Heuristic Solvers for QUBO

Michael Adipoetra - 408045
Gennesaret Kharistio Tjusila - 407687
Francisco Jose Manjon Cabeza Garcia - 369293
Arya Prasetya - 473184

January 2023

Given a lower triangular matrix $Q \in \mathbb{Q}^{n\times n}$, the quadratic unconstrained binary optimisation ($QUBO$) problem is to solve

$$c^* := \max_{x\in\{0,1\}^n} x^T Q x. \tag{1}$$

We call c^* the optimal value and a binary vector x an optimal solution if $x^T Q x = c^*$. In the literature QUBOs are also called unconstrained binary quadratic programs (UBQP). QUBO can generalized to rational matrices $Q \in \mathbb{Q}$ by first applying the transformation $q_{ij} = q_{ij} + q_{ji}$ and $q_{ji} = 0$ for all $i, j \in \{1, \dots, n\}$ with $i > j$. We can also transform QUBOs into a minimisation problem by multiplying the entries of Q with minus one.

Although QUBO are known to be NP-hard in general (Glover et al., 2010), they can be used for a wide variety of applications. Many combinatorial optimisation problems such as machine scheduling (Alidaee et al., 1994) and network flow problems (Ivănescu, 1965) can be formulated as QUBO problems. Furthermore, its equivalence with the spin glass model/Ising Hamiltonian sparked interest towards QUBO in the field of physics (Lucas, 2014).

Methods to solve QUBO problems can be classified in two groups: exact methods, which try to find an optimum solution, and heuristic methods, which try to find solutions (not necessarily optimal) with high objective value. While exact methods are generally favorable in smaller problems, existing exact solvers have generally been unsuccessful in solving large instances (Glover and Kochenberger, 2018). On the contrary, heuristic solvers have had success in finding high-quality solutions.

This project aims to give a comparative analysis of three multistart heuristics from the literature: diversification driven tabu search (DDTS) (Glover et al., 2010), multistart tabu search (MST2) (Palubeckis, 2004), and simulated annealing (SA) (Katayama and Narihisa, 2001). We use instances from QPLIB (Furini et al., 2018) as test bed. To collect the data for this report we create our own implementations of the algorithms and run them on QPLIB instances.

This report is separated into 5 sections. In the first section, two single start heuristic search methods are introduced. These single start search methods will form the basis of the methods we develop further down the line. In the second section, 3 different multi-start heuristic algorithms that we implement (DDTS, MST2, SA) are discussed. In the next section, the experiment setup and instances are described. In section 4, the performance of implemented algorithms are discussed. In the last section, some conclusions regarding behaviors of the 3 different heuristics on different instances are brought forth.

1 Heuristic search methods

We first introduce two single start heuristic search methods that are predominantly used in solving QUBO. Variants of these search methods serve as the backbone on the multi start heuristic algorithms we implemented.

1.1 Local search

A very simple way to solve a QUBO problem is to use local search. We start with an arbitrary solution and continuously update to solutions with higher objective value by evaluating its neighbourhood.

The neighbourhood of a solution can be evaluated by flipping one (1-opt) or more (k-opt) variables in the current solution. Throughout this project, we use the 1-opt neighbourhood. Therefore, we only have a maximum of n neighbours, i.e if $x_0 = (0, 1, 0)$ its neighbors will be $X_{nbhd} = \{(1, 1, 0), (0, 0, 0), (0, 1, 1)\}$. This is a convenient choice as evaluating the objective value of a solution is in $\mathcal{O}(n^2)$. By adopting 1-opt, we were able

to implement a speed-up method presented by Glover et al. (2010), in which the change in objective value of flipping variable $i \in \{1, \ldots, n\}$ for a given lower triangular matrix $Q \in \mathbb{Q}_{\geq 0}^{n \times n}$ can be calculated with

$$\Delta x_i = (1 - 2x_i) \left(row\_value(i) + column\_value(i) + q_{ii} \right), \tag{2}$$

where

$$row\_value(i) = \sum_{j=1}^{i-1} Q_{ij} x_j \quad \text{and} \quad column\_value(i) = \sum_{j=i+1}^{n} Q_{ji} x_j. \tag{3}$$

