Neurocomputing 390 (2020) 1–11

Contents lists available at ScienceDirect

Neurocomputing

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

A pointer network based deep learning algorithm for unconstrained binary quadratic programming problem

Shenshen Gu*, Tao Hao, Hanmei Yao

School of Mechatronic Engineering and Automation, Shanghai University, 99 Shangda Road, Shanghai 200444, PR China

ARTICLE INFO

Article history:
Received 15 March 2019
Revised 15 June 2019
Accepted 30 June 2019
Available online 10 March 2020

Keywords:
UBQP
Pointer network
Supervised learning
Deep reinforcement learning

ABSTRACT

Combinatorial optimization problems have been widely used in various fields. And many types of combinatorial optimization problems can be generalized into the model of unconstrained binary quadratic programming (UBQP). Therefore, designing an effective and efficient algorithm for UBQP problems will also contribute to solving other combinatorial optimization problems. Pointer network is an end-to-end sequential decision structure and combines with deep learning technology. With the utilization of the structural characteristics of combinatorial optimization problems and the ability to extract the rule behind the data by deep learning, pointer network has been successfully applied to solve several classical combinatorial optimization problems. In this paper, a pointer network based algorithm is designed to solve UBQP problems. The network model is trained by supervised learning (SL) and deep reinforcement learning (DRL) respectively. Trained pointer network models are evaluated by self-generated benchmark dataset and ORLIB dataset respectively. Experimental results show that pointer network model trained by SL has strong learning ability to specific distributed dataset. Pointer network model trained by DRL can learn more general distribution data characteristics. In other words, it can quickly solve problems with great generalization ability. As a result, the framework proposed in this paper for UBQP has great potential to solve large scale combinatorial optimization problems.

1. Introduction

Deep learning is an emerging field of machine learning research, aiming to study how to automatically extract multilayer characteristics from data. The core idea of deep learning is to extract characteristics from original data in a data-driven way, from low level to high level, from concrete to abstract, from general to specific, through a series of nonlinear transformations. Up to now, deep learning has caused many breakthroughs in the applications of speech recognition [1], image recognition [2], natural language processing, video recommendation and so on. In recent years, with the developing of deep neural network and the combination with operations research, solving combinatorial optimization problem with deep neural network structure [3] has become a very important academic research.

In view of large scale combinatorial optimization problems, scholars have explored how to use deep neural network in these fields and have made some achievements in recent years. Relevant researches mainly focus on the algorithm design based on the pointer network structure for combinatorial optimization problems [4]. Vinyals created pointer network by introducing pointer structure into the Sequence-to-Sequence model with the attention mechanism [5]. Bello improved the pointer network structure and adopted the policy gradient in reinforcement learning to train the pointer network to solve combinatorial optimization problems [6]. Mirhoseini modified the pointer network of Vinyals by removing the coding Recurrent Neural Network (RNN) and embedded the input into the high dimension vector space to solve more complex combinatorial optimization problems [7]. Pointer network has been successfully applied to solve some classical combinatorial optimization problems such as travelling salesman problem (TSP) [8], 0-1 knapsack problem [9] and maximum cut problem [10] etc.

It is well-known that the unconstrained binary quadratic programming (UBQP) problem is equivalent to many important combinatorial optimization problems. The applicability of this representation method has been proved in a variety of designs, such as statistical physics, circuit layout design [11], financial analysis [12], and epilepsy research [13]. Many graphics-related combinatorial optimization problems can be converted from the formula to UBQP problems, such as maximum cut, maximum clique, minimum fixed point, maximum independent set and maximum weighted independent set. Therefore, it is of great significance to study the solving of UBQP problems.

* Corresponding author.
E-mail address: gushenshen@shu.edu.cn (S. Gu).

https://doi.org/10.1016/j.neucom.2019.06.111
0925-2312/

Problems of large scale UBQP cannot be solved quickly by exact solution method because of the NP hard nature [14]. Heuristic algorithms such as simulated annealing and tabu algorithm [15] can solve large scale problems, but they need to be designed separately for specific problems. According to these thoughts and mechanism, different approaches are designed to meet requirements. Existing heuristic algorithms, such as genetic algorithm and simulated annealing algorithm, are designed for some specific combinatorial optimization problems. The performance of designed algorithm will decrease when the condition changes slightly. Moreover, there is no general algorithm framework for different types of problems, and the designed algorithm is not widely applicable.

The characteristic of pointer network model is the design of input and output, which are both sequence without fixed length and correspondence relation. The sequence problem can be modelled directly by pointer network, with the model being trained in the way of end to end structure. UBQP can be seen as a discrete sequence because of its integer variables. Therefore, the pointer network is suitable for solving the integer programming.

Vinyals successfully solved TSP and convex hull problem with pointer network [5]. This indicates that pointer network has great potential in UBQP problems. We first propose supervised learning (SL) to solve UBQP problems. The symmetric matrix of UBQP problems acts as the input of the pointer network. The optimal solution of this problem is the supervision signal, and stochastic gradient descent is used to train the network model. We solve new UBQP problems with trained pointer network model. However, it is usually not easy to generate data samples for supervised learning to train network model.

Reinforcement learning is a kind of label delay learning, which can solve problem of making data samples in the process of supervised learning. Deep reinforcement learning, an algorithm framework proposed by Google DeepMind team, is an algorithm for decision making learning. It shows great potential in solving combinatorial optimization problems. Bello uses deep reinforcement learning (DRL) to solve TSP [6], Hiroki Nakajima uses reinforcement learning to solve knapsack problem [16]. Khalil solves classical combinatorial optimization problems like maximum cut problems and TSP by Q-learning [17]. DRL combines the respective advantages of deep learning and reinforcement learning. Deep learning is good at nonlinear fitting, and reinforcement learning is suitable for decision learning. In this paper, the actor-critic algorithm in deep reinforcement learning is used to solve UBQP problems. The pointer network is used as the actor network to explore the optimal solution, and the full-connected neural network is used to form the critic network to evaluate the solution. The target function acts as the reward function, and the policy gradient is used to update the network parameters.

The paper is organized as follows. Section 2 introduces the unified model of combinatorial optimization problems. Pointer networks and UBQP problems dataset generation is introduced in Section 3. Section 4 introduces the pointer network of supervised learning. Section 5 illustrates UBQP problems solution based on deep reinforcement learning. Next, in Section 6, the numerical experiment is presented and discussed in detail. Finally, Section 7 concludes this paper.

2. Unified model of combinatorial optimization problems

It is usually impossible to solve combinatorial optimization problem in a reasonable time because of its complexity, although the solution is guaranteed theoretically. Instead, heuristic methods are often preferred alternative methods. They provide no guarantee of convergence but more flexibility to utilize special attributes of the search space. Heuristic algorithms aim at specific characteristics of the problem, and use structural characteristics of problem to design search program. However, these tailored approaches in other problems are often limited. Therefore, it is meaningful to select a general modeling framework to represent combinatorial optimization problems and then design a solution algorithm for the general modelling framework. This transformation is helpful to avoid developing new approaches for each new category of problem.

Table 1
The conversion of BQP problems to UBQP problems.

The classic constraints	Equivalent penalty term
$x+y \le 1$	$P(xy)$
$x+y \ge 1$	$P(1-x-y+xy)$
$x+y = 1$	$P(1-x-y+2xy)$
$x \le y$	$P(x-xy)$
$x_1+x_2+x_3 \le 1$	$P(x_1x_2+x_1x_3+x_2x_3)$

It is often assumed that translating a particular problem into a more general representation would be meaningless because of the loss of domain-specific structural knowledge. However, the study of Kochenberger [18] shows that applying general models to specific problems can often overcome the limitations attributed to this model and achieve quite good results. When a problem of particular structure is transformed into this general form, it sometimes becomes easier to be solved. Therefore, this unified model can be seen as a practical method to solve various important combinatorial optimization problems. UBQP problem has the capability of being a general model for combinatorial optimization problems, and it can deal with constraint conditions by introducing penalty functions which are quadratic infeasibility [19]. UBQP model can not only represents many "special case", but also combines with heuristic algorithm and evolutionary algorithm to solve problem. In addition, some constrained models can be converted to UBQP model by adding penalty items. Hammer, Rudeanu [20], Hansen [21] and Hansen et al [22]. show that any quadratic (or linear) objective and linear equation constraint in bounded integer variables can be reexpressed as UBQP problems model.

