experiments.

Instance	**Size**	**Objective for Hint Vector**			**Bound**	**Gap** [%]	**Time** [ms]
		x_{const}	x_{entries}	x_{sum}			
bqp500.1	500	-115075	-114175	**-115109**	-116586	1.27	3
bqp500.2	500	-127538	**-127650**	-127477	-128223	0.45	3
bqp500.3	500	**-130782**	-129983	-129943	-130812	0.02	3
bqp500.4	500	-128948	**-129242**	-128532	-130097	0.66	3
bqp1000.1	1000	-368340	-367506	**-369305**	-371438	0.57	18
bqp1000.2	1000	-350022	-351446	**-351978**	-354932	0.83	23
bqp1000.3	1000	-368309	**-368423**	-368136	-371236	0.76	19
bqp1000.4	1000	-366644	-366997	**-368254**	-370675	0.65	23
bqp2500.1	2500	-1500929	-1499508	**-1501249**	-1515944	0.97	259
bqp2500.2	2500	-1457404	-1458656	**-1459447**	-1471392	0.81	162
G1	800	-11207	**-11290**	-11277	-11624	2.87	9
G2	800	-11297	**-11310**	-11280	-11620	2.67	9
G3	800	-11314	**-11362**	-11287	-11622	2.24	9
G4	800	**-11294**	-11264	-11251	-11646	3.02	9
G22	2000	-12747	**-12780**	-12770	-13359	4.33	97
G23	2000	-12820	**-12887**	-12856	-13344	3.42	124
G55	5000	**-9755**	-9702	-9601	-10299	5.28	695
p3000.1	3000	-3870867	-3873478	**-3884442**	-3931583	1.20	220
p3000.2	3000	**-5150848**	-5149709	-5140387	-5193073	0.81	290
p3000.3	3000	-5038701	**-5054359**	-5052322	-5111533	1.12	212
p3000.4	3000	-5696427	-5695760	**-5724570**	-5761822	0.65	261
p7000.2	7000	-18064373	**-18073453**	-18071477	-18249948	0.97	1656
p7000.3	7000	**-20180981**	-20175286	-20171702	-20446407	1.30	1446

Table 2: *Results of the greedy rounding heuristic. The table shows the objectives obtained for the three choices of the hint vector, where $x_{const}, x_{entries}$ and x_{sum} correspond to the constant hint vector, the hint vector biased by number of negative entries and the hint vector biased by sums of negative entries, respectively. The best-performing hint vector is highlighted for each instance. Furthermore, the gap of the best-performing hint vector compared to the upper bounds from the literature is shown.*

Performance Analyis Using the best model parameters found we apply the Tabu Search method to all instances considered in the testing of the start heuristic. Here we evaluate the improvements made over the solution of the rounding heuristic, which was set as the initial solution for the Tabu search. Furthermore we compare the performance to the freely available QUBO solver QBsolv. The results are given in table 4.

We observe that the Tabu search was successful in further reducing the gap obtained by the rounding heuristic. In particular for the bqp instances we were able to reduce the gap to near zero and improve on the provided bound for instance bqp500.2. Noticeably the search terminated for all instances before the set time limit was reached, indicating that the alternate stopping criterion in which all moves are marked tabu was reached in all instances. Therefore, we assume that our simple parameter tuning was not sufficient in finding suitable model parameters, as the current parameters lead the search to terminate prematurely. We suspect that further improvements on the objectives could be achieved by choosing a more suitable parameter combination.

When comparing our results to the Qbsolv solver be observe similar results, with Qbsolv performing slightly better on most instances. We note that these results might not be a fair comparison as we enforced a time limit for Qbsolv on each instance equal to the execution time of our approach.

Parameter	**Notation**	**Empirical Test Values**	**Best**
Tenure Ratio	λ	0.05 0.2 0.5	**0.05**
Diversification Length Ratio	δ	0.05 0.2 0.5	**0.5**
Diversification Base Factor	μ	0.1 0.25 0.5	**0.5**
Diversification Scaling Factor	θ	1.1 1.25 1.5	**1.25**
Improvement Threshold Ratio	α	1.0 5.0 10.0	**1**
Explored Move Ratio	τ	0.005 0.01	**0.01**

Table 3: *Parameter tuning results*

Given a more extensive time limit it is plausible that Qbsolv would have achieved significant improvements. However we observe that Qbsolv is much more resource extensive than our approach. For instance p7000.3 Qbsolv failed to produce a solution, due to insufficient memory.