The benefit of this formulation is that $row\_value(i)$ and $column\_value(i)$ only needs to be computed at the beginning of the search. Throughout the searching process, if we decided to flip a variable x_i we can update these values by the following rule.

$$\begin{aligned} column\_value(j) &= column\_value(j) + (1 - 2x_i) q_{ij} && \text{for all } j < i, \\ row\_value(j) &= row\_value(j) + (1 - 2x_i) q_{ji} && \text{for all } j > i. \end{aligned}$$

There are two main updating rules for the local search. One can either update to the best improvement (steepest ascent) or look for the first improvement from a (randomly) ordered neighbour. While the latter may lead to more iterations, it is computationally less expensive per iteration. In this project, both update rules were explored.

Once a solution no longer has a value-improving neighbourhood, the local search terminates. It is important to note that despite this, we are located in a local optimum rather than a global one. For this reason, it is often not enough to use a simple local search to even obtain a close-to-optimal objective value.

1.2 Tabu search

To avoid the problem of stagnating in a local minimum, we introduce the tabu search first proposed by Glover and Laguna (1998). In this method, one allows a worsening move if there are no more value-improving moves in the neighbourhood. To ensure that the solution does not go back to the previous solution, a tabu list is used. After a variable has been flipped, it is banned from being flip again for a certain number of iteration. In our implementation this number is

$$\min(20, n/4) + random\_integer(1, 10)$$

where n is the number of variable in our QUBO instance. This ban is relaxed if at some point during our search we find that flipping a tabu variable give a strictly better solution than any previous solution explored so far.

With this, we have a relatively robust search method to finally tackle problems with local optima. Nevertheless, such a search may still be problematic when dealing with valleys surrounding a local optimum, as the tabu tenure might not be long enough to jump out from an oscillation.

2 Multistart methods

As seen with heuristic search methods, there is a chance that the best objective value stagnates in a search. One way to solve this problem is to consider multistart methods/strategies classified in (Martí et al., 2013).

The idea here is to start with variants of local or tabu search with a restart phase once the search terminates. During a restart, the resulting solution is modified in such a way that it will allow searches to progress closer to a global optimum.

We categorize the multistart method into memory-less and memory-based designs. Memory-less design means we start the new iteration without keeping any information from the previous iteration. On the other hand, memory-based designs keep some of those previous information, which may then be used in the new iteration. In this project, we implemented three different multistart methods: simulated annealing, multistart tabu search and diversification-driven tabu search.

2.1 Simulated Annealing

An alternative to tabu search, instead of allowing worse moves only when there are no improvement left to do, this new method allow the worse move under a probability $p_i = e^{\Delta x_i / T}$.

As the name "annealing" suggests, one starts with a high temperature as one perform a local search (with random updating rule) with higher likelihood of taking worse move. After a few local search, the temperature is cooled down to stabilize the move to a local optimum.

Under the multistart schema, we perform a reannealing, where we restart the annealing process under the running solution for multiple times. Such a method is called simulated annealing-based heuristic, which is proposed by Katayama and Narihisa (2001).

The search phase in this heuristics is guaranteed to terminate as a term limit $TermCount$ is defined to terminate the algorithm if the local search has not given a value-improving move $TermCount$ many times.

This algorithm is a memory-less design, because whenever one restarts the annealing process, we do not use any memory from before. Algorithm 4 in the appendix shows the pseudo-code of the simulated annealing-based heuristic.

In our simulated annealing implementation, the following constants are used

constant	**value**
T_{init}	$0.1n$
$SACount$	1000
$StartTFactor$	1
$TermCount$	10
$TFactor$	0.99

2.2 Multistart tabu search

The second algorithm that was implemented is the multistart tabu search proposed by Palubeckis (2004). In this algorithm, one uses a scoring of the current running solution to perform a restart. In this paper, we implemented the variant $MSTS2$, which incoporate variable scoring and a steep ascent method. The paper use a symmetric matrix instead of a lower triangular matrix in Equation (1). But one can still implement the algorithm by changing some indexing.

