Metaheuristic algorithms

Yang Wang and Jin-Kao Hao

Abstract Metaheuristic algorithms are practically used to produce approximate solutions to large QUBO instances that cannot be solved exactly due to the high computational complexity. This chapter is dedicated to a review on the general metaheuristic approach for solving the QUBO. First, we present some basic components of local search that are widely used in the design of state-of-the-art metaheuristic algorithms for the problem. Then we overview the metaheuristic algorithms in the literature by groups of fast solving heuristics, local search based methods and population based search methods. Finally, we review some of the most popular and effective metaheuristic algorithms and present experimental results on different sets of instances.

1 Basic ingredients of local search

Most of the heuristic and metaheuristic algorithms proposed for solving the QUBO employ a local search procedure to improve the solution quality. Given its importance, we first summarize basic ingredients used in these local search procedures, including the solution representation of a QUBO instance, the move operator along with the fast calculation of the move gain, and the neighbor solution selection strategy.

For a given QUBO instance, its solution $x = \{x_1, x_2, \ldots, x_n\}$ is a boolean vector of length n. To perform neighborhood search, the most widely used move operator is 1-flip, which flips a chosen variable x_i to be the complementary value $1 - x_i$. A total of n neighbor solutions are generated by ap-

Yang Wang
School of Management, Northwestern Polytechnical University, Xi'an, China, e-mail: yangw@nwpu.edu.cn

Jin-Kao Hao (Corresponding author)
LERIA, Université d'Angers, Angers, France e-mail: jin-kao.hao@univ-angers.fr

plying the 1-flip move. Some algorithms introduce the k-flip move that flips k ($2 \leq k \leq n$) variables of a solution simultaneously to the corresponding complementary values, respectively. Compared to the 1-flip move, applying the k-flip move yields a much larger number of neighbor solutions up to C_n^k.

The efficiency of local search mostly depends on the evaluation of move gains to determine the next solution on the search trajectory. The move gain of a neighbor solution is the objective variation between a solution and the neighbor solution. Let x and x' represent two binary solutions where x' is obtained from x by a 1-flip move applied to a single variable x_i. The move gain Δ_i of moving from solution x to the neighbor solution x', recorded as $\Delta_i = f(x') - f(x)$, is calculated as:

$$\Delta_i = (1 - 2x_i)(q_{ii} + 2 \sum_{j=1, j \neq i}^{n} q_{ij} x_j) \tag{1}$$

$$\Delta_j = \begin{cases} -\Delta_i & if \quad i = j \\ \Delta_j + 2q_{ij}(1 - 2x_i)(1 - 2x_j) & otherwise \end{cases} \tag{2}$$

Since the generalized k-flip move can be considered as a sequence of 1-flip moves, it is straightforward to infer the following equation for the k-flip move. Formally, let $i_1, \ldots, i_k$ are the k variables to flip, then the move gain $\Delta_{i_1,\ldots,i_k}$ for simultaneously flipping these k variables can be computed as follows:

$$\begin{aligned} \Delta_{i_1,\ldots,i_k} &= \Delta_{i_1} \\ &+ \Delta_{i_2} + 2q_{i_1 i_2}(1 - 2x_{i_1})(1 - 2x_{i_2}) \\ &+ \Delta_{i_3} + 2q_{i_1 i_3}(1 - 2x_{i_1})(1 - 2x_{i_3}) + 2q_{i_2 i_3}(1 - 2x_{i_2})(1 - 2x_{i_3}) \\ &+ \Delta_{i_4} + 2q_{i_1 i_4}(1 - 2x_{i_1})(1 - 2x_{i_4}) + 2q_{i_2 i_4}(1 - 2x_{i_2})(1 - 2x_{i_4}) \\ &+ 2q_{i_3 i_4}(1 - 2x_{i_3})(1 - 2x_{i4}) \\ &\qquad \vdots \qquad\qquad\qquad\qquad \vdots \\ &= \sum_{r=1}^{k} \Delta_{i_r} + 2 \sum_{r=1}^{k-1} \sum_{s=r+1}^{k} q_{i_r i_s}(1 - 2x_{i_r})(1 - 2x_{i_s}) \end{aligned} \tag{3}$$

The neighbor solution selection strategy decides the neighbor solution that is used to replace the current solution x during the local search. The first move improvement and the best move improvement are two strategies often used. The first move improvement strategy scans the neighborhood of x and chooses the first neighbor solution that is better than x (i.e., with an improving move gain). The best move improvement strategy exhaustively explores the neighborhood of x and retains the neighbor solution with the highest move gain.

2 Fast solving heuristics

Boros et al. [7] developed a Devour Digest Tidy-up procedure (DDT). On the basis of the posiform representation Z of QUBO, DDT includes the Devour, Digest and Tidy-up phases. Devour identifies a term T from L (L denotes the set of all the elements of Z) with the largest coefficient and places it into S. Digest draws logical conclusions by assigning the disjunctive equation of all the elements in S equaling to 0 (in terms of minimization). If no logical conclusion can be drawn, then T is simply removed from L to S, and return to Devour. Otherwise, Tidy-up begins to substitute the logical conclusions previously drawn into Z. The above DDT procedure repeats until L becomes an empty set. Experiments indicated that DDT is especially effective on problem instances of low density.

Consider the case that the DDT method simultaneously sets several variables with value 1 or 0 would result in worse results than to give inferred assignment to only one variable, Glover et al. [10] proposed several one-pass heuristics to guarantee that in each pass only one variable gets the implied assignment. The difference among the proposed one-pass heuristics lies in the different strategies of evaluating contributions of variables. Experimental comparisons among the proposed one-pass heuristics showed that some of them perform quite effectively for certain problem instances, but no single method dominates on every problem instance.