Many practical combinatorial optimization problems can be represented by following constrained model:

$$\begin{aligned} &\min\ x^TQx \\ &\text{s.t. } Ax = b, x \in \{0,1\}^n \end{aligned} \tag{1}$$

where Q is an $n \times n$ square symmetric matrix of real coefficients, $A \in \mathbb{R}^n$ is a real vector, b is a real constant. The above model contains quadratic term and linear objective function. $x_j^2 = x_j$ can be obtained because x is a 0–1 variable. Model with inequality constraint can also be transformed into the model with equality constraint by introducing slack variable. The UBQP problem model replaces the constraint by adding quadratic unfeasible penalty function to target function. It can be rewritten as follows by introducing a positive penalty coefficient P.

$$\begin{aligned} f(x) &= x^TQx + P(Ax-b)^T(Ax-b) \\ &= x^TQx + x^TDx + c \\ &= x^T\tilde{Q}x + c \end{aligned} \tag{2}$$

where c is a constant obtained by matrix multiplication. It can be discarded to transform constrained binary quadratic programming problem into following unconstrained binary quadratic programming problem model:

$$\min\ x^T\tilde{Q}x, x \in \{0,1\}^n \tag{3}$$

Table 1 shows the translation from constrained binary quadratic problems (BQP) to UBQP problems.

3. Preliminaries

In this section, principle of pointer network and generation of UBQP problems benchmark proposed in [25] are introduced briefly.

3.1. Pointer network

Pointer network was first proposed by Vinyals et al. It combines sequence to sequence (Seq2seq) learning framework with modified attention mechanism. Here, Seq2seq model and attention mechanism are introduced briefly to explain the pointer network model.

Seq2seq was proposed in 2014 and then used in machine translation. In Seq2seq model, encoder compresses complete input sequence into a vector with fixed dimensions, and then decoder generates an output sequence based on this vector. When input sequence is long, the intermediate vector cannot store enough information, which becomes a bottleneck of this model. Attention mechanism is designed to solve this problem. In recent years, attention mechanism has been widely used in various types of deep learning tasks such as natural language processing, image recognition and speech recognition, etc. It plays a significant role in improving effect of sequence learning. And it is one of the core technologies in deep learning technology that deserves the most attention and in-depth understanding. Attention mechanism allows decoder to look up the contents of input sequence at any time, so it no longer needs to store all information in intermediate vector.

In general, for a given input sequence of n vectors $X = (x_1, \ldots, x_n)$ and its associated variable length output sequence $Y = (y_1, \ldots, y_m)$, Seq2seq model calculates conditional probability according to the probabilistic chain rule:

$$p(Y|X) = \prod_{i=1}^{m} p(y_i|y_1, \ldots, y_{i-1}, X) \tag{4}$$

For convenient expression, the hidden state of encoder and decoder is defined as $\{e_1, \ldots, e_n\}$ and $\{d_1, \ldots, d_m\}$, then each conditional probability can be expressed as:

$$p(y_i|y_1, \ldots, y_{i-1}, X) = g(y_{i-1}, d_i, c) \tag{5}$$

$$d_i = h(d_{i-1}, y_{i-1}, c_i) \tag{6}$$

$$c = q\left(e_1, \ldots, e_j\right) \tag{7}$$

$$e_j = f\left(x_j, e_{j-1}\right) \tag{8}$$

where $q\left(e_1, \ldots, e_j\right)$ is the attention mechanism, f, g is transformation function associated with the RNN unit, c is a context vector. In this paper, Long Short Term Memory (LSTM) [26] unit is chosen to constitute RNN network.

Pointer network connects encoder and decoder with a modified attention mechanism. Decoder can query entire encoder state sequence, not just the state of the last LSTM unit [27]. In this way, the decoder can focus on different parts of input sequence of encoder during decoding process, which can significantly improve the effect. In modified attention model, c is no longer equal to the constant obtained in the last LSTM unit of encoder. And calculation of c_i in pointer network model is as follows:

$$c_i = \sum_{j=1}^{n} \alpha_j^i e_j \tag{9}$$

The weight α_j^i is defined as follows:

$$\alpha_j^i = \frac{exp\left(u_j^i\right)}{\sum_{k=1}^{n} exp\left(u_k^i\right)} \tag{10}$$

$$u_j^i = v^T \tanh(W_1 e_j + W_2 d_i) \;\; j \in (1, \cdots, n) \tag{11}$$

where v, W_1, and W_2 are learnable parameters.

To solve the problem that the size of encoder output dictionary depends on the length of the input sequence, the pointer network adjusts the standard attention mechanism to create a pointer u_i^j. The pointer can point to an input sequence element, so the last state of encoder is not applied to transmit additional information to decoder. The softmax function can normalize the vector of length n to become the output probability distribution on the input dictionary. Thus, each factor in Formula (4) can be represented by Formula (12):

$$p^i = p(y_i|y_1, \ldots, y_{i-1}, X) = \text{softmax}\left(u^i\right) \tag{12}$$

In prediction, the position index with maximum probability is selected and the process can be described as follows:

$$y_i = argmax\left(p^i\right) \tag{13}$$

The process of pointer networks is shown in Fig. 1. The sequence is sent to decoder. An element is input at each time step until the end of sequence. The end of the sequence is marked with a special end flag. The model then switches to decoding mode and each time step generates an element in the decoder output sequence until the end of the sequence flag appears. Until then, the whole process is terminated [28].

3.2. UBQP benchmark generator

In the training process of pointer network solver, a mass of samples are essential. It is obviously unrealistic to generate samples by randomly generating 0–1 programming problems with exact solutions because of the NP hard nature. Michael X. Zhou presented a benchmark generator method for {-1,1} quadratic programming problem [23]. This method can generate random BQP problems and solve them in polynomial time. In this paper, the dataset generated by this method is marked as Zhou dataset.

For the original problem, $\min\{f(x) = \frac{1}{2}x^T Q x - c^T x | x \in \{-1, 1\}^n\}$, where $Q = Q^T$, and $c \in \mathbb{R}^n$ is given non-zero vector. The dual problem can be described as follows:

$$\begin{aligned} &\text{Find } Q, c, x, \lambda \\ &\text{s.t. } (Q + diag(\lambda))x = c \\ &\quad Q + diag(\lambda) > 0 \\ &\quad x \in \{-1, 1\}^n \end{aligned}$$

The Matlab code to solve dual problem is as follows. In the matlab scripts, the parameter base is set to control the range of elements in Q.

MATLAB SCRIPTS FOR GENERATING A BENCHMARK UBQP PROBLEMS

```
function [Q, c, x, lambda] = generate_Qc(n)
base = 10;
Q = base*randn(n);
Q = round((Q+Q')/2);
lambda = zeros(n, 1);
x = round(rand(n, 1));
x = 2*x-1;
lambda = sum(abs(Q), 2);
c = (Q + diag(lambda))*x;
```

The $\{-1, 1\}$ quadratic programming problems can be converted to UBQP problems. We convert the Q matrix and optimal solution of the obtained UBQP problem pair into the corresponding input and output formats above, so as to solve the problem of generating a mass of UBQP training samples.