Initially one peforms a tabu search on the initial solution, leading to a tabu solution. The tabu solution is then used to generate a new initial solution based on a variable selection by a scoring system and steepest ascent of the variable selection. The score function for the multistart tabu search algorithm in Palubeckis (2004) is defined as:

$$Score(x_i) := \begin{cases} 1 - \frac{\Delta x_i}{\Delta_{min}} & \text{if} \quad \Delta x_i \leq 0,\ \Delta_{min} < 0, \\ 0 & \text{if} \quad \Delta x_i = \Delta_{min} = 0, \\ 1 + \lambda \frac{\Delta x_i}{\Delta_{max}} & \text{if} \quad \Delta x_i > 0. \end{cases} \tag{4}$$

where as before, Δx_i calculates the change of objective value if we flip the variable.

We use this score function for every variable in the tabu solutions. After each variable has a score, one assigns a probability for each variable based on their score, where the probability function is

$$p_i = \frac{Score(x_i)}{\sum_{j \in I} Score(x_j)}. \tag{5}$$

As we can see from the score function, if Δx_i is positive, the variable will have a high score proportional to λ. If Δx_i is negative, then the objective value will be degenerate if we flip the variable. Therefore, we assign it a low score.

Then, we pick subset variables based on their probability. The amount of variable, that we picked, is based on a predefined value $max(\alpha n, 10)$. Once selecting the variables is done, a steepest ascent method in these selected variable is deployed.

The next step is to use steepest ascent to find the direction. Assume we picked k variable, now we want to solve the original QUBO problem only for these variables, meaning we set the non-subset variables as 0. The idea behind it is to get the steepest ascent from the centre of the k-dimensional hyper-cube $(0.5, ..., 0.5)$ to some of its vertex ($0 - 1$ vector). Each step of a climb, fixes the variable to 1 or 0.

For each variable we compute the difference in objective value between setting the variable to 1 and to 0. If the difference is bigger than 0, the corresponding variable is will be fixed to 1, and to 0 else. Then, we pick the biggest difference in absolute value. We accordingly change the variable with the biggest difference to 1 or 0. After the variable is fixed, we proceed to do the same with the next selected variable.

As a result, one will get the direction of the steepest ascent. Now, we change the tabu solution according to the direction of the steepest ascent. This will be a new initial solution for the next tabu search.

Algorithms 2 and 3 in appendix show the pseudo-code of the multistart tabu search algorithm and the steepest ascent procedure.

In our multistart tabu search implementation, the following constants are used

constant	**value**
α	0.4
λ	5000

2.3 Diversification-driven tabu search

Up until now, the algorithm we implement does not follow memory-based design. This last algorithm we implement is diversification-driven tabu search proposed by Glover et al. (2010). The algorithm attempts to diversify the search by applying perturbations in between successive tabu searches. The hope is that by applying this perturbation, we jump to previously unexplored areas of the search space.

The perturbation operation takes into account two things. The first is how often a variable is flipped in the last iteration of tabu search. This value will be set into $FlipFreq$ list. Variable that is flipped less will be more likely to be perturbed since it is less explored in the previous tabu search run. For the second item, we keep a list of the up to top R solution found by the procedure so far. We then determine how strongly determined the variable is. If in a lot of our top solution a variable i is often assigned 1, then it is more likely that the value of the variable in an optimal is 1. $EliteFreq$ stores the number of times the i-th variable is set to 1 in $EliteSol$. While perturbing, we want to perturb variable that are less determined.

One achieves this by assigning to each variable the score function

$$Score(x_i) := \beta \left(1 - \frac{FlipFreq(i)}{max\_Freq}\right) + \frac{EliteFreq(i)\,(r - EliteFreq(i))}{r^2}, \tag{6}$$

where $max\_Freq = \max_{i=1,\dots,n} FlipFreq(i)$. The first term corresponds to the number of flips. The second term corresponds to a negative parabolic function, where if the $EliteFreq(i)$ is low, the score is also low, because as we mentioned, it is more likely that the value of this variable is 0. Same if $EliteFreq(i)$ is high, meaning it is often assigned to 1.

