Knowledge-Based Systems 207 (2020) 106366

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Heuristic algorithms based on deep reinforcement learning for quadratic unconstrained binary optimization

Ming Chen, Yuning Chen *, Yonghao Du, Luona Wei, Yingwu Chen

College of Systems Engineering, National University of Defense Technology, Changsha, 410073, China

ARTICLE INFO

Article history:
Received 10 March 2020
Received in revised form 28 June 2020
Accepted 3 August 2020
Available online 8 August 2020

Keywords:
Unconstrained binary quadratic programming
Heuristic algorithm
Deep reinforcement learning
Neural network

ABSTRACT

The unconstrained binary quadratic programming (UBQP) problem is a difficult combinatorial optimization problem that has been intensively studied in the past decades. Due to its NP-hardness, many heuristic algorithms have been developed for the solution of the UBQP. These algorithms are usually problem-tailored, which lack generality and scalability. To address these issues, a heuristic algorithm based on deep reinforcement learning (DRLH) is proposed in this paper. It features in inputting specific features and using a neural network model called NN to guild the selection of variable at each solution construction step. Also, to improve the algorithm speed and efficiency, two algorithm variants named simplified DRLH (DRLS) and DRLS with hill climbing (DRLS-HC) are developed as well. These three algorithms are examined through extensive experiments in comparison with famous heuristic algorithms from the literature. Experimental results show that the DRLH, DRLS, and DRLS-HC outperform their competitors in terms of both solution quality and computational efficiency. Precisely, the DRLH achieves the best-quality results, while DRLS offers a high-quality solution in a very short time. By adding a hill-climbing procedure to DRLS, the resulting DRLS-HC algorithm is able to obtain almost the same quality result as DRLH with however 5 times less computing time on average. We conducted additional experiments on large-scale instances and various data distributions to verify the generality and scalability of the proposed algorithms, and the results on benchmark instances indicate the ability of the algorithms to be applied to practical problems.

1. Introduction

Nonlinear integer programming is one of the most difficult fields in mathematical programming and operations research. As a typical nonlinear integer programming problem, the unconstrained binary quadratic programming (UBQP) has been intensively studied over the years.

The objective function value (OFV) of the UBQP can be expressed by:

$$Maximize: f(x) = x^T Qx = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j, x_i \in \{0, 1\}, \forall i = 1, 2, \ldots, n \quad (1)$$

where x is the column n-vector of binary (0-1) variables, x^T is the transpose of x and $Q = q_{ij}$ is a $n \times n$ symmetric matrix of constants.

A large number of combinatorial optimization problems can be modeled as the UBQP, such as the set partitioning problem [1,2] and graph coloring problems [3]. The UBQP can be used to model many real-world applications including traffic management, financial analysis, economics, computer science, and machine scheduling. Hence, it is significant to develop proper and effective algorithms for addressing the UBQP problem.

The commonly used algorithms for UBQP problems can be briefly divided into two categories: (1) Exact algorithms, such as branch and bound, and branch and cut [4,5]. Since the exact algorithms are acknowledged to be time-consuming in large-scale optimization, they are not able to obtain satisfying solutions within limited computing time. (2) Heuristic algorithms, such as hill climbing (HC) [6], variable neighborhood search (VNS) [7], tabu search (TS) [8–13], simulated annealing (SA) [14,15], scatter search (SS) [16], ant colony optimization (ACO) [17–19], evolutionary algorithm (EA) [20,21], memetic algorithm (MA) [22,23] and hybrid algorithms [24,25]. These heuristic algorithms can obtain near-optimal solutions within an acceptable time for large-scale UBQP instances.

Regarding those heuristic algorithms, on the one hand, they often lack generality and scalability [26]. Indeed, their performance is usually encouraging for some particular instances, while unsatisfactory for the other. On the other hand, these algorithms

* Corresponding author.
E-mail addresses: chenming_nudt@163.com (M. Chen), cynnudt@hotmail.com (Y. Chen), duyonghao15@163.com (Y. Du), wlnelysion@163.com (L. Wei), ywchen@nudt.edu.cn (Y. Chen).

https://doi.org/10.1016/j.knosys.2020.106366
0950-7051/

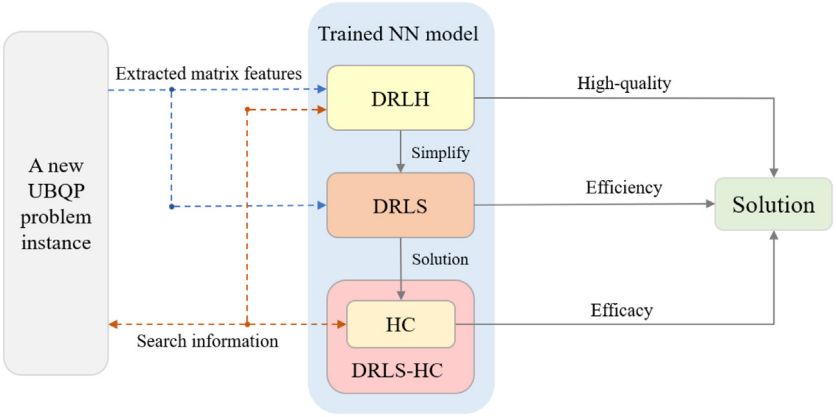

Fig. 1. The graphical abstract.

often require an intensive computational overhead, which is not suitable for practical use.

To address these issues, we resort to neural networks and reinforcement learning. With the development of deep reinforcement learning (DRL) and parameter optimizer [27–33], the training of large-scale model becomes feasible. Some DRL based methods were studied for classical combinatorial optimization problems. Oriol Vinyals et al. [34] proposed a pointer network, which can output satisfactory solutions of the computing planar convex hulls, Delaunay triangulations, and the symmetric planar traveling salesman problem (TSP). Irwan Bello et al. [35] proposed a framework to tackle TSP problems using neural networks and reinforcement learning. MohammadReza Nazari et al. [36] presented an end-to-end framework for the vehicle routing problem (VRP). These studies have shown the ability of the DRL to self-obtain complex heuristic rules and guide the optimization searching when solving a problem. However, we find that the previous problems they addressed are all linear, and whether the DRL can be applied the nonlinear UBQP problems has not been studied.

In this paper, we select the diagonal elements and sum of row vectors as the effective feature engineering and propose a heuristic algorithm based on deep reinforcement learning (DRLH) to solve the UBQP. Specifically, we employ an Actor–critic algorithm to train a neural network called NN that, given a set of UBQP variables, predicts a distribution over different orders of setting variables from 0 to 1. This neural network serves as a complex (black-box) heuristic rule which guild the solution construction at each step of our proposed DRLH algorithm. To further improve the algorithm effectiveness, two DRLH variants named simplified DRLH (DRLS) and DRLS with hill climbing (DRLS-HC) are also developed. We carried out extensive experiments to evaluate the performance of the proposed algorithms. Computational results indicate that the proposed DRLH presents better performance than other well-known heuristic algorithms in the literature. As a simplified version, DRLS is able to obtain slightly worse solutions with significantly reduced computing time. By including a hill-climbing procedure, DRLS-HC produces even better performance than DRLH with much less computational expense, which shows an interesting balance between solution quality and computational efficiency. Also, the results on sensitivities and benchmarks show a good generalization and scalability of our proposed algorithms in different instance size and distributions.