Hanafi et al. [13] devised five alternative DDT heuristics based on different representations of the QUBO formulation, where DDT1 to DDT4 methods respectively have standard, posiform, bi-form and negaform representations and DDT5 has a posiform representation along with a one-pass mechanism. An obviously additional difference of their DDT alternatives from [7, 10] concerns the use of a *r-flip* local search procedure to improve solutions obtained by DDT constructions. Extensive tests on small, medium and large benchmark instances showed that (1) DDT3 with the bi-form representation generally produces the best results for medium and large instances; (2) the proposed *r-flip* local search brings significant result improvements with only a slight increase of time consumption.

Merz and Freisleben [23] proposed a greedy construction heuristic to quickly obtain an improved solution. It starts from a solution with all variables assigned to be 0.5 (the so called third state). At each construction step, it searches a variable and a value either 0 or 1 for the variable such that assigning the value to the variable makes the gain function, i.e., objective function increment of the resulting solution compared to the previous solution, is maximized. This operation repeats until each variable of the solution vector changes its initial value from 0.5 to 1 or 0. Since this greedy construction method always obtains the same solution for a given problem instance, a randomized greedy variant was proposed to overcome the deterministic drawback. For one thing, randomly pick a variable and randomly assign a value 0 or 1 to it for the first step of the construction. For another thing, select a

variable with a probability proportional to the gain value instead of picking a variable with the maximized gain value.

3 Local search based methods

3.1 Simulated annealing

Alkhamis et al. [1] presented a simulated annealing based heuristic (SA-AHA) according to the traditional simulated algorithm framework. It begins with a randomly generated solution and an initial temperature. At each iteration SA-AHA generates a random 1-flip move. If this is an improving move, it is performed; Otherwise, it may still be accepted with a probability $e^{-\Delta/T}$ where Δ indicates the objective function difference between the two solutions and T is the current temperature constant. After the above procedure conducts a certain number of iterations, the temperature is decreased with reference to a cooling function. The above procedure is repeated until no solution has been accepted for 10 consecutive temperatures or when the temperature has fallen below a pre-specified value. Tested on problem instances with up to 100 variables and comparisons with several bounding techniques based algorithms indicated that SA-AHA outperforms these compared methods. Especially, SA-AHA is able to solve hard problem instances very efficiently while bounding algorithms can not solve them in a reasonable computation time. Additional experiments indicated that the efficiency of the SA-AHA algorithm is not affected by matrix density.

Beasley [5] proposed another simulated annealing algorithm (SA-B). The basic iterative procedure of SA-B is the same as SA-AHA. However, in SA-B each iteration applies a different temperature value to determine the probability to accept a worse move. In addition, a local search procedure based on the first improvement strategy is utilized to perform a post-optimization of the solution from the annealing process. Experimental results for 45 instances with up to 500 variables indicated that SA-B converges fast to the best solutions than the reference algorithms but obtains inferior solution quality for several instances. In addition, the author generated 60 instances with up to 2500 variables available from OR-Library. Results on this new set of benchmark instances showed that SA-B is especially effective for 10 largest instances with 2500 variables.

Katayama and Narihisa [16] designed a similar implementation of the simulated annealing methodology as SA-AHA, called SA-KN. An obvious characteristic of SA-KN different from SA-AHA and SA-B lies in the fact that it adopts multiple annealing processes to enhance the search ability. Experimental results for problem instances with variables ranging from 500 to 2500 indicated that SA-KN achieves especially competitive performances for the largest OR-Library instances.

3.2 Tabu search

Glover et al. [9] introduced an adaptive memory tabu search (AMTS) algorithm that uses the 1-flip move and two types of memory structures to record recency and frequency information. Strategic oscillation is employed to alternate between constructive phases (progressively setting variables to 1) and destructive phases (progressively setting variables to 0), which are triggered by critical events, i.e., when the next move causes the objective value to decrease. The amplitude of the oscillation is adaptively controlled by a span parameter. Computational results for instances with up to 500 variables showed that AMTS outperforms the best exact and heuristic methods previously reported in the literature.

Beasley [5] proposed a 1-flip move based tabu search algorithm (TS-B). It begins from an initial solution with each variable assigned to be 0 and marked as non-tabu. During each iteration it conducts a best non-tabu move. This performed move is then marked as tabu for a specified number of following iterations. If the current iteration finds a better solution than the best solution found so far, a local search procedure with first-improvement strategy is launched to further improve this new solution. TS-B repeats the above procedure until the current iteration reaches the maximum allowed iteration. Notice that TS-B does not incorporate the fast evaluation technique and also neglects an aspiration criterion.

Palubeckis [25] examined five multistart tabu search strategies (MSTS) dedicated to the construction of the initial solution. Each multistart tabu search algorithm employs a tabu search procedure (TS-P) to enhance solution quality and a multi-start strategy to produce a new initial solution located in a more promising area. Notice that TS-P is very similar to TS-B except that TS-P employs a tactic to get 1-flip moves fast evaluated. The first restart strategy produces a new initial solution in a random way. The second restart strategy identifies a candidate set of variables that are prone to change their values when moving from the current solution to an optimal one. Then it applies a steepest ascent algorithm that only considers variables in the candidate set and keeps the other variables fixed at specific values. The third one employs a randomized greedy constructive method. The fourth one incorporates a set of elite solutions and calculates the probability of each variable with value 1 in this set. If the probability for a given variable is larger than 0.5, then this variable receives value 1 in the resulting new solution; otherwise it receives value 0. The last restart strategy uses a perturbation scheme of changing the problem instance at hand, followed by a short run of tabu search on the modified instance. Experiments evaluated on 25 largest instances from OR-Library and a set of randomly generated larger and denser instances demonstrated that the algorithm with the second restart strategy (MST2) is the best among the proposed algorithms.

Palubeckis [26] developed an iterated tabu search algorithm (ITS) that combines a tabu search procedure to improve the solution quality and a

perturbation mechanism to create a new initial solution. The tabu search procedure is exactly the one used in [25]. The perturbation mechanism is operated as follows. First, it constructs a candidate list of a specified size which consists of variables with the largest 1-flip move gains with regard to a local optimal solution. Then it randomly selects a variable from this set and flips this variable to move toward a new solution. Finally, it updates the corresponding move gains of variables caused by the move. The above procedure is repeated until the number of perturbed variables reaches the specified count. Experimental results indicated that despite its simplicity, ITS is very competitive compared to other state-of-the-art algorithms.