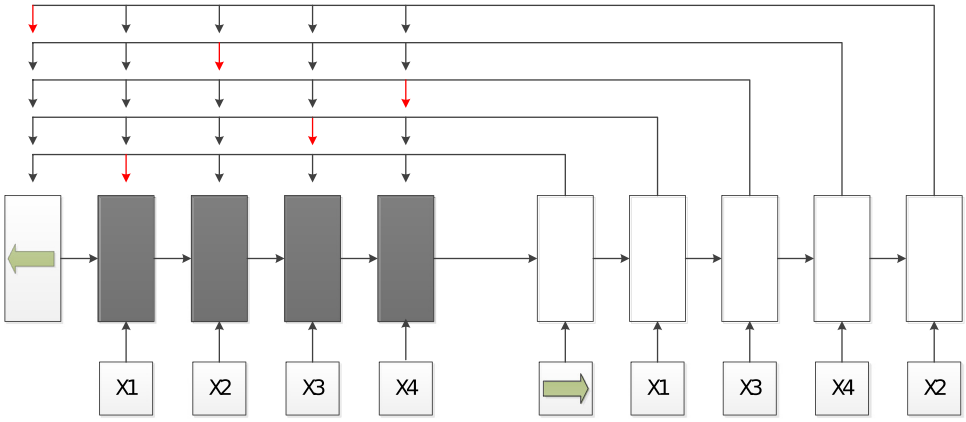

Fig. 1. The architecture of pointer network. (encoder in gray, decoder in white).

4. Supervised learning solution

In this section, supervised learning is used to train pointer network model. The format of input/output and the training method are introduced in detail.

4.1. Input-output mechanism

The characteristic of UBQP problems is that variable value is either 0 or 1, which is a typical selection problem in combinatorial optimization. The problem is equivalent to selecting a set of variables from all variables to take the value of 1, so as to minimize the objective function. Supervised learning algorithm observes the input-output pair, and its goal is to model the mapping relationship between input and output. The target of supervised learning is to determine a function f: $\chi \to \gamma$ from the relationship between the input space χ and the output space γ. We need to design input-output mechanism of pointer network model firstly. In this paper, the UBQP problem is defined as follows:

$$\min_{x\in\{0,1\}^n} x^T Q x$$
$$Q = Q^T = \left(q_{ij}\right)_{n\times n} \tag{14}$$

The main features are stored in Q matrix, then we use deep neural network to extract its data characteristics. The feature of variable $x_i (i = 1, 2, \ldots, n)$ is represented with $q_i = (q_{i1}, q_{i2}, \ldots, q_{in})$, which acts as the input of pointer network encoder. Then encoder network expresses the input through LSTM unit $f_{enc}(., .)$ and expresses it as follows:

$$e_i = f_{enc}(e_{i-1}, q_i)\ i \in (1, \cdots, n) \tag{15}$$

The following is an example to illustrate the output design of the pointer network model.

Example 4.1.

$$\begin{aligned}\min\ f(x) = {} & x_1^2 - 5x_2^2 + 8x_3^2 - 10x_4^2 + 4x_1x_2 - 6x_1x_3 + 8x_1x_4 \\ & + 12x_2x_3 - 14x_2x_4 + 18x_3x_4 \\ & x_i \in \{0, 1\}, i = 1, 2, 3, 4\end{aligned} \tag{16}$$

Hence the symmetric matrix Q for this problem can be described as follows:

$$Q = \begin{pmatrix} 1 & 2 & -3 & 4 \\ 2 & -5 & 6 & -7 \\ -3 & 6 & 8 & 9 \\ 4 & -7 & 9 & -10 \end{pmatrix}$$

The vectors $q_1 = (1, 2, -3, 4)^T$, $q_2 = (2, -5, 6, -7)^T$, $q_3 = (-3, 6, 8, 9)^T$ and $q_4 = (4, -7, 9, -10)^T$ respectively represent the characteristics of the variables x_1, x_2, x_3 and x_4. And (q_1, q_2, q_3, q_4) is input sequence of pointer network.

The optimal solution of above example is $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, and $x_4 = 1$. We input the sequence (q_1, q_2, q_3, q_4) into pointer network, and utilize the optimal solution to guide network to select q_2 and q_4. The input vector selected by the decoder represents the value of the corresponding variable as 1, and the unselected vector represents the value of the corresponding variable as 0. The process is shown in Fig. 2.

In the output part of pointer network model, rules are designed as follows: The pointer network inputs variable characteristics, and the output is variable index location information. One-hot code is used to characterize the output results. For an n dimension UBQP problem, matrix $(n+1)\times(n+1)$ represents the network output. In the above example, the label result can be represented by matrix O_{true}.

$$O_{true} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Rules are as follows:

$$x_j = \begin{cases} 1, & \text{if } o_{ij} = 1, j \neq 0; \\ \text{EOS}, & \text{if } o_{ij} = 1, j = 0; \\ 0, & \text{others.} \end{cases} \tag{17}$$

During the pointer network training, $\text{EOS} = (1, 0, \cdots, 0)^T$ represents the end of solving process. When we finish training model, we utilize the model to solve problem and infer solution, the softmax probability distribution can be obtained. And the corresponding solution may be represented as matrix $O_{predict}$. The variable whose position with maximum probability will be set as 1, and the others are 0. According to the above output matrix, x_2 and x_4 are 1 and the other variables are 0.

$$O_{predict} = \begin{pmatrix} 0.2 & 0.1 & 0.7 & 0 & 0 \\ 0.03 & 0.05 & 0.02 & 0.1 & 0.8 \\ 0.9 & 0.03 & 0.03 & 0.01 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

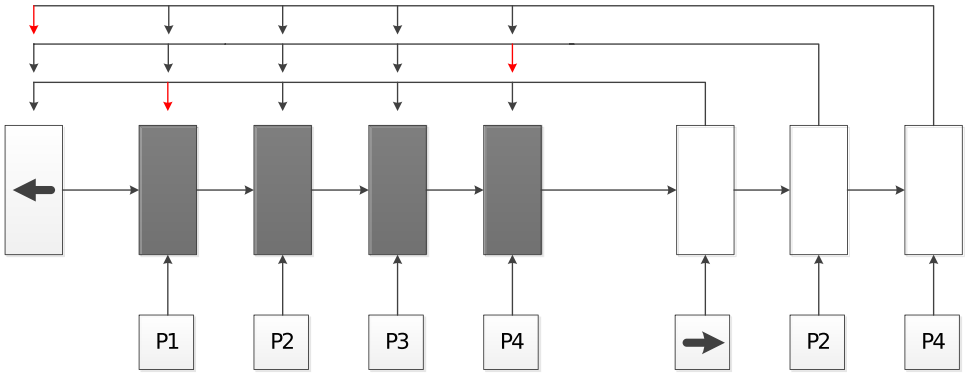

Fig. 2. A pointer network architecture trained by SL. (encoder in gray, decoder in white).

4.2. Algorithm details

Cross entropy is a common loss function in classification problems. It describes the distance between two probability distributions. We choose cross entropy as the loss function of supervised learning pointer network model. The label matrix O_{label} and pointer network output prediction matrix $Q_{predict}$ are converted into one dimension array t and y respectively. The objective of neural network training is to learn parameters to minimize the cross entropy. During the training, we select a batch of data from the training dataset and call it mini-batch. Then we study each mini-batch. Assuming the sample number of each min-batch is N, the loss function is expressed as follows:

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk}\log y_{nk} \tag{18}$$

Stochastic gradient descent (SGD) algorithm is used to train the pointer network. The algorithm can be described in the following way.

Algorithm 1 SGD of pointer network.