The remainder of this paper is organized as follows. In Section 2, the algorithm framework of DRLH, the structure of the NN model, and two variants of the DRLH for UBQP are given. In Section 3, computational studies are presented. Finally, conclusions are drawn in Section 4.

2. The DRLH

In this section, we present the general framework of the proposed DRLH algorithm and its key ingredients including the structure of the neural network model and the method used for training the parameters of the NN model. Finally, two DRLH variants, DRLS and DRLS-HC, that strengthen the solution efficiency and solution quality are described. The relationships of proposed algorithms and their characteristics are shown in Fig. 1. Overall, the DRLH obtains high-quality results, the DRLS produces a good solution in a very short time and the DRLS-HC approaches the DRLH solutions with the fastest computation speed.

2.1. Framework

Algorithm 1 shows the general framework of the proposed DRLH. The algorithm is essentially a construction algorithm. It first zeros all the variables in the initial solution. At each step, a trained NN model is used to identify a 0-value variable which is considered the most suitable to be included in the current solution and set the variable to be 1. The construction process proceeds until the solution can be no longer improved.

Algorithm 1 The general framework of DRLH

Input:
Strategy set: the trained DRL model;
Initialize the solution $X_t = \{x_1, x_2, \cdots, x_n | x_i = 0; i = 1, 2, \cdots, n\}$;
while True **do**
  Candidate $X_{candidate} \leftarrow X_t$
  Candidate variable set $C_t \leftarrow$ all variables $x \in X_{candidate}, x = 0, x \notin mask$ /* the *mask* in Section 2.2.4 */
  if $C_t = \emptyset$ **then**
    break
  end if
  Select a variable $x_j \in C_t$ according to the NN model /* Section 2.2 */
  Set x_j in $X_{candidate}$ to 1
  if $f(X_{candidate}) > f(X_t)$ **then**
    $X_t \leftarrow X_{candidate}$
  else
    break
  end if
end while
return X_t and $f(X_t)$

It is clear that the solution process of the DRLH can be viewed as a Markov's decision process. The next variable to be set 1 is determined by the current state at each step. The inner working logic is explained as follows.

In the UBPQ problem, we are given a $n * n$ matrix Q and a set of binary decision variables $X = \{x_1, x_2, \ldots, x_n\}$.

Given current solution X_t and the NN model, we set the next variable x_{t+1} to 1 at the current problem state. We use the chain rule of probability to decompose the probability of generating sequence $X = \{x_t, t = 0, 1, \ldots, T'\}$ in T''s iteration, as following:

$$P(X|Q) = \prod_{t=0}^{T'} P(x_{t+1}|X_t, Q) \tag{2}$$

$$X_{t+1} = f(x_{t+1}, X_t) \tag{3}$$

In this manner, we can get the probability distribution for potential variables to be set 1. Let us denote the parameters of our model as θ, the formulas above can be transformed to:

$$P(X|Q;\theta) = \prod_{t=0}^{T'} p(x_{t+1}|X_t, Q;\theta) \tag{4}$$

Parameters θ of the DRL model are obtained by maximizing the conditional probability of the training set:

$$\theta^* = \arg\max \sum_{X,Q} \log p(X|Q;\theta) \tag{5}$$

θ should be set to an appropriate value say θ^* to ensure that the model fits the UBQP well. This can be done by using a large amount of data to train the model such that $\theta = \theta^*$. In what follows, we present the structure of the NN model and the training method based on deep reinforcement learning.

2.2. The NN model

The proposed NN model consists of two parts: (1) an encoder–decoder structure well establishes the relations between input and output, which corresponds to inputting a set of potential variables and outputting a variable with the highest probability (which is considered the most suitable one) in UBQP problem. (2) an attention model that comprehensively considers the relationship of the input variables in the encoder–decoder and gives different degrees of attention to them.

2.2.1. Encoder

As a basis of network input, feature extraction determines the static information from the problem. In the UBQP problem, through the effective feature engineering, we extract two features from matrix Q as the inputs to the network, including (1) major diagonal elements of Q, and (2) sum of row vectors of Q, and ignore other features like matrix eigenvalues, eigenvectors, etc. We regard these two feature variables as the UBQP static parameters and denote them as $s^i = \{diag^i, sr^i\}$.

Since the outputs of the network should be independent of the order of data input, traditional recurrent neural networks (RNNs) are not qualified to be our encoder. We propose to use an embedding layer, which encodes the input to a code vector and decreases the encoding complexity and computing time. Specifically, the one-dimensional convolution is used to encode the input parameters to a high-dimensional vector that shares across all input steps. The number of in-channels equals the number of characteristic variables. We set the numbers of channels and filters in the encoder to be 2 and D (according to preliminary experimental experience).

In the embedding layer, each h^i is independent of the other, but all these variables cannot reflect the relationship among static features. For this reason, we need a structure to collect this information. Inspired by Ref. [37], we find that the simple multi-head attention can address this issue well. Therefore, we create a new variable of G based on the encoder. It is an extension of static information after the one-dimensional convolution, as shown in Fig. 2. The G can be seen as a global variable for the problem, and it contains all the interactive information of static inputs.

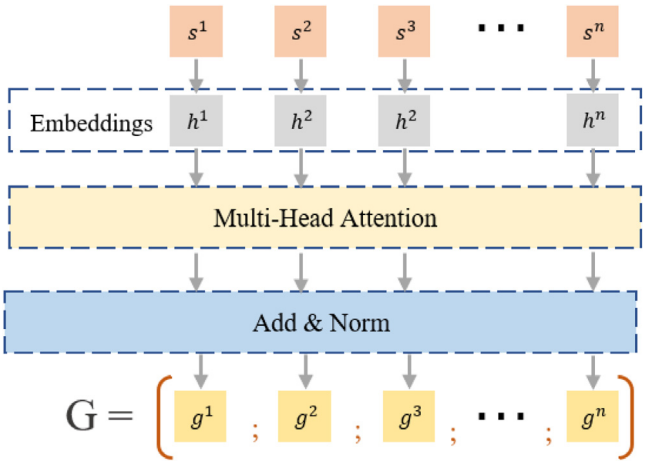

Fig. 2. The structure of G.

2.2.2. Decoder

It is easy to understand that former selections during the construction process of our DRLH affect the later decisions. Hence, it is necessary to take the former selections into account. Also different from the Long Short-Term Memory [38] (LSTM) -based decoder in Ref. [36], the information of former decisions which is useless for solving UBQP, we do not use the LSTM in our decoder. To include more information in our decoder, we set up a dynamic parameter $d_t^i = \{state_t^i, change_t^i\}$ for each variable. The $change_t^i$ parameter is the increment of OFV ($IOFV$) after x_i is included in the current solution. $state_t^i$ is a label that indicates whether x_i can be selected or not. We simply use an embedding layer that takes the dynamic parameter $d_t^i = \{state_t^i, change_t^i\}$ as the input.

2.2.3. Attention

Our brain often only pays attention to a few particularly important parts of our observations, which lead us to obtain the required information and understand the environment. The brain also learns to combine these parts. The attention mechanism follows this principle. In our NN model, the attention layer is attached by the encoder and decoder. It obtains the correlation degree of the candidate variables to be selected. This correlation is reflected by the probability distribution, and the model gives higher attention to the variables with high correlation.