Furthermore we observe that, similar to the rounding heuristic, the worst bounds were achieved for the maximum cut instances, which could either be reasoned by our method not being suited for maximum cut instances or a higher quality of the provided bounds compared to the randomly generated instances.

Conclusion

We presented a three-step heuristic approach for solving quadratic unconstrained binary optimization problems. After performing an initial preprocessing we applied a greedy rounding heuristic, obtaining a start solution which was then refined by a diversification-driven Tabu search method. Our approach was evaluated on a diverse set of instances of varying sizes and densities, originating both from transformed maximum cut problems as well as randomly generated instances.

While the simple rules implemented in our preprocessing routine did not lead to noticeable reductions in problem size, using the greedy rounding heuristic combined with the tabu search we were able to find solutions which are competitive with upper bounds reported in the literature. Furthermore our approach proved to be resource- and time-efficient and was able to find good solutions for large instances without exceeding the resource constraints imposed by the mediocre hardware used. However, we note that in order for out approach to become competitive with modern QUBO solvers, additional refinements would be needed. Nonetheless, we believe that our approach and specifically the diversification procedure we developed shows potential.

We conclude with three potential improvements which we aim to investigate in future research. Firstly, one important aspect of QUBO problems that we did not exploit is sparsity. Note, that the matrix of the problem as well as all memory structures used throughout the approach were implemented using dense data structures. We expect that modifying our approach to utilize sparse data structures instead will lead to significant performance increases, given the sparse nature of most instances. Secondly we aim to improve on the non-successful preprocessing routine by implementing more sophisticated preprocessing rules. Finally we aim to improve the tabu search by introducing an adaptive neighborhood, where the 1-flip neighborhood is enriched by a set of 2-flips, i.e. moves that flip the binary values at two given indices. Here we aim to restrict the set of 2-flips added in an adaptive way, to not inflate the neighborhood size by much. One idea would be to add 2-flips for pairs of *weakly coupled* variables, i.e. variables that are observed to likely be always equal or always unequal in an optimal solution. We aim to identify these pairs of variables as part of our improved preprocessing.

Instance	**Size**	**Computed Objective**	**Bound**	**Gap** [%]	**Qbsolv**	**Time** [ms]
bqp500.1	500	-116586	-116586	0.00	-116586	1004
bqp500.2	500	-128251	-128223	-0.02	-128339	816
bqp500.3	500	-130812	-130812	0.00	-130812	1118
bqp500.4	500	-130067	-130097	0.02	-130097	882
bqp1000.1	1000	-371237	-371438	0.05	-371438	6353
bqp1000.2	1000	-354932	-354932	0.00	-354932	5712
bqp1000.3	1000	-371236	-371236	0.00	-371236	7049
bqp1000.4	1000	-370653	-370675	0.01	-370675	4297
bqp2500.1	2500	-1515015	-1515944	0.06	-1515944	104224
bqp2500.2	2500	-1469282	-1471392	0.14	-1471034	102645
G1	800	-11563	-11624	0.52	-11574	1250
G2	800	-11589	-11620	0.27	-11600	1540
G3	800	-11582	-11622	0.34	-11596	1405
G4	800	-11612	-11646	0.29	-11613	1012
G22	2000	-13274	-13359	0.64	-13306	12799
G23	2000	-13249	-13344	0.71	-13296	15392
G55	5000	-10181	-10299	1.15	-10141	249787
p3000.1	3000	-3921778	-3931583	0.25	-3931583	86200
p3000.2	3000	-5189072	-5193073	0.08	-5193229	137100
p3000.3	3000	-5109041	-5111533	0.05	-5114396	123912
p3000.4	3000	-5753426	-5761822	0.15	-5761822	199268
p7000.2	7000	-18221036	-18249948	0.16	-18255248	739156
p7000.3	7000	-20419936	-20446407	0.13	-	863144

Table 4: *Results of the tabu search. The achieved objectives are compared with the provided bounds and the objectives obtained by the solver Qbsolv.*

References

[1] Abraham P Punnen. *The quadratic unconstrained binary optimization problem*. Springer, 2022.