After all variables are assigned a score. They are sorted in decreasing order according to their score, and then γ many of them are assigned the value 0 or 1 according to the probability

$$P_i := \frac{i^{-\lambda} + uniform\_random(0, 0.05)}{z}, \tag{7}$$

where $\lambda \in \mathbb{R}_{>0}$ and z is a normalization constant.

Finally, we observe that though it is generally preferable to give high scoring variable a high chance to be perturbed. As scores get smaller the difference become less significant, thus we find it beneficial to just pick random samples from the low scoring variables rather than picking them in order of their scores. To achieve this we add a random value to the upper term of the probability.

$n/10$ variables was picked and perturbed. The perturbed solutions are finally used as starting point for a new tabu search procedure until a stop condition is met.

In our diversification-driven tabu search implementation, the following constants are used

constant	**value**
β	0.3
R	20
λ	1.2

3 Test instances

To test the performance with respect to the accuracy of the computed solutions compared to the actual optimal solution and with respect to running time, we have run the three algorithms described in the previous sections for a set of 23 instances from the QPLib instance library (Furini et al., 2018). Table 1 lists all the QUBO instances from the mentioned library together with the corresponding Q density, which shows the percentage of non-zero entries in the Q matrix.

Let Q^0 be the coefficient of quadratic terms and b^0 be the coefficient of the linear terms. We reformulate the objective function from QPLIB as follows

$$\begin{aligned} \frac{1}{2}x^T Q^0 x + b^0 x &= x^T \frac{1}{2} Q^0 x + x^T D x \\ &= x^T \left(\frac{1}{2} Q^0 + D\right) x \\ &= x^T Q x, \end{aligned} \tag{8}$$

where we change the coefficient of linear terms into a diagonal matrix in the first equation. Since Q^0 is a lower triangular matrix from QPLIB, Q is also a triangular matrix. Therefore, we can use our 3 implemented algorithms.

Instance Name	Q Matrix Density	Total Variables
QPLIB_3506	0.8	496
QPLIB_3565	1.4	276
QPLIB_3642	0.4	1035
QPLIB_3650	0.4	946
QPLIB_3693	0.3	1128
QPLIB_3705	1	378
QPLIB_3706	0.6	703
QPLIB_3738	0.9	435
QPLIB_3745	1.2	325
QPLIB_3822	0.5	861
QPLIB_3832	0.7	561
QPLIB_3838	0.5	780
QPLIB_3850	0.3	1225
QPLIB_3852	1.6	231
QPLIB_3877	0.6	630
QPLIB_5721	76.8	300
QPLIB_5725	1.7	343
QPLIB_5755	1	400
QPLIB_5875	78.9	200
QPLIB_5881	29.5	120
QPLIB_5882	78.1	150
QPLIB_5909	9.6	250
QPLIB_5922	9.8	500

Table 1: Test Instances Statistics Summary Furini et al. (2018).

The density ranges from 0.3% to almost 80%, and the number of variables from 120 to over 1200. We have a good mix of densities for low variable count instances. However, high variable count instances have a rather low density at or below 1%.

We run each instance 100 times with each of the three algorithms on the *High-Performance Computing* (HPC) from TU Berlin. The termination criteria are 10 minutes or 100 improvement cut-off, meaning if best solution is stuck for 100 iterations, we terminate the algorithm.

The cluster runs Linux operating system, specifically Alma Linux 8.7 with x86-64 bit architecture. We run the experiment using Intel Xeon E5-2630V4 2.2GHz on one single core. The memory limit is 16 GB. The core runs a single thread. The code was compiled using g++ 8.5.0.

4 Results

Table 2 shows the average gap in percent, i.e., the percetage difference between the best computed solution and the actual global optimum, and the optimum hit count, i.e., the number of times each algorithm finds an optimal solution with the same objective value as the global optimum.