In our NN model, we propose an attention with two layers. The first layer collects the dynamic information and the second one combines static and dynamic information. The two layers of the NN model are shown in Fig. 3 and Fig. 4.

In Fig. 3, the encoder embedding layer lies on the left while the decoder embedding layer is on the right storing the current information of the decoded sequence. Considering the interrelationship among static variables, the variable G is the input of the first attention layer. In decoding step t, we use a dynamic environment variable e_t to extract relevant information from G. In fact, e_t shows the relevance between the inputs and the next decoding. Here, we represent the embedded static input and dynamic input by $\bar{s}^i$ and $\bar{d}_t^i$, respectively. Then environment

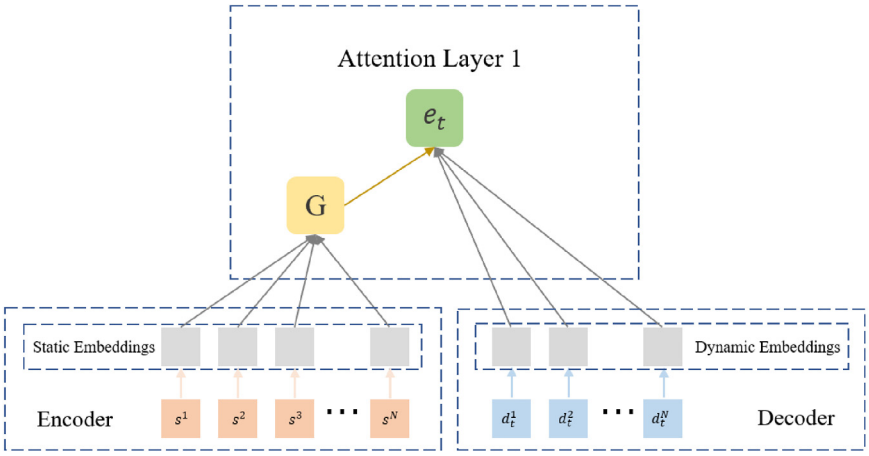

Fig. 3. The first attention layer.

variable e_t can be deduced by the softmax function as follows:

$$u_t^i = v_1^T \tanh(W_1[g^i; \bar{d}_t^i]), g^i \in G \tag{6}$$

$$e_t = (e_t^1, e_t^2, \ldots, e_t^n) = \text{softmax}(u_t) \tag{7}$$

where $[\bar{s}_t^i; h_t]$ means to concatenate the two vectors together.

After obtaining vector e_t, we put it into the second attention layer. Because e_t represents the relationship between each input and the output (best) variable, we obtain a context vector c_t by calculating e_t^i and $\bar{s}^i$:

$$c_t = \sum_{i=1} e_t^i \bar{s}^i \tag{8}$$

Context variable c_t represents the weighted context vector of each x_i.

After the following matrix operation, we normalize the $\tilde{u}_t$ values as follows.

$$\tilde{u}_t^i = v_2^T \tanh(W_2[\bar{s}^i; \bar{d}_t^i; c_t]) \tag{9}$$

$$P(x_{t+1}|X_t, Q) = \text{softmax}(\tilde{u}_t^i) \tag{10}$$

where v_1^T, W_1, v_2^T and W_2 are the network parameters to be trained.

In our DRLH algorithm, a mask mechanism is employed to fix the selection probability to 0 if the variable has been selected.

2.2.4. Selection strategy

With Eq. (10), we can get the probability distribution of each candidate variable before selection. A strategy is needed for variable selection based on candidate probabilities. In this paper, to enable good generalization of our network, and to obtain a stable and high-quality solution, we apply a random strategy and a greedy strategy for training and testing, respectively.

2.3. Model training method

We apply an Actor–Critic algorithm [39] based on the policy gradient method to train our model, which is presented in Algorithm 2

There are two parts needed to be trained in the Actor–Critic algorithm. The first is an Actor network, which is the origin of our NN model and aims to obtain UBQP solutions. The second is a Critic network, which estimates the return expectation on the basis of our problem case and is used for parameter update in the Actor network. In our experiments, these two networks use the same embedding layer. The Critic network is only inputted by static variables. Also, the Critic network has two hidden layers: one dense layer with ReLU activation whose output is a vector and another linear one whose output is a single value.

Algorithm 2 Actor–critic training

Initialize actor network params θ
Initialize critic network params θ_c
for *iterarion* $\leftarrow 1, 2, \cdots$ **do**
  reset gradients: $d\theta \leftarrow 0$, $d\theta_c \leftarrow 0$
  generate N problem instances from ϕ
  for $k = 1, 2, \cdots n$ **do**
    step counter $t \leftarrow 0$
    while not terminated **do**
      select next variable x_{t+1}^k according to the probability distribution $P(y_{t+1}^k|Y_t^k, X_t^k)$
      update new state X_{t+1}^k to X_t^k
      $t \leftarrow t + 1$
    end while
    calculate *OFV* reward R^k
  end for
  $d\theta \leftarrow \frac{1}{N}\sum_{k=1}^{N}(R^k - V(X_0^k; \theta_c))\nabla_\theta \log P(Y^k|X_0^k)$
  $d\theta_c \leftarrow \frac{1}{N}\sum_{k=1}^{N}\nabla_\theta(R^k - V(X_0^k; \theta_c))^2$
  update θ using $d\theta$ and θ_c using $d\theta_c$
end for

2.4. DRLH variants

Since the DRLH needs to calculate the *IOFV* of each candidate solution before updating the current solution, the algorithm is time consuming. To reduce the computing overhead, a DRLH variant algorithm named DRLS is proposed. The difference of DRLS with respect to the DRLH is that the d_t^i in the NN model of the DRLS only considers parameter $state_t^i$, which means the DRLS solutions are directly obtained by the features of the UBQP's matrix Q itself. Since there is no search process in the DRLS, it can be viewed as a complex black-box heuristic rule which is very efficient.

To balance the solution quality and computational efficiency, a hybrid algorithm that combines DRLS and a hill climbing algorithm (DRLS-HC) is proposed. Specifically, the DRLS is used to obtain the initial solution of the hill climbing algorithm. HC always selects the candidate solution with the largest *IOFV* at each iteration, and the pseudo-code of HC is shown in Algorithm 3.

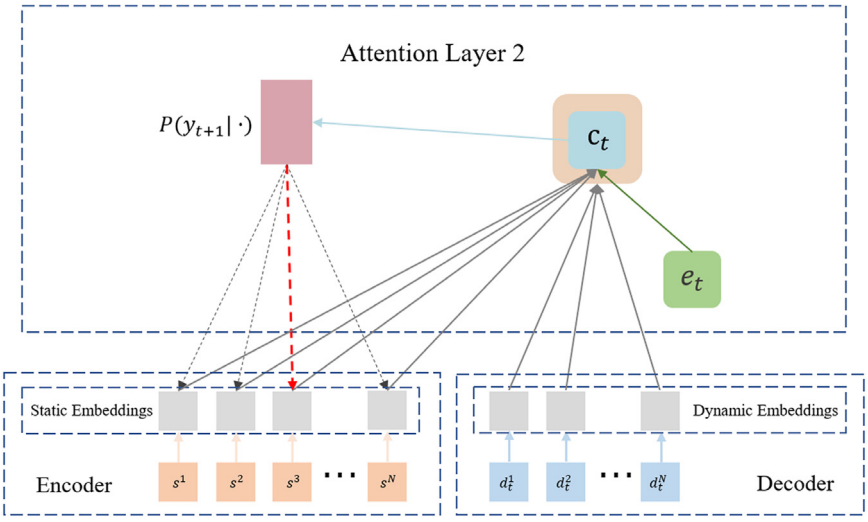

Fig. 4. The second attention layer.