Require: α, the learning rate. m, the mini-batch size. T, number of training steps.
Require: θ_0, initial pointer network parameters. $L(\theta)$, the loss function of pointer network.

```
while L(θ) has not converged do
  for t = 1, ..., T do
    sample m UBQP problems samples from the training set randomly
    ∇_θ L(θ) ← ∇_θ [ −(1/m) Σ_m Σ_k t_mk log y_mk ]
    θ ← θ − α · ∇_θ L(θ)
  end for
end while
```

Regarding the selection of batch size, within a reasonable range, increasing batch size can improve the utilization rate of memory and improve the parallelization efficiency of large matrix multiplication. It also reduces the number of iterations to complete one epoch, which accelerates the process of the same amount of data. That is to say, within a certain range, the larger the batch size is and the more accurate the determined descending direction is, the smaller the training shock is caused. However, increasing batch size blindly may cause the collapse of memory capacity and the reduction of the number of iterations needed to complete one epoch (full data set). To achieve the same precision, the time is greatly increased, so the parameter correction is slower.

5. Deep reinforcement learning solution

The difference between DRL and traditional RL is that the deep neural network is used to parameterize the corresponding variables in Markov process, and the nonlinear performance of the neural network and its gradient solution are used to solve the reinforcement learning. Once relevant variables in reinforcement learning are parameterized, the problem of reinforcement learning can be transformed into deep learning for solving.

There is one difference between DRL and traditional machine learning in data. For DRL method, at the beginning of the study we even do not have to prepare the data. Data is accessed through simulation sampling step by step. The data obtained are unlabeled data or unsupervised learning data. RL data is a kind of label delay training data. In order to get a good training result, a lot of iterative sampling is usually needed, which can also effectively prevent overfitting.

5.1. Actor-critic algorithm

Deep reinforcement learning has made remarkable achievements in combinatorial optimization. Inspired by Bello et al., we combine the structural characteristics of UBQP problems with the actor-critic algorithm of DRL to train pointer network model.

Actor-critic method is mainly divided into actor network and critic network. Both parts use the same encoder of the pointer network. After actor network encodes input sequence, the variables of value 1 are selected according to the probability in the decoder part. Critic network encodes the input sequence and uses the neural network to create a value function to predict optimal solutions to UBQP.

For a given input matrix Q, it is expressed as input sequences s. The value of objective function is specified as $L(\pi|s)$. π is the scheme to select which variables to value as 1. Our training objective is to minimize the value of the objective function. Parameters of pointer network are expressed by θ.

The training objectives of actor network can be expressed as follows:

$$J(\theta|s) = E_{\pi\sim p_\theta(\cdot|s)} L(\pi|s) \tag{19}$$

According to the reinforcement learning algorithm proposed by Williams [24], the gradient in Formula (19) is as follows:

$$\nabla_\theta J(\theta|s) = E_{\pi\sim p_\theta(\cdot|s)}[(L(\pi|s - b(s))\nabla_\theta \log p_\theta(\pi|s))] \tag{20}$$

where $b(s)$ represents the baseline function, which does not rely on the evaluation scheme π to estimate the desired value of the target function. The value of $b(s)$ is actually computed with critic network. Variance of the policy gradient can be reduced by introducing baseline value $b(s)$.

B samples are generated from the training set, and a single sample is sampled once according to the value scheme. For example, $\pi_i \sim p_\theta(\cdot|s_i)$, Formula (20) can be approximated by Monte Carlo sampling as follows:

$$\nabla_\theta J(\theta|s) \approx \frac{1}{B}\sum_{i=1}^{B}(L(\pi_i|s_i) - b(s_i))\nabla_\theta \log p_\theta(\pi|s) \tag{21}$$

The mean square deviation of the prediction value and the actual UBQP objective function value obtained by policy sampling is acted as the loss function in critic network, with the stochastic gradient training the network parameters.

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=1}^{B}\left\|b_{\theta_v}(s_i) - L(\pi_i|s_i)\right\|_2^2 \tag{22}$$

For UBQP problems, the actor-critic algorithm is used to update the network parameters.

Algorithm 2 Actor-critic training.

Input: UBQP problems training set S, number of training steps T, batch size B,
Initialize pointer network parameters θ,
Initialize critic network parameters θ_v.
for $t = 1$ to T **do**
  $s_i \sim$ SAMPLE from training set S for $i \in \{1, \ldots, B\}$
  $\pi_i \sim$ SAMPLESOLUTION $(p_\theta(\cdot|s_i))$ for $i \in \{1, \ldots, B\}$
  $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, \ldots, B\}$
  $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}(\mathcal{L}(\pi_i|s_i) - b_i)\nabla_\theta \log p_\theta(\pi_i|s_i)$
  $\mathcal{L}_v \leftarrow \frac{1}{B}\sum_{i=1}^{B}\left\|b_{\theta_v}(s_i) - \mathcal{L}(\pi_i \mid s_i)\right\|_2^2$
  $\theta \leftarrow ADAM(\theta, g_\theta)$
  $\theta_v \leftarrow ADAM\left(\theta_v, \nabla_{\theta_v}\mathcal{L}_v\right)$
end for

5.2. Modified attention mechanism

Encoder and decoder form the pointer network, and these two parts are both composed of LSTM units. The encoder reads input sequence of UBQP problems. At each time step the encoder reads one characteristic sequence and then converts them into a high dimension hidden state $\{enc_i\}_{i=1}^n$ through linear transformation. $enc_i \in R^d$ and d usually takes 128 or 256. The n dimension input sequence q_i of pointer network is converted to d dimension vector enc_i at time step i. The embedding representation of input variable features is realized.

Decoder also contains hidden states $\{dec_i\}_{i=1}^n, dec_i \in R^d$. At each time step i, pointer pointing mechanism generates a probability distribution of which variable is selected with a value of 1. Once the next variable is selected, it serves as the input for the next time step of the decoder.

Compared with supervised learning, reinforcement learning lacks supervisory signal. In order to prevent the pointer network from selecting the selected variables again, we make the following adjustments to the attention mechanism in the deep reinforcement learning model.

$$u_i = \begin{cases} v^T \cdot \tanh\left(W_{ref} \cdot r_i + W_q \cdot q\right), & \text{if } i \neq \pi(j) \text{ for all} j < i; \\ -\infty, & \text{otherwise.} \end{cases} \tag{23}$$

$$A\left(ref, T; W_{ref}, W_q, v\right) = \text{softmax}(C \cdot \tanh(u/T)) \tag{24}$$

Assigning $-\infty$ to a selected variable makes sure that each variable is selected only once.

In decoding process, query vector $q = dec_i \in R^d$ and the set of reference vectors $ref = \{enc_1, \ldots, enc_k\}$ are needed to obtain probability distribution $A(ref, q)$. This distribution represents the probability that the model points to the reference vector r_i when the query vector q appears.

Moreover, in decoding process, we introduce the temperature parameter T, which is set as 1 during the training. The distribution of $A(ref, q)$ will be much smoother when $T > 1$, preventing overfitting of the model. Activation function $\tanh(\cdot)$ and hyperparameter C are used to control the logarithm range of cross entropy.

Critic network includes three neural network modules: LSTM encoder, LSTM processing module, and two-layer fully connected neural network decoder. The encoder of critic network has the same structure as the actor network encoder. LSTM processing module works like to pointer network. At each processing step, hidden state will be updated based on Glimpse function. We take the output of the Glimpse function as input to the next processing step. At the end of the LSTM processing module, the obtained hidden state is converted into the baseline prediction through the two-layer fully connected neural network.