Overall, there is not a single algorithm which outperforms the other two. However, we see some patterns. For example, for the instances with higher density the diversification-driven tabu search (*DDTS*) and multistart tabu search (*MSTS*) algorithms clearly outperform the simulated annealing (*SA*) algorithm by always finding a solution with objective value equal to the global optimum. The exception to this is instance 5721, which has a density of 76.8%, and for which neither of the three methods ever reached a global optimum. Nevertheless, the average gap in percent of the two tabu search methods is by a factor of 4 significantly lower than for the simulated annealing method for the 5721 instance.

For the instances with the highest number of variables (3642, 3693 and 3850), all with more than 1000 variables, the simulated annealing method achieves an average gap in percentage which is about 2.5 times better than for any of the tabu search methods. The SA method even hits the global optimum once for the 3693 and 3850 instances, while the tabu search methods do not hit it at all for any of the three mentioned instances.

In general, the DDTS algorithm outperforms the MSTS algorithm in terms of average gap in percentage. However, the opposite is true with respect to the number of times a solution with objective value equal to the global optimal objective value.

Figure 1 shows the percentage of total runs (100 runs times 23 instances) per algorithm which achieve an objective value less or equal to n times the second lowest gap in absolute term to the actual optimal objective value. For example, at $n = -1$, we see the percentage of total runs for which the corresponding algorithm reaches a solution with objective value 100% lower than the second best, i.e., the percentage of total runs

	Average Gap %			Opt. Hit Count		
	DDTS	**MSTS**	**SA**	**DDTS**	**MSTS**	**SA**
QPLIB_3506	**0.66**	1.25	0.71	14	1	**15**
QPLIB_3565	0.46	**0.04**	0.91	39	**95**	7
QPLIB_3642	2.85	7.62	**1.11**	0	0	0
QPLIB_3650	2.01	6.36	**0.84**	0	0	**2**
QPLIB_3693	0.20	0.63	**0.06**	0	0	**1**
QPLIB_3705	0.64	**0.09**	0.86	24	**82**	16
QPLIB_3706	1.74	4.61	**1.04**	1	0	**2**
QPLIB_3738	0.91	0.92	**0.76**	2	1	**5**
QPLIB_3745	0.03	**0.00**	0.51	95	**100**	40
QPLIB_3822	1.69	5.64	**0.84**	0	0	**2**
QPLIB_3832	1.17	3.18	**0.94**	2	0	**3**
QPLIB_3838	2.71	5.82	**1.35**	0	0	0
QPLIB_3850	2.59	8.04	**1.05**	0	0	**1**
QPLIB_3852	**0.00**	**0.00**	0.08	**100**	**100**	93
QPLIB_3877	0.79	2.05	**0.75**	**3**	0	2
QPLIB_5721	3.73	**3.00**	12.82	0	0	0
QPLIB_5725	0.59	**0.56**	16.16	**34**	6	0
QPLIB_5755	**3.15**	3.55	18.13	0	0	0
QPLIB_5875	**0.00**	**0.00**	1.65	**100**	**100**	4
QPLIB_5881	**0.00**	**0.00**	1.55	**100**	**100**	1
QPLIB_5882	**0.00**	**0.00**	1.92	**100**	**100**	5
QPLIB_5909	**0.00**	**0.00**	4.58	**100**	**100**	0
QPLIB_5922	**0.00**	**0.00**	1.13	**100**	**100**	0

Table 2: Average Gap in Percentage of Actual Optimal Objective Value and Number of Times the Actual Optimal Objective Value Was Hit.

for which the corresponding algorithm reaches a solution with objective value equal to the best over all three algorithms. At $n = 0$, we see the percentage of total runs for which the corresponding algorithm reaches a solution with objective value equal to the second lowest objective value over all three algorithms.

In figure 1, we can clearly see that the diversification-driven tabu search ($DDTS$) algorithm has a better performance profile than any of the other two algorithms. The simulated annealing (SA) algorithm has a lower percentage of runs within $n \leq 2.5$ times the second lowest gap to the actual optimal objective value. However, it surpasses the multistart tabu search ($MSTS$) algorithm for $n \geq 3$.