Lü et al. [21] studied neighborhood union and token-ring search methods to combine 1-flip (N1) and 2-flip (N2) moves within a tabu search algorithm. The 2-flip move based tabu search considers a constrained set of 2-flip moves requiring that flipping each involved variable produces the move gain ranked top $3\sqrt{n}$ among all the 1-flip moves. In this way, the computational efforts of exploring the neighborhood N2 can be greatly reduced. The neighborhood union includes the strong neighborhood union ($N1 \bigsqcup N2$) that picks each move from both N1 and N2 and the selective neighborhood union ($N1 \bigcup N2$) that select a move from N1 with probability p and N2 with probability $1-p$. The token ring search (N1 $\rightarrow$ N2) continuously performs moves within a single neighborhood until no improvement is possible and then switches to the other neighborhood to carry out moves in the same fashion. Experimental results on random large instances indicated that selective union is superior to the other two neighborhood combinations.

Liu et al. [18] proposed a hybrid r-flip/1-flip tabu search algorithm (HLS) which switches among a hybrid local search phase, a destruction phase and a construction phase. First, the hybrid local search phase that hybrids 1-flip and r-flip local search is launched. This phase behaves like a basic variable neighborhood search procedure [12] but excludes useless r-flip moves by several orders according to a theorem. When no improved move is found, the hybrid local search phase terminates. Meantime, the destruction phase is followed to carry out the 1-flip move with the least damage to the current solution. The performed move is marked as tabu and the destruction phase continues until an improving non-tabu move occurs. At this point, a construction phase is triggered to perform the best non-tabu move. If the obtained solution is better than the best solution ever found, the algorithm returns to the hybrid local search phase. If no variable exists that can make further improvement, the algorithm then returns to the destruction phase. Tested results showed the superiority of the proposed hybrid r-flip/1-flip tabu search especially for solving large instances with high density.

Shylo and Shylo [27] developed a global equilibrium search (GES) algorithm that performs multiple temperature cycles. Each temperature cycle alternates between an initial solution generation phase and a tabu search phase. The use of information from the whole search history helps to determine the probability of a variable receiving value 1 in the generated solution.

The tabu search procedure performs the best 1-flip move with an additional requirement that this move leads to a solution far enough from a reference set in terms of hamming distance. Experimental results showed that GES performs quite well in terms of solution quality and computing time.

Wang et al. [29] devised GRASP-TS and GRASP-TS/PM algorithms that hybrid GRASP with tabu search. GRASP-TS uses a basic GRASP algorithm with single solution search while GRASP-TS/PM launches each tabu search by introducing a population management strategy based on an elite reference set. Specifically, GRASP-TS uses an adaptive random greedy function to construct an initial solution from scratch. GRASP-TS/PM makes use of a restart/recovery strategy to produce a solution, in which partial solution components inherit corresponding elements of an elite solution fetched from a population and the remaining solution components are rebuilt as in the GRASP-TS procedure. Experiments indicated that GRASP-TS and GRASP-TS/PM are very competitive with state-of-the-art algorithms.

4 Population based search methods

Amini et al. [2] presented a scatter search approach (SS) that is mainly composed of a diversification generation method, a solution improvement method, a reference set update method, a subset generation method and a solution combination method. The diversification generation method systematically generates a collection of diverse trial solutions based on a seed solution in a way of setting an incremental parameter that determines which bits of the seed solution should be flipped. The improvement method performs a compound move that sequentially cycles among 1-flip, 2-flip and 3-flip candidate moves until no attractive move can be identified. The reference set update method replaces solutions in the reference set with new candidate solutions according to the quality measurement. In order to build a new solution, a linear combination of selected solutions from the reference set is applied. Since some variables would receive fractional values in the combined solution, a rounding procedure is followed to make this solution feasible. Experiments showed that the proposed scatter search method is very robust, especially for large problem instances.

Lodi et al. [19] introduced an evolutionary heuristic (EH) with the following features. First, EH uses a preprocessing phase to fix certain variables at their optimal values and reduce the problem size. This type of fixation belongs to permanent fixation since for each successive round of local search, these variables are excluded from consideration. Second, a local search procedure based on the alternation between construction and destruction phases is employed to get an improved solution. Finally, EH uses a uniform crossover operator to generate offspring solutions, where variables with common values in parental solutions are temporarily fixed in this round of local search.

Experimental results showed that EH can match the best known results for problem instances with up to 500 variables in a very short computation time. A further analysis demonstrated that the preprocessing phase is effective for small problem instances but is impossible to reduce the problem size for large ones.

Merz and Freisleben [22] devised a hybrid genetic algorithm (GLS-MF), in which a simple local search is incorporated into the traditional genetic algorithm. The local search procedure uses the 1-flip move and best move improvement strategy. The crossover operator is a variant of uniform crossover, requiring the generated offspring solution has the same hamming distance from the parents. Once the newly generated offspring solution satisfies the updating criterion, it becomes a member of the population and replaces the worst solution. A diversification component is launched when the average hamming distance of the population drops below a threshold $d = 10$ or the population is not updated for more than 30 consecutive generations. Experimental results showed that the simple evolutionary algorithm alone is able to find the best known results for problem instances with less than 200 variables but for larger instances, it is essential to incorporate local search to attain high quality solutions.

Lü et al. [20] proposed a hybrid metaheuristic approach (HMA) which integrates a basic tabu search procedure into a genetic search framework. First, HMA combines a traditional uniform crossover operator with a diversification guided path relinking operator to guarantee the quality and diversity of an offspring solution. Second, HMA replaces the Hamming distance by a new distance by reference to variable's importance and employs a quality-and-distance criterion to update the population as in GTA. Finally, a tabu search procedure is responsible for intensified examination around the offspring solutions. Computational results showed HMA is among the best performing procedures for solving challenging QUBO problem instances.