Algorithm 3 HC Algorithm

```
Input solution X_t = {x_1, x_2, ..., x_n};
while True do
    IOFV_t ← the positive IOFV set of all variables x ∈ X_candidate, x = 0
    if IOFV_t = ∅ then
        break
    end if
    select the biggest IOFV* with its corresponding variable x* in IOFV_t
    set x* in X_candidate to 1
end while
```

3. Experimental study

In this section, the experiment setup is introduced first, including the dataset for model training, the competitors and the algorithm parameter setting. Then, a series of experiments and the results are presented to examine the performance of the proposed DRLH, DRLS, and DRLS-HC.

3.1. Experiment setup

3.1.1. Dataset

Considering the large amount of the data required for the NN model training, existing benchmarks are not enough for the training purposes. We thus propose an instance generator as follows. In the UBQP problem, the element of Q can be positive or negative, so we abstract the problem and set the elements in Q within $(-1, 1)$. The value of each element in Q follows the specific distribution Φ. To make the DRLH more general, we use the uniform random distribution to generate datasets in the experiments.

The termination time in different DRLH and DRLS cases are not the same. Especially in batch training, it is not clear at what time a standardized output matrix can be obtained. Therefore, we add a variable x_0 valued 0 at the initial solution per case and add an all-zero row and column vectors to the Q matrix each case. This can make the current case stay at the decoding point x_0 while the current case should stop but the other cases in the batch have not been completed. No matter the value of x_0, the final *OFV* of the problem will not be affected.

3.1.2. Competitors

To examine the performance of the proposed DRLH, DRLS and DRLS-HC, we employ the following three competitors:

To examine the adaptability of the DRLH, we used three heuristic algorithms: the first acceptance heuristic algorithm (FA), HC and Greedy (GY). HC is mention in Section 2.4. The GY is from Ref. [40]. We zeroed the initialize solutions of these three competitors. It can be seen that the algorithms in this part are all based on the information of *IOFV*.

We compare DRLS with three construction algorithms: Rand, Diag, and SR. The Rand algorithm randomly select a subset of all variables from $X = \{x_1, x_2, \ldots, x_n\}$ and set them to 1. The Diag algorithm sets the variable $x_i \in X$ with a positive diagonal elements $q_{i,i}$ in Q in Eq. (11) to be 1. The SR algorithm takes the sum of the row vectors of matrix Q as features, and row_i to present the result of the ith row vectors. Given each vector row_i, we set the solution by the Eq. (12) shows. Also to evaluate the performance of variable G, based on the DRLS, we trained an algorithm named DRLS-G without G. We only test the DRLS-G on instances with 50 variables, using the same training data and epoch setting as the DRLS.

$$x_i = \begin{cases} 1, & q_{i,i} \geq 0 \\ 0, & q_{i,i} < 0 \end{cases} \tag{11}$$

$$x_i = \begin{cases} 1, & row_i \geq 0 \\ 0, & row_i < 0 \end{cases} \tag{12}$$

To examine the performance of the DRLS-HC, we designed its three competitors including Rand with hill climbing (R-HC), Diag with hill climbing (D-HC) and SR with hill climbing (SR-HC). In addition, to obtain a high-quality solution, we also combine DRLH and HC, resulting in an algorithm called DRLH-HC. All the HC in these algorithms have the same procedures of solution.

3.1.3. Algorithm parameter

To train the network parameters θ and θ_c in our Actor–Critic network, we generated N instances with the Φ distributing mentioned above. For θ and θ_c, the Xavier initialization strategy [41] are employed.

The parameter settings of the NN model are shown in Tables 1 and 2. We applied the Adam optimizer [27] with a learning rate η of 0.0005 to train both Actor and Critic networks.

Table 1
Main parameters of the Actor network.

Actor network
Global: Q, K, V-dim = 256, Head = 8, Layers = 3
Encoder: Conv-1D($D_{inputsize}$, Filter = 128, kernel size = 1, stride = 1)
Decoder: Conv-1D($D_{inputsize}$, Filter = 128, kernel size = 1, stride = 1)

Table 2
Main parameters of the Critic network.

Critic network
Encoder: Conv-1D($D_{inputsize}$, Filter = 128, kernel size = 1, stride = 1)
Conv-1D(256, Filter = 20, kernel size = 1, stride = 1)
Conv-1D(20, Filter = 20, kernel size = 1, stride = 1)
Conv-1D(20, Filter = 1, kernel size = 1, stride = 1)

Table 3
The results of different algorithms.

Method	50		100		150		200	
	OFV	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time
FA	96.88	10.90	272.81	34.46	501.96	114.85	769.99	188.53
HC	107.51	35.23	311.91	130.56	577.45	435.53	894.26	698.02
GY	108.89	34.58	314.99	120.09	580.65	397.44	898.17	624.03
DRLH-50	116.85	89.44	**334.37**	314.48	615.82	722.32	950.21	1198.82
DRLH-100	116.45	88.19	334.26	309.73	**616.14**	734.38	**952.05**	1182.92
DRLH-150	116.89	89.69	334.17	315.15	615.27	731.36	949.73	1201.38
DRLH-200	**116.90**	89.55	334.41	314.39	615.75	712.54	950.29	1198.66

All the experiments were conducted using a single GPU RTX 2080-Ti, and a i9-9900 K CPU with 64GB RAM. We used the deep learning framework embedded in Pytorch 1.02 coded in Python 3.7.

3.2. Experimental results

For the test data, we generate 1,000 instances with 50, 100, 150 and 200 variables respectively according to the ϕ distribution which is the same as the training data. For each instance size, we use the average *OFV* and the total running time of the 1000 instances to evaluate our model. The batch size is set to 1.

3.2.1. Results of DRLH

To validate the generalization capabilities of the DRLH, we train 50-,100-,150-,200-size DRL model for DRLH. Since the training of DRL is computationally expensive, we generated 20,000 training data for each instance size and each model was trained 2 epoch. The batch size is set depending on the GPU memory.

The average *OFV* and total computing time of these methods are shown in Table 3. It is clear that DRLH has achieved the best result on each instance size. Obviously, the two feature information extracted from Q, *diag* and *sr*, play a vital role in guiding the selection and have been effectively utilized by the NN model of DRLH. Besides, as it shows in Fig. 5, although the NN in DRLH is obtained under 50-size training, it is able to handle 200-size instance as well. It is worth noting that there is no fundamental difference between different size NN models at the same instance size. This means under the current instance size, data distribution and problem background, the DRLH has achieved a good generalization capability. As it shows in Fig. 6, in addition to FA, the computing time of the other algorithms is almost on the same level. Moreover, the computing time trend of DRLH is similar to its competitors, but the computing time of DRLH is slightly slower because of the variable selection in DRLH is through the NN model. Simultaneously, the selection procedure in its competitors are rules that do not need calculation, or the computation is minimal. However, although the trained NN model parameters are constant, they are enormous.