5 Conclusion

Although QUBO problems are in general NP-hard, their extended number of applications result in the need for finding methods which can at least provide solutions which are near enough to the optimum. We have analysed three algorithms with mixed results. In general, we can conclude that all three algorithms generate results which can be considered good enough on average. While the simulated annealing algorithm seems to perform bad for high density matrices, the tabu search algorithms deliver results at or very near to the global optimum. Overall, the lowest average gap in percentage within instances and across algorithms is very low. Hence, running the three algorithms in parallel or at least one of them several times might be a strategy good enough to find solutions with average gap below 5%, which in many applications could be considered good enough solutions. In case only one algorithm had to be chosen based on the results presented here, the diversification-driven tabu search algorithm would be our choice, due to the higher percentage or runs within a given n factor of the second best gap.

Nevertheless, we would like to remark that the set of instances analysed is only 23. Moreover, the number of variables was at most 1225. Hence, further benchmark analyses for other instances or for instances specific for the actual application of the methods should be performed, before applying these three algorithms for some particular QUBO problem or structure.

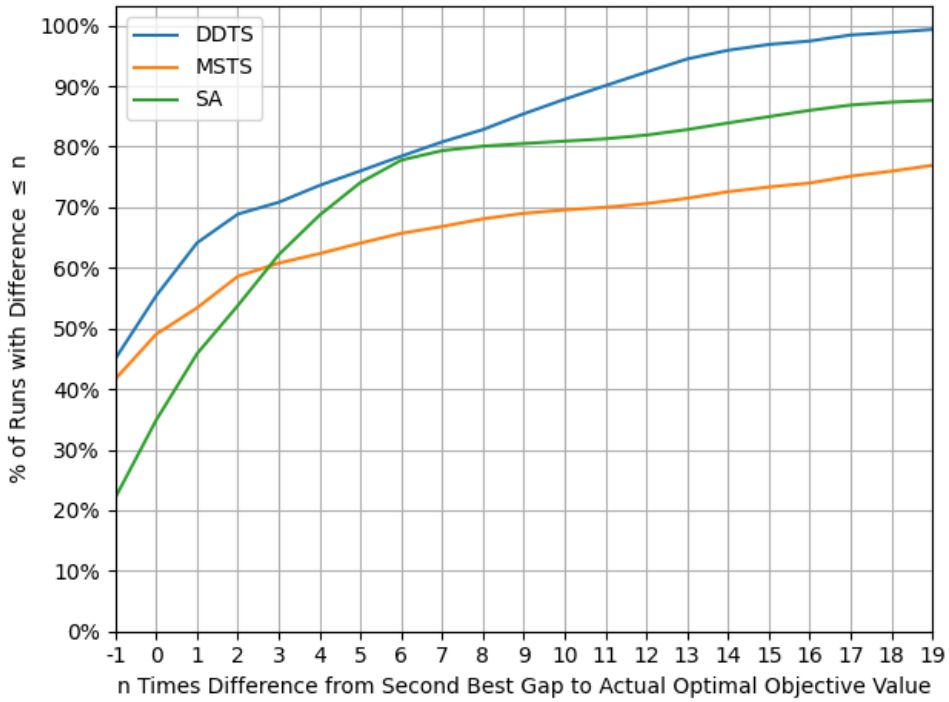

Figure 1: Percentage of Total Runs (100 Runs Times 23 Instances) per Algorithm which Achieve An Objective Value Less or Equal to n Times The Second Lowest Gap in Absolute Term to The Actual Optimal Objective Value.

References

Alidaee, B., Kochenberger, G. A., and Ahmadian, A. (1994). 0-1 quadratic programming approach for optimum solutions of two scheduling problems. *International Journal of Systems Science*, 25(2):401–408.

Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., Sahinidis, N., Vigerske, S., and Wiegele, A. (2018). QPLIB: A library of quadratic programming instances. *Mathematical Programming Computation*.