5 Selected metaheuristic approaches for QUBO

5.1 Diversification-driven tabu search

Glover et al. [11] presented a diversification-driven tabu search (D^2TS) algorithm that alternates between a basic tabu search procedure and a memory-based perturbation procedure guided by a long-term memory. The general scheme of D^2TS works as follows. Starting from a random initial solution, D^2TS uses tabu search to reach local optimum. Then, the perturbation operator is applied to displace the solution to a new region, whereupon a new round of tabu search is launched. To achieve a more effective diversification, the perturbation operator is guided by information from a special memory structure for obtaining improved results in this context.

The tabu search procedure employs the 1-flip neighborhood. Each time a move is carried out, the reverse move is forbidden for the next $TabuTenure$ iterations. The tabu tenure is set to be $TabuTenure(i) = tt + rand(10)$, where tt is a selected constant and rand(10) takes a random value from 1 to 10. Once a move is performed, a subset of move values affected by the move is updated using a fast incremental evaluation technique. Accompanying this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the best solution found so far. The TS procedure stops when the best solution cannot be improved within a given number of moves.

The perturbation procedure includes assigning a score to each variable, selecting a certain number of highly-scored variables (critical elements), and perturbing the solution using the chosen critical elements. The scoring function depends on the information of several memory structures, including a flipping frequency vector $FlipFreq(i)$, an elite set of solutions *EliteSol* and a consistency vector $EliteFreq(i)$. $FlipFreq(i)$ records the number of times the a variable x_i has been flipped from the beginning until the current iteration, which is collected in the tabu search phase. *EliteSol* stores a set of locally optimal solutions found by tabu search. Each time a new local optimum is found that has the objective value superior to that of the worst local solution in *EliteSol*, the new solution replaces this worst solution. $EliteFreq(i)$ records the total number of times a variable x_i is assigned value 1 in the elite solutions currently stored in *EliteSol*. This memory is used to favor retaining the value assignments that occur more often in the best solutions found to date. The scoring function ranks each variable by taking into account its flip frequency $FlipFreq(i)$ and its elite value frequency $EliteFreq(i)$, which takes the following form:

$$Score(x_i) = \frac{EliteFreq(i)(r - EliteFreq(i))}{r^2} + \beta(1 - \frac{FlipFreq(i)}{maxFreq}) \quad (4)$$

The selection step sorts all the variables in non-increasing order according to their scores and then adaptively selects a specified number of critical variables to be randomly assigned a value 0 or 1. The higher the score a variable has, the greater the probability it will be chosen. The perturbation step flips the values of the selected critical variables. This perturbed solution is then used to initiate a new round of tabu search.

5.2 Memetic search

Merz and Katayama [24] conducted a landscape analysis and observed that (1) local optima of the QUBO instances are contained in a small fraction of the search space; (2) the fitness of local optima and the distance to the opti-

mum are correlated. Based on this, they designed a memetic algorithm (MA-MK) in which an innovative variation as the crossover operator is utilized to generate good starting solutions. The MA-MK algorithm is composed of population initialization, randomized k-opt local search, crossover and variation, as well as selection and diversification strategies.

The population initialization procedure repeats generating individuals in the following way until the population size reaches 40. The method to generate an individual includes two steps. The first step employs a randomized greedy heuristic introduced in [23] to produce a seeding solution. The second step applies a randomized k-opt local search to optimize the solution to local optimum.

The randomized k-opt local search is based on the ideas of Lin and Kernighan for solving the traveling salesman problem (TSP) and the graph partitioning problem that searches a small fraction of the k-opt neighborhood efficiently. The randomized k-opt local search performs the following iterations until performing a k-flip move can not yield an improving solution. For each k-opt iteration, all the bits of a solution are sorted in a random order and only the bits of getting the positive move gains are flipped. The bit with the maximum move gain is subsequently flipped. The above-mentioned procedure are repeated until all the bits have been flipped. The best solution is recorded as the resulting solution in this iteration.

The crossover operator introduced the move gain to determine the variation of the offspring solution in order to prevent rediscovering local optima already visited to the most extent. Specifically, the common and the non-common bits of the parent solutions are identified and the initial offspring solution is set to be any parent solution. The bits in the non-common and common sets are operated alternatively. For the non-common set, all the non-common bits with the positive 1-flip move gains are identified and such a bit is randomly selected. For the common set, the common bit with the maximum associated 1-flip move gain is identified even if the move gain is negative. If a bit is flipped, it is removed from the corresponding set. The above-mentioned procedure is repeated for a number of times equal to the size of the non-common set.

In each generation, a new population needs to be formed after offspring individuals are generated. Among the old individuals in the previous generation and the newly generated offspring individuals, those with the highest fitness are selected to maintain the restricted population size. If no new best individual in the population was found for more than 30 generations, a diversification restart strategy is triggered. All the individuals except for the best one in the population are mutated by flipping randomly chosen $n/3$ bits for each individual of length n. After that, each individual is optimized by the randomized k-opt local search to obtain a renewal set of local optima and the search is started again with the newly diversified population.

5.3 Path relinking

Wang et al. [30] proposed two path relinking algorithms, which is composed of a reference set initialization method, a solution improvement method, a reference set update method, a relinking method and a path solution selection method. The proposed algorithms differ from each other mainly on the way they generate the path, one employing a greedy strategy and the other employing a random construction.

The general scheme of the path relinking algorithm works as follows. It starts with the creation of an initial set of elite solutions RefSet, based on which an index set of pairwise solutions PairSet is generated. For each index pair (i, j), a relinking method is applied to generate two paths connecting the elite solutions x^i and x^j, one of which is from x^i to x^j and the other is from x^j to x^i. Then, one solution on each path is selected according to a path solution selection method and refined by a solution improvement method using the same tabu search procedure as in [20]. The resulting solution is subject to the RefSet updating procedure. The above-mentioned procedure is terminated once all the elements in PairSet are examined. If the given stopping criterion is not satisfied, RefSet and PairSet are rebuilt to continue the search.