The Glimpse function is parameterized by $W_{ref}^g, W_q^g \in R^{d\times d}$, $v^g \in R^d$ and shown below:

$$p = A(ref, q; W_{ref}^g, W_q^g, v^g) \tag{25}$$

$$G\left(ref, q; W_{ref}^g, W_q^g, v^g\right) = \sum_{i=1}^{k} r_i p_i \tag{26}$$

5.3. Input-output mechanism

According to Example 4.1, we introduce the special variable Split in order to solve this type of problem with the pointer network model. Split is the splitter that takes either 1 or 0 as the variable. We make the following provision: all variables before Split are 1s, all variables after Split are 0s. Therefore, when we solve problem with reinforcement learning, we only need to pay attention to the variables selected before Split, so as to obtain solution. In order to solve this problem with pointer network, we need to make some changes to the above Q matrix. We use zero vector to represent Split, then the corresponding Q matrix can be transformed into a symmetric matrix P.

$$P = \begin{pmatrix} 1 & 2 & -3 & 4 & 0 \\ 2 & -5 & 6 & -7 & 0 \\ -3 & 6 & 8 & 9 & 0 \\ 4 & -7 & 9 & -10 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The solving process is shown in Fig. 3. The output result of pointer network in this mode is the variable index value, indicating that the variable x_2 and x_4 are selected as the value of 1s and other variables are 0s.

6. Experimental results and analysis

In this paper, the pointer networks of SL and DRL are implemented respectively based on TensorFlow deep learning framework [29]. Our models are trained and tested on deep learning platform equipped with NVIDIA GeForce 1080Ti and Intel Core i7-7700 CPU, then the UBQP problem of different dimensions is solved on this model. To verify the validity of this method, we define the accuracy as follows:

$$Accuracy = \frac{v(\text{Ptr-Net})}{v(\text{Opt})} \times 100\% \tag{27}$$

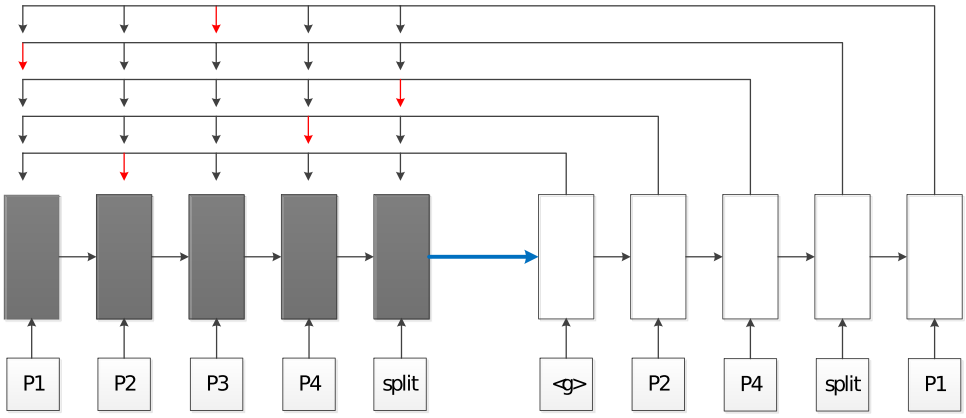

Fig. 3. A pointer network architecture trained by RL. (encoder in gray, decoder in white).

where v(Ptr-Net) is obtained by pointer network and v(Opt) is the optimal value.

We make dataset according to a benchmark generator in Section 3.2 named Zhou dataset. What's more, the most popular dataset currently is J.E. Beasleys OR-library (ORLIB). We perform experiments respectively on Zhou dataset and ORLIB dataset. In the pointer network model for deep reinforcement learning, L2 norm is introduced to conduct gradient clipping at the loss function to prevent overfitting. Batch-normalization (BN) layer is also used in attention mechanism. The BN layer is added directly into the network to make it learn data distribution better. The initial parameters of the pointer network are generated randomly by the uniform distribution set in [-0.08, 0.08]. The initial learning rate is set to 0.001. During training process, batch size is set to 10. Each layer of LSTM contains 256 hidden units. In the pointer networks based on DRL framework, temperature hyperparameter T is set to 3 during inference phase. The initial reward baseline $b(s)$ is set to 100. No parameters will change with the dimension of UBQP problems except the input sequence length.

In this paper, we use validation set to adjust learning rate, batch size and other parameters. Moreover, this paper only presents an idea and preliminary implementation of using pointer network to solve UBQP problem, and there may be some room for optimization in parameter setting to be studied in the future, so as to improve it. In this paper, we select batch size data such as 100, 50 and 20, etc. After the validation set test, the accuracy when the value is 100 and 50 has little difference. After comprehensive consideration of time and accuracy, the current value is selected.

6.1. Experiments on Zhou dataset

For different dimensions problems, 1000 samples are used as the training set, and 100 samples are used as the testing set. The maximum training number of iteration is 100,000. Both the training set and the testing set are generated by the same probability distribution. The density of Q matrix in the UBQP sample is 94.6%.

The parameters of the pointer network trained by SL is optimized by SGD algorithm. Pointer networks are applied to solve UBQP50, UBQP100, UBQP150 and UBQP200 successfully in our experiments. Fig. 4 shows the average accuracy and the proportion of the optimal solution of UBQP problems with different dimensions. For different dimensions of UBQP problems, the time of solving 100 UBQP problems samples depend on trained pointer network model is curved in Fig. 5.

It can be seen from Fig. 5 that the time of solving UBQP problems with the pointer network model increases nearly linearly with the increase of dimension. We generate four testing datasets, UBQP50, UBQP100, UBQP150 and UBQP200, each dataset includes

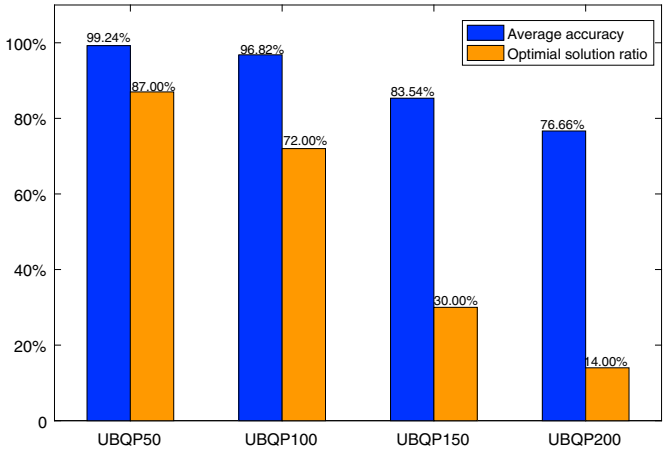

Fig. 4. Average accuracy and proportion of optimal solution for different dimensional UBQP problems by SL.

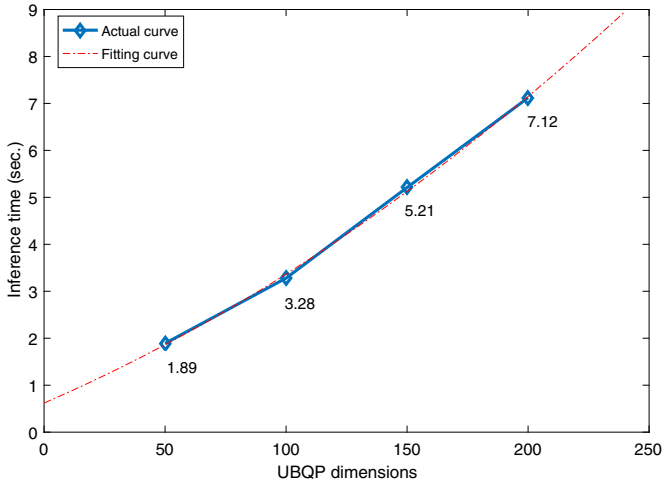

Fig. 5. Time for different dimensional UBQP problems by SL.

10 instances. The optimal solution of UBQP problems with different dimensions, the solution value obtained by the pointer networks trained by SL and the accuracy of the solution are shown in Table 2. It can be seen that the pointer network model trained by supervised learning method can learn the characteristics of UBQP problems of a certain kind of specific data distribution and quickly solve the problems with the same data distribution.

The pointer networks based on DRL are also trained with Zhou dataset. For different dimensions of UBQP problems, actor-

Table 2
Solution and accuracy of different dimensional UBQP by SL.