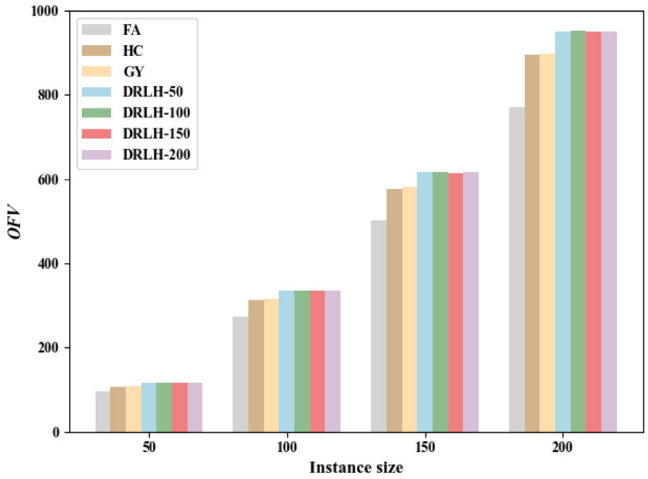

Fig. 5. The average *OFV* of different algorithms.

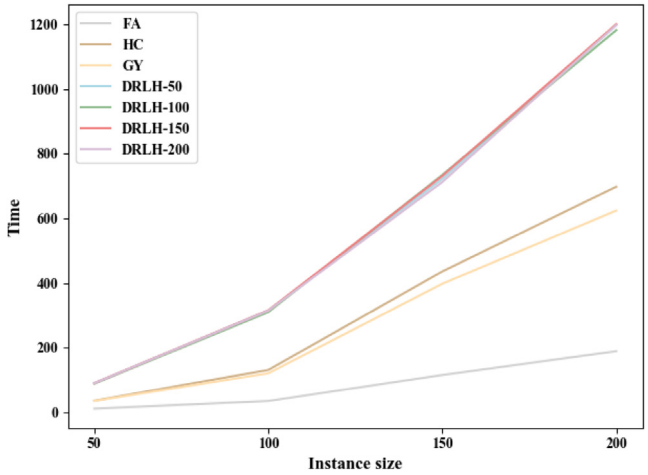

Fig. 6. The total computing time (s) on 1000 instances of different algorithms.

Table 4
The results of different methods.

Algorithms	50		100		150		200	
	OFV	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time
Rand	0.38	0.05	0.51	0.05	1.62	0.12	0.30	0.10
Diag	11.40	0.03	24.28	0.03	36.93	0.05	53.23	0.08
SR	79.82	0.01	228.25	0.01	422.18	0.01	654.61	0.07
DRLS-G-50	96.08	11.07	262.82	17.73	471.24	32.31	719.82	40.23
DRLS-50	96.08	10.65	263.02	17.98	473.21	31.05	721.82	41.29
DRLS-100	**96.72**	10.99	**277.16**	18.42	469.10	42.93	719.95	51.89
DRLS-150	95.96	10.71	262.87	17.95	**474.06**	32.83	720.01	40.46
DRLS-200	95.94	10.64	264.45	18.10	493.90	36.59	**766.31**	49.70

3.2.2. Results of DRLS

Also to validate the generalization capabilities of the DRLS, we trained 50-,100-,150-,200-size NN model for DRLS. Different from the DRLH, the training of DRLS is much cheaper. 100,000 training instances are generated and the model was trained 10 epochs. The batch size of each size model is also determined by the GPU memory.

The average *OFV* and total computing time (s) of these methods are shown in Table 4. Apparently, the DRLS has achieved the best result on each instance size among these heuristic rules, which illustrates the significant role of feature engineering in this algorithm and the effective use of features by DRLS. It is

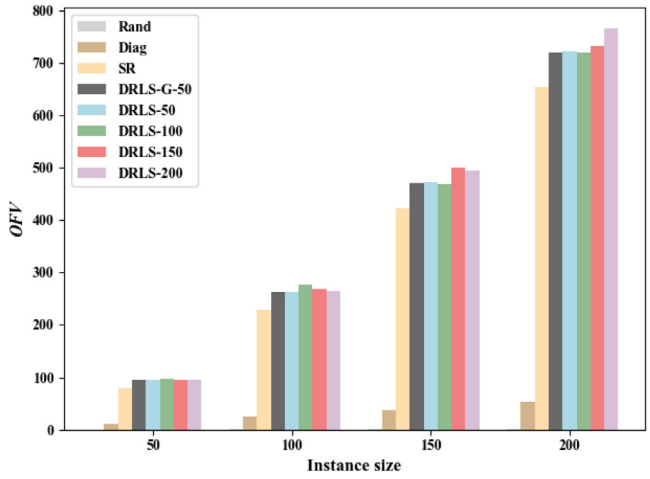

Fig. 7. The average *OFV* of different algorithms.

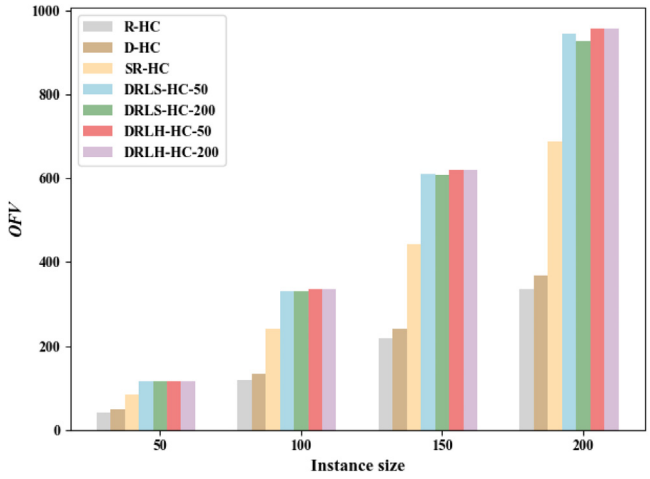

Fig. 9. The average *OFV* of different algorithms.

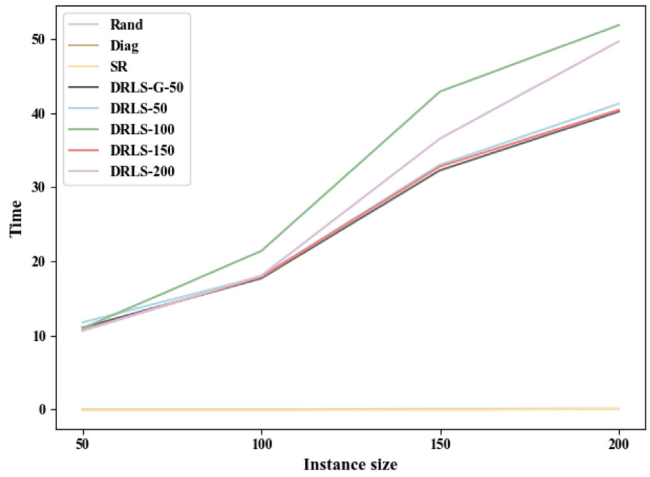

Fig. 8. The total computing time (s) on 1000 instances of different algorithms.