The RefSet initialization method is used to construct an elite set of high-quality solutions, where each solution is obtained in two steps. The first step generates a randomized solution, where each variable receives value 0 or 1 with an equal probability of 0.5. The second step employs a tabu search based solution improvement method to refine the quality of this solution. Afterwards, the RefSet updating procedure is invoked. The improved solution is permitted to be added into RefSet if it is distinct from any solution in RefSet and better than the worst solution. Once this condition is satisfied, the worst solution is replaced by the improved solution. When PairSet becomes empty, RefSet is recreated. The best solution previously found becomes a member of the new RefSet and the remaining solutions are generated in the same way as in constructing RefSet in the first round.

The path relinking method builds a path connecting an initiating solution where the path starts with and a guiding solution where the path ends at. The path consists of a sequence of intermediate solutions, each of which is generated by exploring the neighborhood of the initiating and guiding solutions. To be specific, identify the set of non-common variables NC where the initiating solution $x^i = x(0)$ and the guiding solution x^g have different values. Meanwhile, initialize another vector where each entry Δ_t denotes the objective difference resulting after flipping the variable $x_t \in NC$ from the previous solution $x(k-1)$ on the path. The path relinking method performs a total of $|NC| - 1$ iterations to construct a path. At each path construction step k, either use a greedy strategy to select the variable having the maximum $\Delta_{t \in NC}$ value in the algorithm PR1 or use a random strategy to randomly select a non-common variable in the algorithm PR2. The path solution $x(k)$

is determined by assigning the same value as the guiding solution x^g for the newly chosen variable and assigning the same values as $x(k-1)$ for all the other variables.

Since two consecutive solutions on a relinking path differ only by flipping a single variable, it is not productive to apply an improvement method to each solution on the path since many of these solutions would lead to the same local optimum. Hence, the path solution selection method chooses only a single solution on the path explored via path relinking. Specifically, it first constructs a candidate solution list that includes the solutions with a Hamming distance of at least $|NC|/3$ from both the initiating and guiding solutions. Then, the candidate solution with the maximum objective value is picked for further amelioration by the solution improvement method.

5.4 Automatic grammar-based design of heuristic algorithms

de Souza and Ritt [28] designed an automatic algorithm by combining the problem-specific components of state-of-the-art algorithms from the literature. The designed automatic algorithm employs a grammar in Backus-Naur form to model the space of heuristic strategies and their parameters. The grammar is composed of a set of rules, through which the heuristic algorithms can be instantiated. All heuristic strategies are categorized into construction methods, search methods and recombination methods. The construction methods start from an empty solution and apply the greedy randomized heuristics to iteratively set value 0 or 1 to each variable. The search methods include local search components, tabu search components and iterated local search components, which start with an initial solution and perform 1-flip moves to explore the search space. The differences among local search components lie in the move selection strategies that transfer from the current solution to its neighbor solution, such as the first improvement strategy, a round-robin strategy, a best improvement strategy, etc. Two tabu search components are differentiated, one of which always selects the best move and the other selects a random move with a small probability. In addition to different local search strategies, the iterated local search components include perturbation strategies of random perturbation, least-loss perturbation and frequency based perturbation as well as different strategies to define the perturbation strength. The recombination methods employ the path relinking algorithms that evolve a population of elite solutions.

To perform exploration of the grammar based search space, they used the irace tool described in [17] to rank different candidates of the algorithms found in the literature of QUBO and newly generated algorithms. The irace tool implements an iterated racing procedure that iteratively selects a candidate and an instance from a set of elite candidates and calls the resulting

algorithm. After performing each iteration, the average difference between the objective value of the found solution and the best known value is used as a result metric for irace to rank different candidates. The process is repeated until irace has finished the specified number of algorithm runs. Experimental results indicate that the automatic approach can find algorithms that outperform state-of-the-art algorithms.

5.5 A systematic evaluation of heuristics

Dunning et al. [8] implemented a total of 37 Max-Cut and QUBO heuristics selected from the literature and performed a systematic evaluation on a library of 3296 instances. Because no single heuristic outperforms all others across all problem instances, a regression tree model is built for each of the 37 heuristics to determine the rank on a given instance. Based on this, key insights into when heuristics perform well or poorly according to the problem instance characteristics are identified. Moreover, a random forest model is built for each heuristic to predict the probability of a heuristic that will perform the best for a given instance. By selecting a set of heuristics with the highest predicted probability, an algorithm portfolio is finally constructed to produce the best solution. Unlike many hyper-heuristics found in the literature, the proposed approach does not construct a new heuristic by selecting from a set of heuristic components. In addition, it does not include and implement several advanced algorithms and thus lacks of comparisons with these algorithms. Results indicate that the proposed algorithm portfolio dominates each of the 37 Max-Cut and QUBO heuristics it combines. The open-source implementations of the heuristics are publicly available [1].

6 Computational results

6.1 Benchmark instances

Three sets of test problems are often used in the QUBO literature to evaluate the performance of algorithms. The first set is composed of 10 large instances from the OR-Library [4, 5]. [2] They all have a density of 0.1 and are named by b2500.1,. . .,b2500.10. The second set of benchmarks consists of 21 randomly generated large problem instances named p3000.1,. . .,p7000.3 with sizes ranging from n=3000 to 7000 and with densities from 0.5 to 1.0

[1] https://github.com/MQLib/MQLib

[2] http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/bqpinfo.html

[25, 26]. [3] These large instances are particularly challenging QUBO problems, especially in the case of instances with more than 5000 variables. The third set of benchmarks includes 54 instances derived from the MaxCut problem, named G1,...,G54, with variable sizes ranging from n=800 to 2000 [6, 14].[4] These instances are created by using a machine-independent graph generator, composed of toroidal, planar and random weighted graphs with weight values 1, 0 or -1. The small test instances from the OR-Library whose sizes range from n = 500 to 1000 can be solved relatively easily by many algorithms.