Sample	Optimal Value	Solution	Accuracy	Sample	Optimal value	Solution	Accuracy
P50.1	−18398	−18398	100.00%	P150.1	−121384	−110764	91.25%
P50.2	−13522	−13522	100.00%	P150.2	−138064	−76638	55.51%
P50.3	−14408	−14408	100.00%	P150.3	−142708	−142708	100.00%
P50.4	−15250	−15250	100.00%	P150.4	−155802	−119888	76.95%
P50.5	−13684	−13154	96.13%	P150.5	−113260	−82272	72.64%
P50.6	−11244	−11244	100.00%	P150.6	−130866	−130866	100.00%
P50.7	−15624	−15624	100.00%	P150.7	−153156	−153156	100.00%
P50.8	−11364	−11364	100.00%	P150.8	−129598	−129598	100.00%
P50.9	−14976	−14976	100.00%	P150.9	−115102	−96620	83.94%
P50.10	−18536	−18536	100.00%	P150.10	−114250	−112674	98.62%
P100.1	−55970	−55970	100.00%	P200.1	−226606	−224342	99.00%
P100.2	−56404	−56404	100.00%	P200.2	−210696	−147300	69.91%
P100.3	−59352	−59352	100.00%	P200.3	−230052	−230052	100.00%
P100.4	−51688	−51688	100.00%	P200.4	−188448	−148216	78.65%
P100.5	−53340	−53340	100.00%	P200.5	−226160	−210496	93.07%
P100.6	−58626	−58626	100.00%	P200.6	−242646	−238034	98.10%
P100.7	−51462	−51462	100.00%	P200.7	−200146	−175044	87.46%
P100.8	−55148	−55148	100.00%	P200.8	−242456	−143414	59.15%
P100.9	−59632	−58590	98.25%	P200.9	−205172	−100194	48.83%
P100.10	−48722	−45370	93.12%	P200.10	−247700	−247700	100.00%

Table 3
Solution and accuracy of different dimensional UBQP by RL.

Sample	Optimal value	Solution	Accuracy	Sample	Optimal value	Solution	Accuracy
R120.1	−70896	−69492	98.02%	R160.6	−151100	−149242	98.77%
R120.2	−76238	−74858	98.19%	R160.7	−145518	−128016	87.97%
R120.3	−81462	−80134	98.37%	R160.8	−130334	−128336	98.47%
R120.4	−74404	−72838	97.90%	R160.9	−145576	−143546	98.61%
R120.5	−68048	−66702	98.02%	R160.10	−141168	−139410	98.75%
R120.6	−96586	−95354	98.72%	R180.1	−179192	−175116	97.73%
R120.7	−72502	−71204	98.21%	R180.2	−187938	−186060	99.00%
R120.8	−94694	−93380	98.61%	R180.3	−188806	−186762	98.92%
R120.9	−99882	−98546	98.66%	R180.4	−182760	−176900	96.79%
R120.10	−77784	−76536	98.40%	R180.5	−177962	−148530	83.46%
R140.1	−98246	−92846	94.50%	R180.6	−170942	−168818	98.76%
R140.2	−123546	−123546	100.00%	R180.7	−182860	−180792	98.87%
R140.3	−109628	−108090	98.60%	R180.8	−165860	−161610	97.44%
R140.4	−107428	−105870	98.55%	R180.9	−176778	−174700	98.82%
R140.5	−120006	−120006	100.00%	R180.10	−180026	−178042	98.90%
R140.6	−127476	−127476	100.00%	R200.1	−217384	−215194	98.99%
R140.7	−120716	−120716	100.00%	R200.2	−196458	−194106	98.80%
R140.8	−113458	−113458	100.00%	R200.3	−232756	−230680	99.11%
R140.9	−123828	−123828	100.00%	R200.4	−231426	−229182	99.03%
R140.10	−110848	−110848	100.00%	R200.5	−217716	−215852	99.14%
R160.1	−126394	−124476	98.48%	R200.6	−264684	−262278	99.09%
R160.2	−145322	−143530	98.77%	R200.7	−227734	−225398	98.97%
R160.3	−154538	−152742	98.84%	R200.8	−241256	−239102	99.11%
R160.4	−142276	−135042	94.92%	R200.9	−218842	−216698	99.02%
R160.5	−133608	−131878	98.71%	R200.10	−212178	−209772	98.87%

critic algorithm is applied to train the corresponding pointer network models. Table 3 shows the experimental results of UBQP120, UBQP140, UBQP160, UBQP180 and UBQP200.

It can be seen that the pointer network model trained by DRL can also successfully learn the method of solving UBQP problems. Fig. 6 curves the accuracy standard deviation of UBQP samples solution from pointer networks trained by SL and DRL method. Compared with pointer network models trained by SL, pointer network models trained by DRL can get smaller accuracy standard deviation of UBQP samples solution. This means that pointer network model trained by DRL is more versatile. Therefore, DRL method is more suitable as a unified method for combinatorial optimization problem.

6.2. Experiments on ORLIB dataset

50 dimension and 100 dimension UBQP samples are selected from the OR-library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html) and converted to the problems that minimize

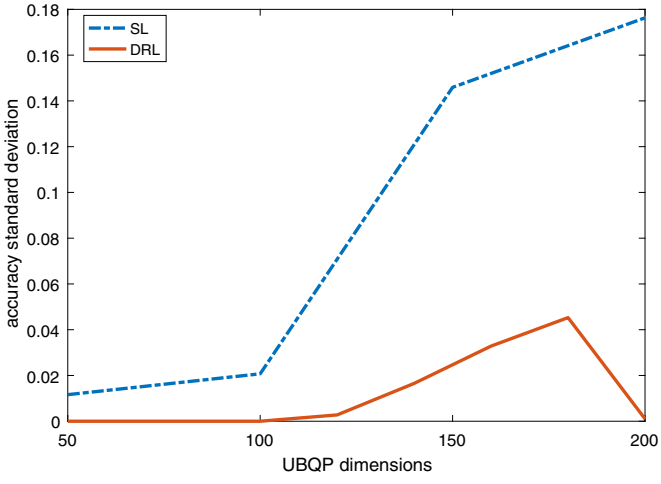

Fig. 6. Accuracy standard deviation of SL and DRL.

Table 4
50-dimensional UBQP solution on ORLIB dataset.

Sample	Optimal value	50a		50b		50c		50d		50e	
		Solution	Accuracy	Solution	Accuracy	Solution	Accuracy	Solution	Accuracy	Solution	Accuracy
1	−2098	−120	5.72%	−1263	60.20%	−1673	79.74%	−1761	83.94%	−1781	84.89%
2	−3702	−618	16.69%	−1028	27.77%	−3460	93.46%	−3524	95.19%	−3552	95.95%
3	−4626	−1098	23.74%	−3400	73.50%	−3910	84.52%	−4290	92.74%	−3994	86.34%
4	−3544	−23	0.65%	−2686	75.79%	−3284	92.66%	−3490	98.48%	−3544	100.00%
5	−4012	−881	21.96%	−2404	59.92%	−3862	96.26%	−3846	95.86%	−3934	98.06%
6	−3693	206	−5.58%	−901	24.40%	−2973	80.50%	−3516	95.21%	−3678	99.59%
7	−4520	−210	4.65%	−3842	85.00%	−4246	93.94%	−4350	96.24%	−4410	97.57%
8	−4216	−602	14.28%	−3576	84.82%	−3544	84.06%	−3610	85.63%	−4160	98.67%
9	−3780	−461	12.20%	−2719	71.93%	−3462	91.59%	−3708	98.10%	−3702	97.94%
10	−3507	−896	25.55%	−2081	59.34%	−2833	80.78%	−3303	94.18%	−3443	98.18%
Average	——	——	12.48%	——	63.40%	——	88.19%	——	93.90%	——	96.02%

the value of UBQP problems equivalently, named UBQP50 and UBQP100.