[2] Ksenia Bestuzheva et al. “Enabling Research through the SCIP Optimization Suite 8.0”. In: *ACM Trans. Math. Softw.* 49.2 (June 2023). ISSN: 0098-3500. DOI: 10.1145/3585516. URL: https://doi.org/10.1145/3585516.

[3] Duan Li, XL Sun, and CL Liu. “An exact solution method for unconstrained quadratic 0–1 programming: a geometric approach”. In: *Journal of Global Optimization* 52.4 (2012), pp. 797–829.

[4] Christoph Helmberg and Franz Rendl. “Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes”. In: *Mathematical programming* 82 (1998), pp. 291–315.

[5] Gary Kochenberger et al. “The unconstrained binary quadratic programming problem: A survey”. In: *Journal of Combinatorial Optimization* 28 (July 2014). DOI: 10.1007/s10878-014-9734-0.

[6] Talal M Alkhamis, Merza Hasan, and Mohamed A Ahmed. “Simulated annealing for the unconstrained quadratic pseudo-Boolean function”. In: *European journal of operational research* 108.3 (1998), pp. 641–652.

[7] Fred Glover, Zhipeng Lü, and Jin-Kao Hao. "Diversification-driven tabu search for unconstrained binary quadratic problems". en. In: *4OR* 8.3 (Oct. 2010), pp. 239–253. ISSN: 1619-4500, 1614-2411. DOI: `10.1007/s10288-009-0115-y`. URL: `http://link.springer.com/10.1007/s10288-009-0115-y` (visited on 01/14/2024).

[8] Mohammad M Amini. "A scatter search approach to unconstrained quadratic binary programs". In: *New ideas in optimization* (1999), pp. 317–329.

[9] Iain Dunning, Swati Gupta, and John Silberholz. "What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO". In: *INFORMS Journal on Computing* 30.3 (2018), pp. 608–624.

[10] Mark Lewis and Fred Glover. "Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis". In: *Networks* 70.2 (2017), pp. 79–97.

[11] Endre Boros, Peter L Hammer, and Gabriel Tavares. *Preprocessing of unconstrained quadratic binary optimization*. Tech. rep. Technical Report RRR 10-2006, RUTCOR, 2006.

[12] Fred Glover and Manuel Laguna. "Tabu Search". In: *Handbook of Combinatorial Optimization: Volume1–3*. Ed. by Ding-Zhu Du and Panos M. Pardalos. Boston, MA: Springer US, 1998, pp. 2093–2229. ISBN: 978-1-4613-0303-9. DOI: `10.1007/978-1-4613-0303-9_33`. URL: `https://doi.org/10.1007/978-1-4613-0303-9_33`.

[13] John E Beasley. *Heuristic algorithms for the unconstrained binary quadratic programming problem*. Tech. rep. Working Paper, The Management School, Imperial College, London, England, 1998.

[14] Gintaras Palubeckis. "Multistart tabu search strategies for the unconstrained binary quadratic optimization problem". In: *Annals of Operations Research* 131 (2004), pp. 259–282.

[15] GitHub. *qubo-benchmark-instances*. https://github.com/rliang/qubo-benchmark-instances. 2021.

[16] GitHub. *Qbsolv*. https://github.com/dwavesystems/qbsolv. 2021.

[17] GitHub. *QuboOptimizer*. https://github.com/janerikhein/QuboOptimizer. 2024.