It is worth noting that a completely fair comparison of algorithms is impossible since the compared algorithms are implemented by different authors and run under different conditions. The presented comparisons on the QUBO instances as well as that on the MaxCut problem are thus presented only for indicative purposes and should be interpreted with caution.

6.2 Computational results on the QUBO instances

The PR1 and PR2 algorithms were tested on a PC with Pentium 2.83GHz CPU and 8GB RAM. The running CPU time for each run of PR1 and PR2 is set to 60 seconds for solving the 10 OR-Library instances and set to 5, 10, 20, 30 and 50 minutes for solving the instances of second set with 3000, 4000, 5000, 6000 and 7000 variables. The time limits are comparable to that used by the Diversification-Driven Tabu Search (D^2TS) [11], Iterated Tabu Search (ITS) [26], MultiStart Tabu Search (MST2) [25], Memetic Algorithm (MA) [24] and Automatic Algorithm (AAC_R) [28] after considering the computing performance of different machines.

Table 1 shows the results obtained by the 6 reference algorithms for solving the 10 bxxx.y instances. Columns 1 and 2 respectively give the instance name and the best known result BKR reported in the literature. The following columns list the average solution gap to the best known result $BKR - f_{avr}$. Given that all the reference algorithms are capable of finding the best known results, we do not report for each instance the tabulated solution gap 0 between the best solution value found by each algorithm and the best known result. The last row "Average" indicates the summary of each algorithm's average performance over this set of instances.

As shown in Table 1, D^2TS is able to reach the best known results during each run for all the 10 instances. PR1 and PR2 perform slightly worse by failing for 1 instance. The average solution gaps to the best known results obtained by PR1, PR2, ITS and MST2 are 1.3, 5.8, 2 and 1.1, respectively, which are quite small compared to the solution values. MA performs the

[3] http://www.soften.ktu.lt/∼gintaras/ubqop_its.html

[4] http://www.stanford.edu/∼yyye/yyye/Gset

worst among all the algorithms by obtaining an average solution gap of 226 to the best known results.

Table 1 Average results comparison on the instances of the first set

Instance	BKR	Average solution gap (i.e., $BKR - f_{avg}$)					
		PR1[30]	PR2[30]	ITS[26]	MST2[25]	D^2TS[11]	MA[24]
b2500.1	1515944	0	0	0	0	0	13
b2500.2	1471392	0	58	9	0	0	645
b2500.3	1414192	13	0	11	11	0	173
b2500.4	1507701	0	0	0	0	0	0
b2500.5	1491816	0	0	0	0	0	55
b2500.6	1469162	0	0	0	0	0	190
b2500.7	1479040	0	0	0	0	0	416
b2500.8	1484199	0	0	0	0	0	3
b2500.9	1482413	0	0	0	0	0	321
b2500.10	1483355	0	0	0	0	0	446
Average		1.3	5.8	2	1.1	0	226

Tables 2 and 3 show the best and average solution gaps to the best known results for solving the 21 pxxx.y instances. We replace the algorithm MA by AAC_R since the latter is recently proposed and reports much better results. Table 2 indicates that PR1 and PR2 achieve the best known results for all the 21 challenging instances. AAC_R and D^2TS perform slightly worse since they fail to reach the best known results for 1 and 2 instances, respectively. ITS and MST2 obtain the worst gaps of 306.8 and 308.9 on average with respect to the best solution found. Table 3 indicates AAC_R performs the best with an average solution gap of 211.7. PR1 and PR2 obtain the average solution gaps of 457.1 and 690.4, respectively, which are slightly worse than AAC_R. D^2TS obtains the worst average solution gap of 2082.9 among all the algorithms.

6.3 Computational results on the MaxCut instances

The maximum cut problem can be naturally transformed into the QUBO model. Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$, each edge $e(i, j)$ is associated with a weight w_{ij}, the maximum cut problem (MaxCut) asks for a partition of V into two disjoint subsets such that the total weight of the cut (edges crossing the two subsets) is maximized. Formally, the objective function of MaxCut is:

$$\begin{aligned} &\text{Maximize:} \quad f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i (1 - x_j), \\ &\text{subject to:} \quad x_i \in \{0, 1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{5}$$

Table 2 Best results comparison on the instances of the second set

Instance	BKR	best solution gap (i.e., $BKR - f_{best}$)					
		PR1[30]	PR2[30]	ITS[26]	MST2[25]	D^2TS[11]	AAC_R[28]
p3000.1	3931583	0	0	0	0	0	0
p3000.2	5193073	0	0	0	0	0	0
p3000.3	5111533	0	0	0	0	0	0
p3000.4	5761822	0	0	0	0	0	0
p3000.5	5675625	0	0	0	0	0	0
p4000.1	6181830	0	0	0	0	0	0
p4000.2	7801355	0	0	0	0	0	0
p4000.3	7741685	0	0	0	0	0	0
p4000.4	8711822	0	0	0	0	0	0
p4000.5	8908979	0	0	0	0	0	0
p5000.1	8559680	0	0	700	325	325	0
p5000.2	10836019	0	0	0	582	0	0
p5000.3	10489137	0	0	0	0	0	0
p5000.4	12252318	0	0	934	1643	0	0
p5000.5	12731803	0	0	0	0	0	0
p6000.1	11384976	0	0	0	0	0	0
p6000.2	14333855	0	0	88	0	0	0
p6000.3	16132915	0	0	2729	0	0	0
p7000.1	14478676	0	0	340	1607	0	0
p7000.2	18249948	0	0	1651	2330	104	8
p7000.3	20446407	0	0	0	0	0	0
Average		0	0	306.8	308.9	20.4	0.4