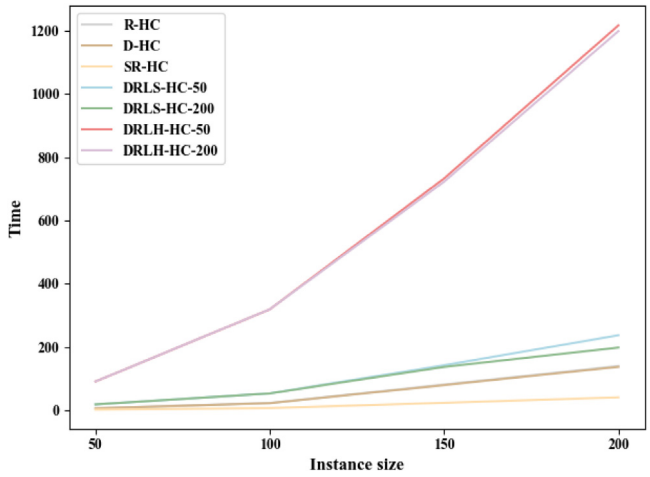

Fig. 10. The total computing time (s) on 1000 instances of different algorithms.

worth noting that these two feature-related algorithms Diag and SR have achieved positive *OFV*, especially SR has found a feasible solution. As it shows in Fig. 7, the DRLS has achieved a good generalization ability. But different from DRLH, it is worth mentioning that NN models of DRLS we trained on different sizes have performed better on the specific instance size. For example, DRLS-200 brings better results at instance size 200 than other DRLS size algorithms. In addition, from Table 4 shows, it can be seen that after adding the variable *G*, the performance is improved.

As can be seen from Fig. 8, all these algorithms have extremely short computing time. Moreover, the computing time of DRLS has an approximately linear relationship with the scale of the problem, which has indicated its advantages when dealing with large-scale problems.

3.2.3. Results of DRLS-HC

The results are shown in Table 5. From Fig. 9, we can see that DRLH-HC has achieved the best *OFV* under all instance sizes. It should be noted from Figs. 9 and 10 that the DRLS-HC has achieved the approximate best *OFV* with a huge advantage in time consumption, and this trend becomes more apparent when the instance scale increasing. It means that in real-applications that require high timeliness, DRLS-HC can obtain a high-quality solution quickly. So it can be clearly found through experiments that the efficiency of DRLH is greatly improved.

The reason for such high-quality results is that, in the solution of DRLS, each variable that has been set 1 will increase the OFV. Different from other competitors, the DRLS is one-way construction. After applying a HC, it can obtain a good solution with a low computational cost.

3.3. Large-scale experiments

To examine the performance of proposed algorithms at large-scale problems, we generated 10 instances of each 500-, 1000-, 1500-, and 2000-size based on the distribution Φ. The batch size is also set to 1 when examining.

The average *OFV* and total computing time (s) of these methods are shown in Table 6. In the single heuristic algorithms, DRLH has obtained the best solution but followed the biggest computational time, and DRLS can find a feasible solution in a very short time, but the quality of the solution is unsatisfactory.

In the hybrid heuristic algorithms, obviously, DRLH-HC has obtained the optimal solution among all the algorithms but also with the longest computing time. It is worth noting that, compared to DRLH, the improvement of DRLH-HC is tiny and the computational cost increment is very low. This fully indicates that the DRLH has excellent performance for solving the UBQP problems. For DRLS-HC, the quality of its solution is very close to DRLH-HC but the computing time is much shorter.

Table 5
The results of different algorithms.

Algorithms	50		100		150		200	
	OFV	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time
R-HC	42.15	6.78	118.45	23.29	218.31	81.34	336.06	140.28
D-HC	48.37	6.35	134.52	22.64	241.96	79.46	369.32	137.60
SR-HC	84.15	2.46	240.19	7.22	444.04	23.77	688.54	41.05
DRLS-HC-50	115.59	18.73	331.50	53.53	611.30	142.70	944.53	237.05
DRLS-HC-200	115.62	18.79	331.41	53.31	606.80	137.43	927.71	198.31
DRLH-HC-50	117.59	91.08	336.70	319.22	**620.28**	732.85	**957.57**	1216.74
DRLH-HC-200	**117.62**	91.16	**336.78**	319.41	620.11	722.60	957.46	1198.66

Table 6
The results of different algorithms in large instance scale.

Algorithms	500		1000		1500		2000	
	OFV	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time
FA	2938.45	9.91	8691.84	35.15	15542.45	83.87	24633.99	190.08
HC	3463.73	48.68	10165.56	173.05	18467.85	409.56	28961.36	910.23
GY	3473.33	41.47	10208.91	147.58	18503.25	354.50	29069.54	825.69
R-HC	1320.59	10.63	3713.33	29.12	6894.55	69.99	10633.75	159.56
D-HC	1385.25	11.89	4101.79	29.51	7299.89	71.78	10916.57	157.45
SR-HC	2615.80	8.05	7961.00	8.42	14039.26	20.55	22279.30	42.57
DRLH-50	3647.68	81.23	10739.85	305.54	19489.34	745.80	30447.02	1355.13
DRLH-200	3632.29	71.30	10707.33	306.10	19390.70	744.65	30324.48	1341.31
DRLH-HC-50	**3686.18**	82.74	10837.00	310.76	**19684.60**	759.36	**30749.83**	1381.51
DRLH-HC-200	3668.73	72.13	**10825.51**	311.66	19618.03	758.77	30699.39	1370.48
DRLS-50	2513.87	1.24	7321.11	1.62	12960.33	2.55	20992.66	3.02
DRLS-200	2514.62	0.92	7380.45	1.59	13304.86	2.61	21156.10	2.95
DRLS-HC-50	3638.65	2.90	10731.88	58.29	19475.10	149.28	30454.81	266.70
DRLS-HC-200	3638.65	2.59	10720.99	59.37	19451.47	143.22	30430.64	284.59

Table 7
The results of different algorithms in different distributions.

Algorithms	D0		D1		D2		D3		D4	
	OFV	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time	*OFV*	Time
Diag	11.40	0.03	20.20	0.03	7.57	0.03	4.68	0.03	1.10	0.03
SR	79.82	0.01	141.60	0.01	67.23	0.01	51.09	0.01	25.26	0.01
FA	96.88	10.90	170.39	12.23	81.88	14.24	62.35	14.71	27.42	17.48
HC	107.51	35.23	188.27	39.47	90.50	44.85	68.02	44.43	27.95	37.44
GY	108.89	34.58	190.93	38.93	91.78	43.60	69.19	42.18	28.14	32.78
R-HC	42.15	6.78	70.98	7.82	35.24	8.65	26.14	8.35	11.86	7.78
D-HC	48.37	6.35	84.41	7.24	56.27	13.47	55.87	20.08	31.42	27.70
SR-HC	84.15	2.46	148.48	2.66	70.59	2.95	53.56	2.98	26.25	2.63
DRLH	116.90	89.55	204.19	88.30	98.04	91.89	73.98	90.60	32.39	81.14
DRLH-HC	**117.62**	91.16	**205.43**	89.92	**98.65**	93.58	**74.31**	92.09	**32.54**	82.33
DRLS	95.94	10.64	168.91	12.63	80.37	10.63	60.18	10.29	27.69	9.57
DRLS-HC	115.62	18.79	202.42	22.06	96.98	18.83	73.31	18.76	35.15	19.17