Instance	Size	Density	Bound	Type	Instance	Size	Density	Bound	Type
bqp50.1	50	8.76	-2098	randomly generated	bqp50.2	50	9.72	-3702	randomly generated
bqp50.3	50	10.72	-4626	randomly generated	bqp50.4	50	9.04	-3544	randomly generated
bqp50.5	50	10.76	-4012	randomly generated	bqp50.6	50	8.64	-3693	randomly generated
bqp50.7	50	10.04	-4520	randomly generated	bqp50.8	50	11.36	-4216	randomly generated
bqp50.9	50	10.04	-3780	randomly generated	bqp50.10	50	8.76	-3507	randomly generated
bqp100.1	100	9.39	-7970	randomly generated	bqp100.2	100	9.76	-11036	randomly generated
bqp100.3	100	9.95	-12723	randomly generated	bqp100.4	100	9.62	-10368	randomly generated
bqp100.5	100	9.3	-9083	randomly generated	bqp100.6	100	10.38	-10210	randomly generated
bqp100.7	100	9.47	-10125	randomly generated	bqp100.8	100	9.91	-11435	randomly generated
bqp100.9	100	10.13	-11455	randomly generated	bqp100.10	100	9.82	-12565	randomly generated
bqp250.1	250	9.93	-45607	randomly generated	bqp250.2	250	9.75	-44810	randomly generated
bqp250.3	250	9.85	-49037	randomly generated	bqp250.4	250	10.11	-41274	randomly generated
bqp250.5	250	10.0	-47961	randomly generated	bqp250.6	250	10.22	-41014	randomly generated
bqp250.7	250	9.92	-46757	randomly generated	bqp250.8	250	9.69	-35726	randomly generated
bqp250.9	250	10.1	-48916	randomly generated	bqp250.10	250	9.78	-40442	randomly generated
bqp500.1	500	9.92	-116586	randomly generated	bqp500.2	500	9.84	-128223	randomly generated
bqp500.3	500	10.03	-130812	randomly generated	bqp500.4	500	9.84	-130097	randomly generated
bqp500.5	500	9.88	-125487	randomly generated	bqp500.6	500	9.87	-121772	randomly generated
bqp500.7	500	9.95	-122201	randomly generated	bqp500.8	500	9.84	-123559	randomly generated
bqp500.9	500	9.9	-120798	randomly generated	bqp500.10	500	9.94	-130619	randomly generated
bqp1000.1	1000	9.9	-371438	randomly generated	bqp1000.2	1000	9.9	-354932	randomly generated
bqp1000.3	1000	9.95	-371236	randomly generated	bqp1000.4	1000	9.93	-370675	randomly generated
bqp1000.5	1000	9.91	-352760	randomly generated	bqp1000.6	1000	9.98	-359629	randomly generated
bqp1000.7	1000	9.89	-371193	randomly generated	bqp1000.8	1000	9.9	-351994	randomly generated
bqp1000.9	1000	9.91	-349337	randomly generated	bqp1000.10	1000	9.82	-351415	randomly generated
bqp2500.1	2500	9.94	-1515944	randomly generated	bqp2500.2	2500	9.9	-1471392	randomly generated
bqp2500.3	2500	9.89	-1414192	randomly generated	bqp2500.4	2500	9.9	-1507701	randomly generated
bqp2500.5	2500	9.9	-1491816	randomly generated	bqp2500.6	2500	9.88	-1469162	randomly generated
bqp2500.7	2500	9.92	-1479040	randomly generated	bqp2500.8	2500	9.89	-1484199	randomly generated
bqp2500.9	2500	9.92	-1482413	randomly generated	bqp2500.10	2500	9.9	-1483355	randomly generated
G1	800	6.12	-11624	Maximum Cut	G2	800	6.12	-11620	Maximum Cut
G3	800	6.12	-11622	Maximum Cut	G4	800	6.12	-11646	Maximum Cut
G5	800	6.12	-11631	Maximum Cut	G6	800	6.12	-2178	Maximum Cut
G7	800	6.12	-2006	Maximum Cut	G8	800	6.12	-2005	Maximum Cut