For UBQP50, we first used Zhou dataset to train pointer network by SL and DRL and obtained model 50a and 50b respectively. Then based on 50b, UBQP50 samples from OR-library acts as the input of the pointer networks based on DRL and then we obtained model 50c, 50d and 50e by increasing 10,000, 50,000 and 100,000 iterations respectively. From the results in the Table 4, the result of model 50a was poor. The result of model 50b was slightly better. It can be found that the model trained by deep reinforcement learning can learn the general rules of UBQP problems and the generalization ability is good. Based on the 50b model, put the UBQP samples and let them learn continuously in the framework of the actor-critic framework, the quality of the solution increases with the number of iterations as shown in Fig. 7.

Similarly, for UBQP100, the model 100a and model 100b are obtained respectively. And then based on 100b, UBQP100 samples from OR-library acts as the input of the pointer networks based on DRL and then we obtain model 100c, 100d, 100e by increasing 10,000, 50,000 and 100,000 iterations, respectively. The results of UBQP100 via different models are shown in Fig. 8. The similar results can be found in Table 5. Compared with the model trained by DRL, the result of model 100a trained by SL is even worse.

6.3. Summary of experimental results

In these experiments, SL and DRL are used to train the pointer network model to solve different dimensions UBQP problems. It is found that the pointer network model of supervised learning training has a strong learning ability for UBQP datasets with low dimension and specific probability distribution, but the effect of high dimension problems is not very good. The UBQP problem is solved by deep reinforcement learning method, and the accuracy standard deviation of the solution is smaller. It shows that deep reinforcement learning model can find the essential characteristics of UBQP problems and obtain the better solution of combinatorial optimization problem from data. Comparatively speaking, the framework of deep reinforcement learning is more suitable for solving large scale combinatorial problems. On the one hand, it does not need to know the exact solution of the training set, on the other

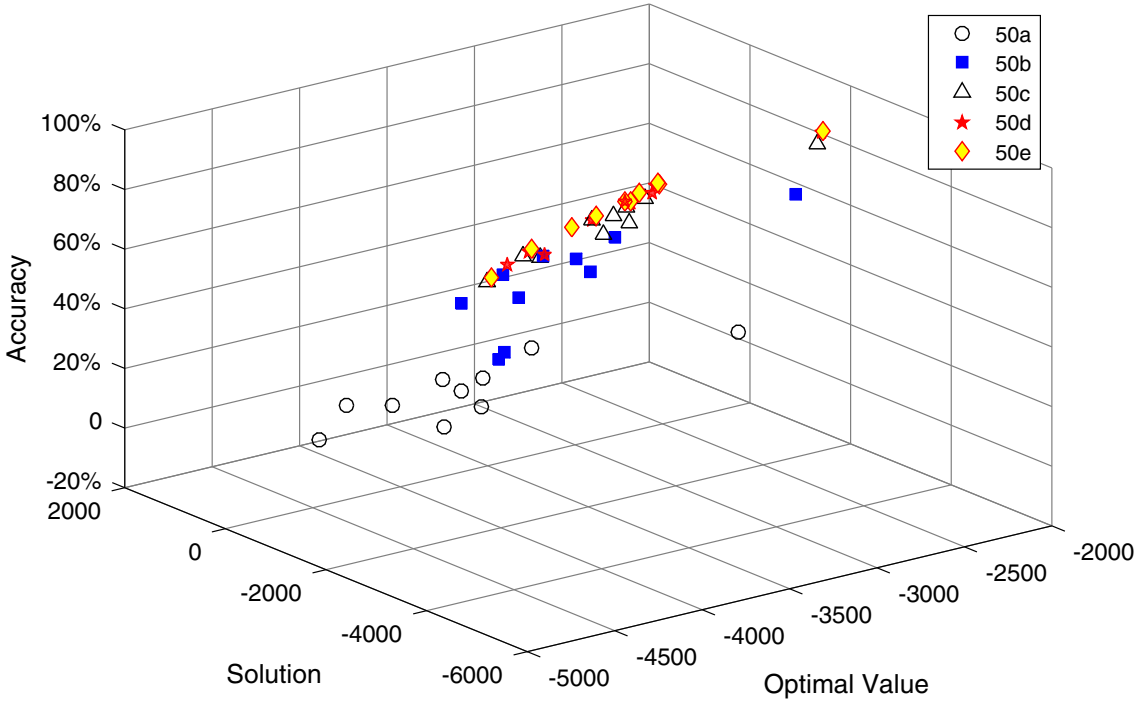

Fig. 7. Results of UBQP50 via different model.

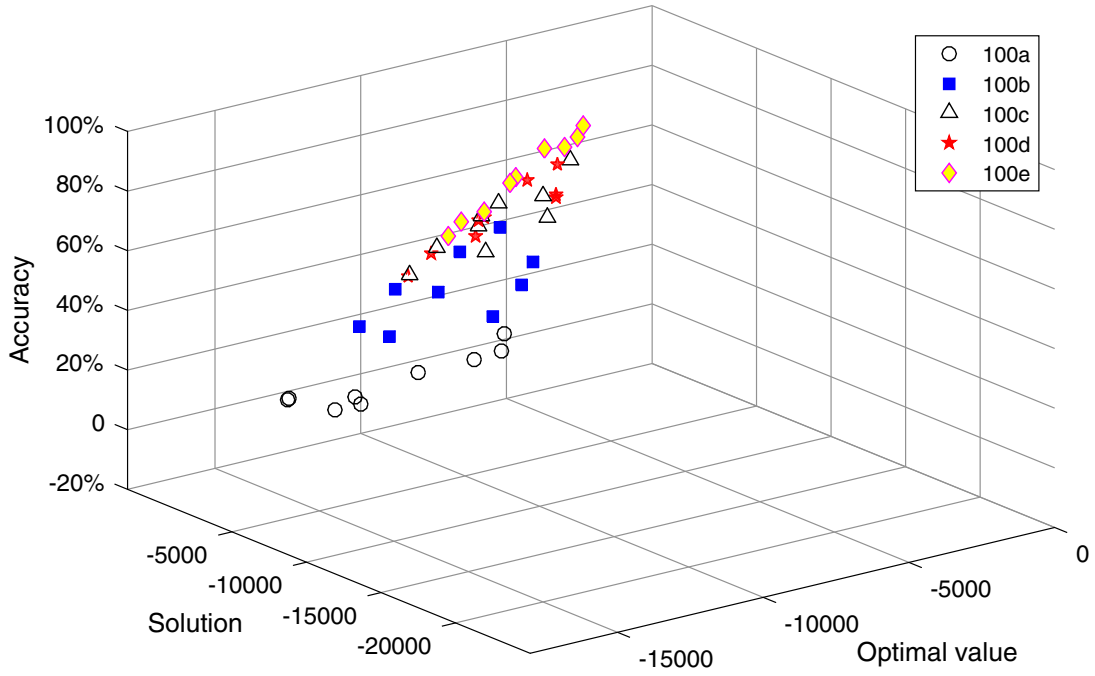

Fig. 8. Results of UBQP100 via different model.

Table 5
100-dimensional UBQP solution on ORLIB dataset.

Sample	Optimal value	100a		100b		100c		100d		100e	
		Solution	Accuracy	Solution	Accuracy	Solution	Accuracy	Solution	Accuracy	Solution	Accuracy
1	−11109	177	−1.59%	−6790	61.12%	−8075	72.69%	−8492	76.44%	−10141	91.29%
2	−7225	−198	2.74%	−1420	19.65%	−4844	67.04%	−5731	79.32%	−6243	86.41%
3	−6333	−244	3.85%	−2382	37.61%	−4866	76.84%	−3969	62.67%	−5718	90.29%
4	−6467	−694	10.73%	−1922	29.72%	−3589	55.50%	−4175	64.56%	−5593	86.49%
5	−9205	−300	3.26%	−5790	62.90%	−4864	52.84%	−7605	82.62%	−8812	95.73%
6	−10705	578	−5.40%	−4547	42.48%	−8647	80.78%	−7124	66.55%	−9773	91.29%
7	−11589	639	−5.51%	−3036	26.20%	−9180	79.21%	−9021	77.84%	−9394	81.06%
8	−14017	−945	6.74%	−5788	41.29%	−9116	65.04%	−9066	64.68%	−11696	83.44%
9	−13999	−975	6.96%	−8142	58.16%	−10957	78.27%	−10575	75.54%	−12566	89.76%
10	−18930	−846	4.47%	−15555	82.17%	−13986	73.88%	−14958	79.02%	−17744	93.73%
Average	––	––	2.56%	––	50.53%	––	71.29%	––	73.66%	––	89.14%

hand, the reward mechanism of deep reinforcement learning is more in line with the combinatorial problems.