3.4. Sensitivity experiments

To examine the sensitivity to data distribution of proposed algorithms, we tested with four different distributions as follow:

Distribution 0 (D0): the uniform random distribution;

Distribution 1 (D1): standard normal distribution;

Distribution 2 (D2): the uniform random distribution but the elements in Q has a probability of 0 for 30%;

Distribution 3 (D3): the uniform random distribution but the elements in Q has a probability of 0 for 60%;

Distribution 4 (D4): the uniform random distribution but the elements in Q has a probability of 0 for 90%;

The average *OFV* and total computing time (s) of these methods are shown in Table 7. Apparently, as shown in Fig. 11, data distributions have little impact on the performance of our proposed algorithms, which outperformed in all tests. Besides, DRLS has approached the solutions of the local search FA and achieved better solution quality in distribution D4. It is clear that the strong generality of our proposed algorithms in different distributions mainly because of the effective features extracted from the problem. As shown in Table 7, the two construction algorithm Diag and SR, separately based on feature *diag* and *sr*, have achieved the positive *IOFV*, and notably, the SR achieved a feasible solution in all five distributions. Therefore, the trained DRLS and DRLH have taken full advantage of *diag* and *sr* to ensure scalability and high-quality solutions.

3.5. Benchmark experiments

To examine the applicability of algorithms to real-life conditions, we tested UBQP benchmarks composed of 1 instance with a size of 50, 100, 250, 500, 1000, 2500 from ORLIB [42].

The average *OFV* and total computing time (s) of these methods are shown in Tables 8 and 9. The performance of proposed algorithms has a similar advantage as they did in other experiments. It is worth noting that the generation of the benchmark instances also follows a specific data distribution. In these experimental instances, the most noticeable trait is that nearly 95% of the Q's elements are 0. As shown in the previous experiment, data distributions have little impact on the performance of our proposed algorithms, so the results on benchmarks are justified.

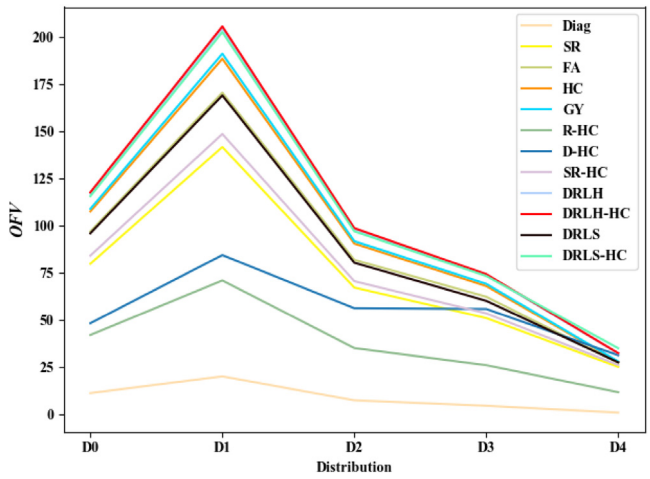

Fig. 11. The average *OFV* of different algorithms on different distributions.

Table 8
The results of different algorithms in benchmarks.

Algorithms	bqp50.1		bqp100.1		bqp250.1	
	OFV	Time	*OFV*	Time	*OFV*	Time
Diag	687	0.00	2773	0.00	−3062	0.00
SR	3057	0.00	9834	0.00	26,001	0.00
FA	3287	0.02	11,117	0.06	33,270	0.33
HC	3103	0.03	11,421	0.19	37,833	1.14
GY	3367	0.03	12,233	0.17	36,193	0.96
R-HC	408	0.01	4199	0.03	12,870	0.26
D-HC	3391	0.02	10,936	0.09	37,708	0.79
SR-HC	3065	0.00	9834	0.00	27,007	0.05
DRLH	3385	0.25	12,479	0.47	39,288	2.41
DRLH-HC	**3393**	0.25	**12,547**	0.47	**39,786**	2.46
DRLS	3385	0.01	11,297	0.01	25,109	0.04
DRLS-HC	**3393**	0.02	12,359	0.03	39,309	0.40

Table 9
The results of different algorithms in benchmarks.

Algorithms	bqp500.1		bqp1000.1		bqp2500.1	
	OFV	Time	*OFV*	Time	*OFV*	Time
Diag	19,115	0.00	31,151	0.00	26,8791	0.01
SR	88,833	0.00	257,314	0.00	1,093,556	0.00
FA	108,713	0.85	292,269	3.24	1,191,303	37.42
HC	122,538	3.46	328,580	14.81	1,407,737	184.85
GY	123,601	3.07	324,939	13.04	1,384,640	164.23
R-HC	52,236	0.57	124,983	2.66	491,316	30.29
D-HC	124,568	2.26	323,408	8.60	1,325,947	103.58
SR-HC	94,790	0.27	266,599	0.76	1,137,312	8.43
DRLH	128,567	7.42	344,798	31.29	1,450,217	329.77
DRLH-HC	**129,550**	7.55	**348,296**	31.67	**1,462,856**	335.29
DRLS	98,400	0.09	284,659	0.47	1,103,421	0.69
DRLS-HC	128,377	1.12	340,403	5.31	1,437,917	51.26

4. Conclusion and future directions

Common heuristic rules are difficult to obtain good solutions of general UBQP problems, especially when addressing large-scale cases. Inspired by the DRL based studies, the DRLH algorithm based on a NN model including an embedding layer and a two-layer attention network is proposed in this paper. Also, to further improve the algorithm speed and efficiency, two DRLH variants named the DRLS and DRLS-HC are developed, respectively.

Experiments on the test data we generated show that our algorithms can obtain competitive results. The DRLH obtains high-quality results, DRLS produces a feasible solution in very short time, and the DRLS-HC approaches the DRLH solutions with the fastest computation speed. Our algorithms show greater outperformance when addressing large-scale cases. In addition, the NN models in our DRLH and DRLS were trained with data in different sizes, but they show good adaptability and performance when addressing the problems in other sizes; hence, the DRLH also presents good performance in generalization. The results on sensitivity experiment show good scalability of our proposed algorithms in various distributions. Also, benchmark experiments have verified the ability of the algorithms to be applied to practical problems. When encountered with a new UBQP problem, it can be well addressed by the NN model in any size we trained in a quick and proper manner.

In future studies, since the good performance of our feature engineering, we would like to develop more effective features that can further improve the algorithms. Current features are based on experiments and manual selection, whether we can use machine learning to learn the appropriate features is our future direction. Also, Due to the training of the NN model especially in DRLH requires considerable time, we would develop the efficiency of training and optimization of NN parameters. The last, we would find a good manner to further improve the optimization quality and speed of our algorithms with limited training data.

CRediT authorship contribution statement

Ming Chen: Conceptualization, Methodology, Software, Validation, Writing - original draft, Writing - review & editing, Visualization. **Yuning Chen:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing. **Yonghao Du:** Formal analysis, Writing - original draft. **Luona Wei:** Investigation, Writing - review & editing, Visualization. **Yingwu Chen:** Validation, Supervision, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work is partially supported by the National Natural Science Foundation of China, Grant 71701203 and Natural Science Foundation of Hunan Province, Grant No. 2018JJ3618.