Table 3 Average results comparison on the instances of the second set

Instance	f_{prev}	average solution gap (i.e., $BKR - f_{avg}$)					
		PR1[30]	PR2[30]	ITS[26]	MST2[25]	D^2TS[11]	AAC_R[28]
p3000.1	3931583	0	80	0	0	0	0
p3000.2	5193073	0	0	97	97	0	0
p3000.3	5111533	36	72	344	287	0	108
p3000.4	5761822	0	0	154	77	0	0
p3000.5	5675625	90	279	501	382	0	49
p4000.1	6181830	0	0	0	0	0	0
p4000.2	7801355	71	314	1285	804	0	323
p4000.3	7741685	0	64	471	1284	0	3
p4000.4	8711822	0	0	438	667	0	0
p4000.5	8908979	491	385	572	717	0	0
p5000.1	8559680	612	918	971	581	656	387
p5000.2	10836019	620	499	1068	978	12533	339
p5000.3	10489137	995	318	1266	1874	12876	77
p5000.4	12252318	1258	1168	1952	2570	1962	554
p5000.5	12731803	51	166	835	1233	239	37
p6000.1	11384976	201	822	57	34	0	18
p6000.2	14333855	221	577	1709	1269	1286	148
p6000.3	16132915	1744	2017	3064	2673	787	672
p7000.1	14478676	935	1523	1139	2515	2138	903
p7000.2	18249948	1942	2986	4301	3814	8712	828
p7000.3	20446407	332	2311	3078	7868	2551	0
Average		457.1	690.4	1109.6	1415.4	2082.9	211.7

The following corresponding relation is apparent in comparison with the formulation of QUBO:

$$q_{ii} = \sum_{j=1, j\neq i}^{n} w_{ij}, \quad q_{ij} = -w_{ij}, \quad (i \neq j) \tag{6}$$

According to Eq. (6), a MaxCut problem instance can be reformulated into a QUBO instance. Thus, the algorithms designed for solving the QUBO problem are directly applicable to the MaxCut problem.

Table 4 reports the computational results on the 54 MaxCut instances. For each execution of the compared algorithms, the time limit for solving each instance is set to be 30 minutes. Columns 1 gives the instance name. Columns 2 to 4 list the best solution values found by PR1, PR2 and D^{2}TS. Columns 5 to 8 list the average solution values found by PR1, PR2, AAC$_M$ and AAC$_R$. Both AAC$_M$ and AAC$_R$ are proposed in [28], where AAC$_R$ uses the pxxx.y instances of as training inputs to irace and AAC$_M$ is trained on the MaxCut instances. The last row "Matched" indicates the number of instances where each algorithm obtains the best results among the compared algorithms. In terms of the best solution values, PR2 matches the best results for 47 out of 54 instances, performing better than PR1 and D^{2}TS that match 38 and 29 best results, respectively. In terms of the average solution values, AAC$_M$ performs the best by matching the best results for 44 instances, much better than AAC$_R$ that only matches 2 best results. This indicates that the behavior of the automatic approach is closely correlated to the problem structure.

References

1. T.M. Alkhamis, M. Hasan, M.A. Ahmed, Simulated annealing for the unconstrained quadratic pseudo-boolean function, European Journal of Operational Research, 108 (1998) 641–652.
2. M.M. Amini, B. Alidaee, G. Kochenberger, A scatter search approach to unconstrained quadratic binary programs, In D. Corne, M. Dorigo, and F. Glover (Eds.), New Ideas in Optimization, pp. 317–329, McGraw-Hill, 1999.
3. E.A.J. Anacleto, C. Meneses, S.V. Ravelo, Closed-form formulas for evaluating r-flip moves to the unconstrained binary quadratic programming problem, Computers and Operations Research, 113 (2020) 104774.
4. J.E. Beasley, Obtaining test problems via Internet, Journal of Global Optimization, 8 (1996) 429–433.
5. J.E. Beasley, Heuristic algorithms for the unconstrained binary quadratic programming problem, PhD thesis, Imperial College, England, December 1998.
6. Benson S.J., Y. Ye, X. Zhang, Mixed linear and semidefinite programming for combinatorial and quadratic Optimization, Optimization Methods and Software, 11 (1999) 515-544.
7. E. Boros, P.L. Hammer, X. Sun, The ddt method for quadratic 0-1 minimization, Rrr 39-89, RUTCOR Research Center, 1989.