In our experiments, Zhou dataset is generated by the same probability distribution and ORLIB dataset is generated by the different probability distribution. It is easy for deep neural network to learn the characteristic and rule of specific probability distribution data. Therefore, the effect in Zhou dataset is better than that in ORLIB dataset.

Although the process of training pointer network takes a lot of time, the speed of solving UBQP problems using the trained pointer network model is quite fast and its solution time increases almost linearly with dimension increase. Moreover, it can solve many UBQP problems in batches at the same time.

7. Conclusions

This paper proposes a new method for large scale combinatorial optimization problems, converting different types of combinatorial optimization problems to UBQP. And then pointer networks based on deep learning are applied to solve UBQP problems. We describe the architecture of pointer network to solve UBQP problems based on SL and DRL respectively. The generation of the UBQP problem dataset and the training method of the model are introduced in detail. According to the experimental results, the effectiveness of the pointer network for UBQP problems is verified. The conclusions can be obtained without considering dimension reduction. For the low dimension UBQP, the solution accuracy of point network trained by SL is better than DRL. For the high dimension UBQP, the accuracy standard deviation of the solution obtained by pointer network trained by DRL is smaller. In short, pointer networks based on deep learning have great potential to solve UBQP problems and other combinatorial optimization problems.

Acknowledgement

The work described in the paper was supported by the National Science Foundation of China under Grants 61876105 and 61503233.

References

[1] C.C. Chiu, T.N. Sainath, Y. Wu, et al., State-of-the-art speech recognition with sequence-to-sequence models, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 4774–4778.
[2] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (2) (2012) 1097–C1105.
[3] F. Yang, T. Jin, T.Y. Liu, et al., Boosting dynamic programming with neural networks for solving NP-hard problems, in: Proceedings of the Asian Conference on Machine Learning, 2018, pp. 726–739.

[4] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, Adv. Neural Inf. Process. Syst (2014) 3104–3112.
[5] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, Adv. Neural Inf. Process. Syst. (2015) 2692–2700.
[6] I. Bello, H. Pham, Q.V. Le, et al., Neural combinatorial optimization with reinforcement learning[j]. arXiv preprint arXiv:1611.09940 (2016).
[7] A. Mirhoseini, H. Pham, Q.V. Le, et al., Device placement optimization with reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2017, pp. 2430–2439.
[8] S. Gu, T. Hao, S. Yang, The Implementation of a Pointer Network Model for Traveling Salesman Problem on a Xilinx Pynq Board, in: Proceedings of the International Symposium on Neural Networks, Springer, Cham., 2018. pp. 130–138
[9] S. Gu, T. Hao, A Pointer Network Based Deep Learning Algorithm for 0-1 Knapsack Problem. Advanced Computational Intelligence (ICACI), in: 2018 Tenth International Conference on. pp. 473–477, 2018.
[10] S. Gu, Y. Yang, A Pointer Network Based Deep Learning Algorithm for the Max–cut Problem, in: International Conference on Neural Information Processing, Springer, Cham., 2018. pp. 238–248
[11] M. Grotschel, M. Junger, G. Reinelt, An application of combinatorial optimization to statistical physics and circuit layout design, Oper. Res. 36 (3) (1988) 493–513.
[12] R.D. McBride, J.S. Yormack, An implicit enumeration algorithm for quadratic integer programming, Manag. Sci. 26 (1980) 282–C–296.
[13] L.D. Iasemidis, D.S. Shiau, J.C. Sackellares, P. Pardalos, Transition to epileptic seizures: optimization, DIMACS Series Discret. Math Theor. Comput. Sci. 55 (2010) 55–73.
[14] S. Gu, R. Cui, An efficient algorithm for the subset sum problem based on finite-time convergent recurrent neural network, Neurocomputing 149 (2015) 13–21.
[15] Y. Zhou, J. Wang, Z. Wu, et al., A multi-objective tabu search algorithm based on decomposition for multi-objective unconstrained binary quadratic programming problem, Knowl. Based Syst. 141 (2018) 18–30.
[16] H. Nakajima, H. Iima, The Solution of Combinatorial Optimization Problems Based on Reinforcement Learning, in: Proceedings of the International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, ACM, 2017, pp. 78–82.
[17] E. Khalil, H. Dai, Y. Zhang, et al., Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. (2017) 6348–6358.
[18] G.A. Kochenberger, F. Glover, B. Alidaee, et al., A unified modeling and solution framework for combinatorial optimization problems, OR Spectrum. 26 (2) (2004) 237–250.
[19] G. Kochenberger, J.K. Hao, F. Glover, et al., The unconstrained binary quadratic programming problem: a survey, J. Comb. Optim. 28 (1) (2014) 58–81.
[20] P. Hammer, S. Rudeanu, Boolean Methods in Operations Research, Springer, Berlin Heidelberg New York, 1968.
[21] P.B. Hansen, Methods of nonlinear 0–1 programming, Ann. Discret. Math. 5 (1979) 53–70.
[22] P. Hansen, B. Jaumard, V. Mathon, Constrained nonlinear 0–1 programming, INFORMS J. Comput. 5 (2) (1993) 97–C–119.
[23] J.E. Beasley, Heuristic algorithms for the unconstrained binary quadratic programming problem, 1998. London, England.
[24] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 3–4.
[25] M.X. Zhou, A benchmark generator for boolean quadratic programming, 2014. arXiv:1406.4812.
[26] H. Sak, A. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, in: Proceedings of the Fifteenth Annual Conference of the International Speech Communication Association, 2014.
[27] A. Milan, S.H. Rezatofighi, R. Garg, et al., Data-driven approximations to NP-hard problems, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[28] A. Mottini, R. Acuna-Agost, Deep Choice Model Using Pointer Networks for Airline Itinerary Prediction, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017. pp. 1575–1583
[29] G. Zaccone, M. R. Karim, A. Menshawy, Deep Learning with Tensorflow, Packt Publishing Ltd, 2017.

Shenshen Gu was born in Shanghai, China. He received his B.E degree in Computer Science from Shanghai University of Engineering Science in 2002, his M.E. degree in Computer Science from Shanghai University in 2005, and his Ph.D. degree in Automation and Computer-Aided Engineering from the Chinese University of Hong Kong in 2009. He then joined the School of Mechatronics Engineering and Automation at Shanghai University, where he is currently an associate professor. Dr. Shenshen Gu's research interests include optimization, optimal control and neural networks.

Tao Hao was born in Yangzhou, Jiangsu, China. He received his B.E degree in Electrical Engineering and Automation from Shanghai University in 2016. He is currently an M.Phil. student at the School of Mechatronic Engineering and Automation, Shanghai University under the supervision of Dr. Shenshen Gu. His research interests include optimization, deep learning and neural networks.

Hanmei Yao was born in Anqing, Anhui, China. She received her B.E degree in Electrical Engineering and Automation from Wenzhou University in 2018. She is currently an M.Phil. student at the School of Mechatronic Engineering and Automation, Shanghai University under the supervision of Dr. Shenshen Gu. Her research interests include optimization, machine learning and neural networks.