G9	800	6.12	-2054	Maximum Cut	G10	800	6.12	-2000	Maximum Cut
G11	800	0.62	-564	Maximum Cut	G12	800	0.62	-556	Maximum Cut
G13	800	0.62	-582	Maximum Cut	G14	800	1.59	-3064	Maximum Cut
G15	800	1.58	-3050	Maximum Cut	G16	800	1.58	-3052	Maximum Cut
G17	800	1.58	-3047	Maximum Cut	G18	800	1.59	-992	Maximum Cut
G19	800	1.58	-906	Maximum Cut	G20	800	1.58	-941	Maximum Cut
G21	800	1.58	-931	Maximum Cut	G22	2000	1.05	-13359	Maximum Cut
G23	2000	1.05	-13344	Maximum Cut	G24	2000	1.05	-13337	Maximum Cut
G25	2000	1.05	-13340	Maximum Cut	G26	2000	1.05	-13328	Maximum Cut
G27	2000	1.05	-3341	Maximum Cut	G28	2000	1.05	-3298	Maximum Cut
G29	2000	1.05	-3405	Maximum Cut	G30	2000	1.05	-3413	Maximum Cut
G31	2000	1.05	-3310	Maximum Cut	G32	2000	0.25	-1410	Maximum Cut
G33	2000	0.25	-1382	Maximum Cut	G34	2000	0.25	-1384	Maximum Cut
G35	2000	0.64	-7687	Maximum Cut	G36	2000	0.64	-7680	Maximum Cut
G37	2000	0.64	-7691	Maximum Cut	G38	2000	0.64	-7688	Maximum Cut
G39	2000	0.64	-2408	Maximum Cut	G40	2000	0.64	-2400	Maximum Cut
G41	2000	0.64	-2405	Maximum Cut	G42	2000	0.64	-2481	Maximum Cut
G43	1000	2.1	-6660	Maximum Cut	G44	1000	2.1	-6650	Maximum Cut
G45	1000	2.1	-6654	Maximum Cut	G46	1000	2.1	-6649	Maximum Cut
G47	1000	2.1	-6665	Maximum Cut	G48	3000	0.17	-6000	Maximum Cut
G49	3000	0.17	-6000	Maximum Cut	G50	3000	0.17	-5880	Maximum Cut
G51	1000	1.28	-3848	Maximum Cut	G52	1000	1.28	-3851	Maximum Cut
G53	1000	1.28	-3850	Maximum Cut	G54	1000	1.28	-3852	Maximum Cut
G55	5000	0.12	-10299	Maximum Cut	G56	5000	0.12	-4017	Maximum Cut
G57	5000	0.1	-3494	Maximum Cut	G58	5000	0.26	-19293	Maximum Cut
G59	5000	0.26	-6087	Maximum Cut	G60	7000	0.08	-14190	Maximum Cut
G61	7000	0.08	-5798	Maximum Cut	G62	7000	0.07	-4870	Maximum Cut
G63	7000	0.18	-27045	Maximum Cut	G64	7000	0.18	-8751	Maximum Cut
G65	8000	0.06	-5562	Maximum Cut	G66	9000	0.06	-6364	Maximum Cut
G67	10000	0.05	-6950	Maximum Cut	G70	10000	0.03	-9591	Maximum Cut
G72	10000	0.05	-7006	Maximum Cut	G77	14000	0.04	-9938	Maximum Cut
G81	20000	0.03	-14048	Maximum Cut	p3000.1	3000	50	-3931583	randomly generated
p3000.2	3000	80	-5193073	randomly generated	p3000.3	3000	80	-5111533	randomly generated
p3000.4	3000	100	-5761822	randomly generated	p3000.5	3000	100	-5675625	randomly generated
p4000.1	4000	50	-6181830	randomly generated	p4000.2	4000	80	-7801355	randomly generated
p4000.3	4000	80	-7741685	randomly generated	p4000.4	4000	100	-8711822	randomly generated
p4000.5	4000	100	-8908979	randomly generated	p5000.1	5000	50	-8559680	randomly generated
p5000.2	5000	80	-10836019	randomly generated	p5000.3	5000	80	-10489137	randomly generated
p5000.4	5000	100	-12252318	randomly generated	p5000.5	5000	100	-12731803	randomly generated
p6000.1	6000	50	-11384976	randomly generated	p6000.2	6000	80	-14333855	randomly generated
p6000.3	6000	100	-16132915	randomly generated	p7000.1	7000	50	-14478676	randomly generated
p7000.2	7000	80	-18249948	randomly generated	p7000.3	7000	100	-20446407	randomly generated

Table 5: *Overview over all instances considered.*