Table 4 Computational results comparison on the MaxCut instances

Instance	f_{best}			f_{avg}			
	PR1[30]	PR2[30]	D^{2}TS[11]	PR1[30]	PR2[30]	AAC$_M$[28]	AAC$_R$[28]
G1	11624	11624	11624	11624	11624	11624	11607.8
G2	11620	11620	11620	11620	11620	11620	11606.2
G3	11620	11620	11620	11620	11620	11622	11611.2
G4	11646	11646	11646	11646	11646	11646	11633.2
G5	11631	11631	11631	11631	11631	11631	11620.1
G6	2178	2178	2178	2178	2178	2178	2172.1
G7	2006	2006	2006	2006	2006	2006	1995.8
G8	2005	2005	2005	2005	2005	2005	1998.5
G9	2054	2054	2054	2054	2054	2054	2044.4
G10	2000	2000	2000	2000	1999.8	2000	1989.8
G11	564	564	564	564	564	564	556.4
G12	556	556	556	556	556	556	546
G13	582	582	580	582	582	582	568
G14	3063	3064	3061	3062.1	3062.6	3062.6	3055.9
G15	3050	3050	3050	3049.3	3049.3	3049.9	3035.8
G16	3052	3052	3052	3051.3	3051.4	3051.9	3038.4
G17	3047	3047	3046	3045.5	3046.4	3046.7	3034.3
G18	992	992	991	992	992	992	974.3
G19	906	906	904	906	906	906	886.9
G20	941	941	941	941	941	941	915.1
G21	931	931	931	931	931	931	903.5
G22	13359	13359	13359	13353.5	13354.5	13349.2	13338.4
G23	13342	13342	13342	13333	13331.6	13332.1	13330
G24	13337	13333	13337	13327.3	13325.3	13321.9	13321.1
G25	13338	13339	13332	13328	13328.2	13329.2	13324.9
G26	13324	13326	13328	13313.7	13312.3	13314.8	13311.7
G27	3337	3336	3336	3327.3	3326.9	3328.2	3316.2
G28	3296	3296	3295	3286	3288.9	3287.7	3285.2
G29	3404	3405	3391	3395.2	3391.9	3390.2	3383.4
G30	3412	3411	3403	3404.6	3404.8	3405.5	3397.8
G31	3306	3306	3288	3299.7	3299.5	3300.7	3293.8
G32	1408	1410	1406	1400.9	1404.6	1401.3	1375.4
G33	1382	1382	1378	1373.9	1376.1	1373.3	1352.2
G34	1382	1384	1378	1375.4	1378.2	1376	1352.3
G35	7674	7679	7678	7663.3	7670.8	7668.8	7642.5
G36	7666	7671	7670	7653.1	7658.7	7660	7633.5
G37	7673	7682	7682	7663.3	7667.9	7670	7645
G38	7674	7682	7683	7663.4	7670.4	7671.7	7642.5
G39	2402	2407	2397	2391.3	2391.1	2395.2	2341.5
G40	2394	2399	2390	2381.2	2383.3	2387.8	2324.9
G41	2402	2404	2400	2380	2388.9	2393	2329.2
G42	2475	2478	2469	2462.3	2466.2	2467.8	2404.9
G43	6660	6660	6660	6660	6659.9	6660	6649.3
G44	6650	6650	6639	6649.9	6649.9	6650	6641.4
G45	6654	6654	6652	6653.9	6653.9	6654	6647.4
G46	6649	6649	6649	6648.2	6648.8	6649	6641.5
G47	6657	6657	6665	6656.6	6656.8	6656.9	6648
G48	6000	6000	6000	6000	6000	6000	6000
G49	6000	6000	6000	6000	6000	6000	6000
G50	5880	5880	5880	5880	5880	5880	5875
G51	3848	3848	3847	3844.6	3846.4	3846.6	3832.4
G52	3851	3851	3849	3847.6	3848.4	3849.9	3836.1
G53	3849	3850	3848	3846.9	3847.7	3848	3835.2
G54	3852	3851	3851	3848.6	3847.8	3850.1	3836.3
Matched	38	47	29	25	27	44	2

8. I Dunning, S Gupta, J Silberholz, What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO, Informs Journal On Computing, 30 (2018), 608-624.
9. F. Glover, G. Kochenberge, B. Alidaee, Adaptive memory tabu search for binary quadratic programs, Management Science, 44 (1998) 336–345.
10. F. Glover, C. Rego, B. Alidaee, G. Kochenberge, One-pass heuristic for large-scale unconstrained binary quadratic problems, European Journal of Operational Research, 137 (2002) 272–287.
11. F. Glover, Z. Lü, J.K. Hao, Diversification-driven tabu search for unconstrained binary quadratic problems, 4OR: A Quarterly Journal of Operations Research, 8 (2010) 239–253.
12. P. Hansen, N. Mladenović, Variable neighborhood search, In F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research Management Science, 57 (2003) 145–184.
13. S. Hanafi, A.R. Rebai, M. Vasquez, Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems, Journal of Heuristics, 19 (2013) 645–677.
14. C Helmberg, F. Rendl, A spectral bundle method for semidefinite programming, SIAM Journal on Optimization, 10 (1999), 673–696.
15. K. Katayama, M. Tani, H. Narihisa, Solving large binary quadratic programming problems by effective genetic local search algorithm, In Proceedings of the 2000 Genetic and Evolutionary Computation Conference, pp. 643–650, 2000.
16. K. Katayama, H. Narihisa, Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem, European Journal of Operational Research, 134 (2001) 103–119.
17. M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: iterated racing for automatic algorithm configuration, Operations Research Perspectives, 3 (2016) 43–58.
18. W. Liu, D. Wilkins, B. Alidaee, A hybrid multi-exchange local search for unconstrained binary quadratic program, Working papers series, Hearin center for enterprise science, 2006.
19. A. Lodi, K. Allemand, T.M. Liebling, An evolutionary heuristic for quadratic 0-1 programming, European Journal of Operational Research, 119 (1999) 662–670.
20. Z. Lü, F. Glover, J.K. Hao, A hybrid metaheuristic approach to solving the ubqp problem, European Journal of Operational Research, 207 (2010) 1254–1262
21. Z. Lü, F. Glover, J.K. Hao, Neighborhood combination for unconstrained binary quadratic problems, In M. Caserta and S. Voss (Eds.): MIC-2009 Post-Conference Book, pp. 49–61, Springer Berlin, 2012.
22. P. Merz, B. Freisleben, Genetic algorithms for binary quadratic programming, In Proceedings of the 1999 Genetic and Evolutionary Computation Conference, 1 (1999) 417–424.
23. P. Merz, B. Freisleben, Greedy and local search heuristics for unconstrained binary quadratic programming, Journal of Heuristics, 8 (2002) 197–213.
24. P. Merz, K. Katayama, Memetic algorithms for the unconstrained binary quadratic programming problem, Biosystems, 78 (2004) 99–118.
25. G. Palubeckis, Multistart tabu search strategies for the unconstrained binary quadratic optimization problem, Annals of Operations Research, 131 (2004) 259–282.
26. G. Palubeckis, Iterated tabu search for the unconstrained binary quadratic optimization problem, Informatica, 17 (2006) 279–296.
27. V.P. Shylo, O.V. Shylo, Solving unconstrained binary quadratic programming problem by global equilibrium search, Cybernetics and System Analysis, 47 (2011) 889–897.

28. M. de Souza, M Ritt, Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming, A. Liefooghe and M. Lopez-Ibanez (Eds.): EvoCOP 2018, LNCS 10782, pp. 67–84, 2018.
29. Y. Wang, Z. Lü, F. Glover, J.K. Hao, Probabilistic GRASP-Tabu search algorithms for the UBQP problem, Computers & Operations Research, 40 (2013) 3100-3107.
30. Y. Wang, Z. Lü, F. Glover, J.K. Hao. Path relinking for unconstrained binary quadratic programming, European Journal of Operational Research, 223 (2012) 595–604.