Glover, F. and Laguna, M. (1998). *Tabu search*. Springer.

Glover, F., Lü, Z., and Hao, J.-K. (2010). Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR*, 8:239–253.

Glover, F. W. and Kochenberger, G. A. (2018). A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538.

Ivănescu, P. L. (1965). Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13(3):388–399.

Katayama, K. and Narihisa, H. (2001). Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research*, 134(1):103–119.

Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2. arXiv:1302.5843 [cond-mat, physics:quant-ph].

Martí, R., Resende, M. G., and Ribeiro, C. C. (2013). Multi-start methods for combinatorial optimization. 226(1):1–8.

Palubeckis, G. (2004). Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research*, 131:259–282.

Appendix: Algorithms' pseudocode

Algorithm 1 summarises the pseudo-code of the diversification-driven tabu search algorithm as defined in Glover et al. (2010).

Algorithm 1: Diversification-Driven Tabu Seach for UBQP Glover et al. (2010).

Input : $Q \in \mathbb{R}^{n \times n}$ matrix.
Output: S^*: the best solution found so far.

Set $EliteSol = \{\}$, $r = 0$, $EliteFreq(i) = 0$ for $i = 1, \ldots, n$;
Randomly generate an initial solution S_0;
while $r < R$ **do**
    $S^* = Tabu\_Search(S_0)$;
    if S^* *is not in* $EliteSol$ **then**
        Insert S^* into $EliteSol$: $EliteSol = EliteSol + \{S^*\}$;
        $r = r + 1$;
        $EliteFreq = EliteFreq + S^*$;
    end
    Randomly select a solution S' from $EliteSol$;
    $S_0 = Perturbation\_Operator(S')$;
end
while *Stop condition is not met* **do**
    Randomly select a solution S' from $EliteSol$;
    $S_0 = Perturbation\_Operator(S')$;
    $S^* = Tabu\_Search(S_0)$;
    $S_w =$ The worst solution in $EliteSol$ in terms of solution quality;
    if S^* *is not in* $EliteSol$ *and* $f(S^*) > f(S_w)$ **then**
        $EliteSol = EliteSol + \{S^*\} - \{S_w\}$;
        $EliteFreq = EliteFreq + S^* - S_w$;
    end
end

Algorithm 2 summarises the pseudo-code of the multistart tabu search algorithm as defined in Palubeckis (2004). Moreover, algorithm 3 shows the steepest ascent procedure as in Palubeckis (2004).

Algorithm 2: Multistart Tabu Search Algorithm 2 Palubeckis (2004).

Input : $Q \in \mathbb{R}^{n\times n}$ matrix.
Output: x^*: the best solution found so far with value f^*.

With probability p for fixing components at 1, randomly generate a vector $x \in \mathbb{B}^n$;
Set $q'_{i,j} := q_{i,j}$ if $x_i = x_j$, and $q'_{i,j} := -q_{i,j}$ else for $i, j = 1, \ldots, n$;
Set $q'_i = (1 - 2x_i)\left(q_{i,i} + \sum_{j,x_j=1}(q_{i,j} + q_{j,i})\right)$ for $i = 1, \ldots, n$;
Set $x^* := x$, $f^* := f(x)$, $S := \{x\}$, $t := 1$ and $m := 0$;
Apply $STS(x, f(x), x^*, f^*, z)$;
while $t < t^*$ **do**
    Set $I := \{1, \ldots, n\}$, $I^* = \emptyset$ and $d_i := q'_i$ for $i = 1, \ldots, n$;
    Set $n' := \max(10, \lfloor \alpha n \rfloor)$;
    while $|I^*| < n'$ **do**
        if $d_i = d_j$ *for all* $i, j \in I$ **then**
            Set $e_i := 1$ for all $i \in I$;
        end
        else
            for $i = 1, \ldots, n$ **do**
                if $d_i \leq 0$ *and* $d_{\min} < 0$ **then**
                    Set $e_i := 1 - \frac{d_i}{d_{\min}}$;
                end
                else if $d_i = d_{\min} = 0$ **then**
                    Set $e_i := 0$;
                end
                else
                    Set $e_i = 1 + \lambda \frac{d_i}{d_{\max}}$;
                end
            end
        end
        Using probabilities $\frac{e_i}{\sum_{i\in I} e_i}$, randomly select an element $k \in I$;
        Set $I := I \setminus \{k\}$ and $I^* := I^* \cup \{k\}$;
        for $i \in I$ **do**
            Set $d_i := d_i + q'_{i,k} + q'_{k,i}$;
        end
    end
    Apply $STEEPEST\_ASCENT(I^*)$ procedure to get x';
    Set $x_i := 1 - x_i$ for each $i \in I^*$ such that $x'_i = 1$;
    Set $q'_{i,j} := q_{i,j}$ if $x_i = x_j$, and $q'_{i,j} := -q_{i,j}$ else for $i, j = 1, \ldots, n$;
    Set $q'_i = (1 - 2x_i)\left(q_{i,i} + \sum_{j,x_j=1}(q_{i,j} + q_{j,i})\right)$ for $i = 1, \ldots, n$;
    Apply $STS(x, f(x), x^*, f^*, z)$;
    Set $t := t + 1$;
end