References

[1] Bahram Alidaee, Gary Kochenberger, Karen Lewis, Mark Lewis, Haibo Wang, A new approach for modeling and solving set packing problems, European J. Oper. Res. 186 (2008) 504–512.
[2] Mark Lewis, Gary Kochenberger, Bahram Alidaee, A new modeling and solution approach for the set-partitioning problem, Comput. OR 35 (2008) 807–813.
[3] Mark Lewis, Gary Kochenberger, Bahram Alidaee, A new modeling and solution approach for the set-partitioning problem, Comput. OR 35 (2008) 807–813.
[4] Alain Billionnet, Alain Sutter, Minimization of a quadratic pseudo-Boolean function, European J. Oper. Res. 78 (1994) 106–115.
[5] P. Pardalos, G. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming, Computing 45 (1990) 131–144.
[6] Endre Boros, Peter Hammer, Gabriel Tavares, Local search heuristics for quadratic unconstrained binary optimization (QUBO), J. Heuristics 13 (2007) 99–132.
[7] Peter Merz, Bernd Freisleben, Greedy and local search heuristics for unconstrained binary quadratic programming, J. Heuristics 8 (2000).
[8] Yang Wang, Zhipeng Lü, Fred Glover, Jin-Kao Hao, Backbone guided tabu search for solving the UBQP problem, J. Heuristics 19 (2011) 1–17.
[9] Fred Glover, Gary Kochenberger, Bahram Alidaee, Mehdi Amini, Tabu search with critical event memory: An enhanced application for binary quadratic programs, 1999.

[10] Fred Glover, Tao Ye, Abraham Punnen, Gary Kochenberger, Integrating tabu search and VLSN search to develop enhanced algorithms: a case study using bipartite boolean quadratic programs, European J. Oper. Res. 241 (2013).
[11] Gintaras Palubeckis, Multistart tabu search strategies for the unconstrained binary quadratic optimization problem, Ann. Oper. Res. 131 (2004) 259–282.
[12] Fred Glover, Zhipeng Lü, Jin-Kao Hao, Diversification-driven tabu search for unconstrained binary quadratic problems, 4OR 8 (2010) 239–253.
[13] Ying Zhou, A decomposition-based multi-objective tabu search algorithm for tri-objective unconstrained binary quadratic programming problem, 2017, pp. 101–107.
[14] Talal Alkhamis, Merza Hasan, Mohamed Ahmed, Simulated annealing for the unconstrained quadratic pseudo-Boolean function, European J. Oper. Res. 108 (1998) 641–652.
[15] Kengo Katayama, Hiroyuki Narihisa, Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem, European J. Oper. Res. 134 (2001) 103–119.
[16] Charles Fleurent, Fred Glover, Philippe Michelon, Z. Valli, A scatter search approach for unconstrained continuous optimization, 1996, pp. 643–648.
[17] Wei Ping, Xiong Weiqing, Binary ant colony algorithm with controllable search bias for unconstrained binary quadratic problem, 2012, pp. 3414–3418.
[18] Murilo Zangari, Aurora Pozo, Roberto Santana, Alexander Mendiburu, A decomposition-based binary ACO algorithm for the multiobjective UBQP, Neurocomputing (2017).
[19] Murilo Zangari, Aurora Pozo, Roberto Santana, Alexander Mendiburu, A decomposition-based binary ACO algorithm for the multiobjective UBQP, Neurocomputing (2017).
[20] Andrea Lodi, Kim Allemand, Thomas Liebling, Evolutionary heuristic for quadratic 0-1 programming, European J. Oper. Res. 119 (1999) 662–670.
[21] Istvan Borgulya, A parallel evolutionary algorithm for unconstrained binary quadratic problems, 33, 2008, pp. 603–604.
[22] Zhipeng Lü, Jin-Kao Hao, Fred Glover, A study of memetic search with multi-parent combination for UBQP, 6022, 2010, pp. 154–165.
[23] Peter Merz, Kengo Katayama, Memetic algorithms for the unconstrained binary quadratic programming problem, Bio Syst. 78 (2005) 99–118.
[24] Zhipeng Lü, Fred Glover, Jin-Kao Hao, A hybrid metaheuristic approach to solving the UBQP problem, European J. Oper. Res. 207 (2010) 1254–1262.
[25] Arnaud Liefooghe, Sebastien Verel, Jin-Kao Hao, A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming, Appl. Soft Comput. 16 (2014) 10–19.
[26] W. Michiels, E.H.L. Aarts, Jan Korst, Theory of local search, in: Handbook of Heuristics, 2018, pp. 299–339.
[27] Diederik Kingma, Jimmy Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, 2014.
[28] Yueting Xu, Huiling Chen, Luo Jie, Qian Zhang, Shan Jiao, Xiaoqin Zhang, Enhanced moth-flame optimizer with mutation strategy for global optimization, Inform. Sci. 492 (2019).
[29] Liming Shen, Huiling Chen, Zhe Yu, Wenchang Kang, Bingyu Zhang, Huaizhong Li, Bo Yang, Dayou Liu, Evolving support vector machines using fruit fly optimization for medical data classification, Knowl.-Based Syst. 96 (2016).
[30] Yueting Xu, Huiling Chen, Luo Jie, Qian Zhang, Shan Jiao, Xiaoqin Zhang, Enhanced moth-flame optimizer with mutation strategy for global optimization, Inform. Sci. 492 (2019).
[31] Huiling Chen, Qian Zhang, Luo Jie, Yueting Xu, Xiaoqin Zhang, An enhanced bacterial foraging optimization and its application for training kernel extreme learning machine, Appl. Soft Comput. (2019) 105884.
[32] Xuehua Zhao, Daoliang li, Bo Yang, Chao Ma, Yungang Zhu, Huiling Chen, Feature selection based on improved ant colony optimization for online detection of foreign fiber in cotton, Appl. Soft Comput. 24 (2014) 585–596.
[33] Mingjing Wang, Huiling Chen, Chaotic multi-swarm whale optimizer boosted support vector machine for medical diagnosis, Appl. Soft Comput. 88 (2019) 105946.
[34] Oriol Vinyals, Meire Fortunato, Navdeep Jaitly, Pointer networks, Adv. Neural Inf. Process. Syst. 28 (2015).
[35] Irwan Bello, Hieu Pham, Quoc Le, Mohammad Norouzi, Samy Bengio, Neural combinatorial optimization with reinforcement learning, 2016.
[36] Mohammadreza Nazari, Afshin Oroojlooy jadid, Lawrence Snyder, Martin Takáč, Deep reinforcement learning for solving the vehicle routing problem, 2018.
[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, Illia Polosukhin, Attention is all you need, 2017.
[38] Klaus Greff, Rupesh Srivastava, Jan Koutní k, Bas Steunebrink, Jürgen Schmidhuber, LSTM: A search space odyssey, IEEE Trans. Neural Netw. Learn. Syst. 28 (2015).
[39] Jan Peters, Stefan Schaal, Natural actor-critic, Neurocomputing 71 (2008) 1180–1190.
[40] Yang Wang, Zhipeng Lü, Fred Glover, Jin-Kao Hao, Probabilistic GRASP-Tabu Search algorithms for the UBQP problem, Comput. Oper. Res. 40 (12) 3100–3107.
[41] Xavier Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, J. Mach. Learn. Res. 9 (2010) 249–256.
[42] John Beasley, Obtaining test problems via internet, J. Global Optim. 8 (1996) 429–433.