Algorithm 3: Steepest Ascent Procedure Palubeckis (2004).

Input : I^* set.
Output: x' solution.

for $i \in I^*$ **do**
  Set $h_i^1 := q'_i$;
  Set $h_i^2 := 2\sum_{j \in I^*} q'_{i,j}$;
end
for $k = 1, \ldots, |I^*|$ **do**
  Set $V_1 := -\infty$, $V_2 := -\infty$;
  for $i \in I^*$ **do**
    Set $l_i^1 := 2h_i^1 + h_i^2$;
    Set $l_i^2 := h_i^1$;
    if $l_i^1 > V_1$ *or* $l_i^1 = 0$ *and* $l_i^2 \geq 0$ **then**
      Set $r := 1$;
    end
    else
      Set $l_i^1 := -l_i^1$;
      Set $l_i^2 := -l_i^2$;
      Set $r := 0$;
    end
    if $l_i^1 > V_1$ *or* $l_i^1 = V_1$ *and* $l_i^2 > V_2$ **then**
      Set $V_1 := l_i^1$;
      Set $V_2 := l_i^2$;
      Set $j := i$;
      Set $v := r$;
    end
  end
  Set $x'_j := v$;
  Set $I^* := I^* \setminus \{j\}$;
  for $i \in I^*$ **do**
    Set $h_i^2 := h_i^2 - 2q'_{i,j}$ **if** $v = 1$ **then**
      Set $h_i^1 := h_i^1 + 2q'_{i,j}$;
    end
  end
end
Set $x'_i := 0$ for all $i \in \{1, \ldots, n\} \setminus I^*$ and $x'_i = 1$ else;

Algorithm 4 summarises the pseudo-code of the simulated annealing algorithm with reannealing process as defined in Katayama and Narihisa (2001).

Algorithm 4: Simulated Annealing Katayama and Narihisa (2001).

Input : $Q \in \mathbb{R}^{n \times n}$ matrix.
Output: x_{best}: the best solution found so far.

Generate an initial random solution x_{best};
Set $t = 0$;
while $t < SACount$ **do**
    Set $x = x_{best}$;
    Set $T = T_{init}$;
    while *Not yet frozen* **do**
        Set $l = 0$;
        while $l < TermCount$ **do**
            Generate neighbour solution x' of x randomly;
            Set $\delta := f(x) - f(x')$;
            if $\delta \leq 0$ **then**
                Set $x := x'$;
                Set $x_{best} := x'$;
                Set $l = 0$;
            end
            else
                Set $x := x'$ with probability $e^{-\delta/T}$;
                Set $l := l + 1$;
            end
        end
        Set $T = TFactor \cdot T$;
    end
    Set $T_{init} = StartTFactor \cdot T_{init}$;
    Set $t := t + 1$;
end