RBNets: A Reinforcement Learning Approach for Learning Bayesian Network Structure

Zuowu Zheng, Chao Wang, Xiaofeng Gao(✉), and Guihai Chen

MoE Key Lab of Artificial Intelligence, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{waydrow,wangchao.2014}@sjtu.edu.cn, {gao-xf,gchen}@cs.sjtu.edu.cn

Abstract. Bayesian networks are graphical models that are capable of encoding complex statistical and causal dependencies, thereby facilitating powerful probabilistic inferences. To apply these models to real-world problems, it is first necessary to determine the Bayesian network structure, which represents the dependencies. Classic methods for this problem typically employ score-based search techniques, which are often heuristic in nature and have limited running times and performances that do not scale well for larger problems. In this paper, we propose a novel technique called RBNets, which uses deep reinforcement learning along with an exploration strategy guided by Upper Confidence Bound for learning Bayesian Network structures. RBNets solves the highest-value path problem and progressively finds better solutions. We demonstrate the efficiency and effectiveness of our approach against several state-of-the-art methods in extensive experiments using both real-world and synthetic datasets.

Keywords: Bayesian network · Structure learning · Reinforcement learning

1 Introduction

A Bayesian network is a probabilistic graphical model that represents probabilistic dependencies between random variables in a domain compactly and intuitively. It has a wide range of applications in data mining, classification problems, medical diagnosis and engineering decisions, etc. Learning the structure of Bayesian networks involves finding the acyclic graph that fits a discrete dataset best over the random variables. It is the basis for solving practical problems, but is a very challenging task in machine learning.

This work was supported by the National Key R&D Program of China [2020YFB1707900], the National Natural Science Foundation of China [62272302, 62202055, 62172276], Shanghai Municipal Science and Technology Major Project [2021SHZDZX0102], and CCF-Ant Research Fund [CCF-AFSG RF20220218].

D. Koutra et al. (Eds.): ECML PKDD 2023, LNAI 14171, pp. 193–208, 2023.
https://doi.org/10.1007/978-3-031-43418-1_12

In this work, we consider the problem of learning an appropriate Bayesian network structure for a given dataset and scoring function. Such score-based learning has been shown to be NP-hard [6], so much early research focused on local search strategies, searching for a structure that optimizes a particular scoring function, such as greedy hill climbing approaches [9,10,12], ordering-based search [14,19,25], and ant colony optimization [4]. They all take as input scores of candidate parent sets of all variables, and use various optimization techniques to find a structure that is a good predictor of the data. Unfortunately, these algorithms are unable to guarantee the quality of the learned networks. Thus, it motivates the research of more principled search algorithms.

Over the past several decades, exact algorithms have been studied extensively and there have been proposals based on A* search [5,28], dynamic programming [21,23], branch and bound [3,16], model averaging [15], and integer linear programming [7,8,13]. Besides, some reinforcement learning based methods are proposed, such as RL-BIC [29] and CORL [26]. These methods achieve good performance on smaller networks but fail in domains with a large number of variables unless the cardinality of parent sets is severely restricted.

In this paper, we view the problem of learning a Bayesian network structure via the optimization of a scoring function as a path-finding problem in an order graph [28]. A straightforward approach to solve this path-finding problem is to resort to A* [28]. However, because the number of nodes in this order graph is exponential in the number of variables, even A* struggles to solve this problem as the number of variables increases. The goal of this paper is to propose a novel method based on reinforcement learning (RL) to improve the learning performance.

RL is a machine learning method [24] concerned with how agents ought to take actions in an environment so as to maximize some notion of expected cumulative reward. Recently, it has been shown [17,22] that it can scale to decision-making problems that were previously intractable, such as high-dimensional state and action spaces. It is therefore natural to use RL to tackle this path-finding problem. To demonstrate this idea, we apply Deep Q-network (DQN) to find a highest-value path in the order graph, which achieves good performance in many fields. The proposed approach RBNets amounts to doing a stochastic search guided by the costs of the edges of the graph while A* searches for a shortest path guided by a heuristic function. Upper Confidence Bound (UCB) based strategy is utilized for better exploration rather than simple ϵ-greedy strategy.

The contributions of this paper are as follows: (1) we propose a novel method RBNets based on deep reinforcement learning for Bayesian network structure learning; (2) we integrate it with a UCB-based exploration strategy to tackle the dilemma of exploration and exploitation; (3) we thoroughly validate our propositions on diverse sets of experiments using several real-world and synthetic datasets, which shows the efficiency and effectiveness of our proposition.

2 Preliminaries

In this section, we first review the problem of Bayesian Network Structure Learning (BNSL). Then we introduce the local score and pruning rules for calculating the candidate parent sets. We formulate the BNSL as a shortest-path problem [28] using order graph. It is the basis of our proposed method.

2.1 Bayesian Network Structure Learning

A Bayesian network is a directed acyclic graph (DAG) defined as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{X_1, X_2, ..., X_n\}$ is a set of random variables and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a collection of arcs. A directed arc from X_i to X_j denotes a probabilistic dependence between the two variables, which also means X_i is a parent of X_j. The parent set of X_j is denoted by Π_j. Numerically, a conditional probability distribution $P(X_j \mid \Pi_j)$ describes the dependence between X_j and the variables in Π_j. The joint probability over all variables factorizes as the product of all the conditional probability distributions in the Bayesian network, $P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \Pi_i)$.

Given a dataset $\mathcal{D} = \{D_1, D_2, ..., D_m\}$, where D_i is a set of values over variables in $\mathcal{V}$. The goal of structure learning is to find a DAG G that optimizes a given scoring function, which measures the goodness of fit of a network structure to $\mathcal{D}$. In this work, as customary, we assume that each variable is discrete with a finite number of possible values, and no data point has missing values in $\mathcal{D}$. Thus we can define the BNSL as follows.

Definition 1 (BNSL). *The optimal Bayesian network structure*

$$G^* = \arg\max_{G \in \mathcal{G}} \hat{s}(G, \mathcal{D}), \tag{1}$$

where $\mathcal{G}$ is the set of all possible DAGs and $\hat{s}$ is the scoring function.

To make the problem tractable, the standard approach is to use a scoring function that is decomposable over the Bayesian network's structure, i.e., the score of a network can be decomposed into a sum of node scores $\hat{s}(G) = \sum_i^n \hat{s}_i(\Pi_i)$ [12]. The values of $\hat{s}_i(\Pi_i)$ are often called *local* scores.

2.2 Local Scores

Many decomposable scoring functions can be used to measure the quality of a network structure, such as the K2, BDeu, BDe, MDL or BIC scores [15]. For concreteness, we present our work with the Bayesian Information Criterion (BIC), i.e., $\hat{s}_i = BIC$. However, our method could be extended to other decomposable scoring functions. BIC is defined as follows.

$$BIC(G) = \sum_{i=1}^{n} BIC(X_i, \Pi_i) \qquad \text{where}$$

$$BIC(X, \Pi) = \sum_{\pi \in \Pi} \sum_{x \in X} \left(m_{x,\pi} \log \hat{\theta}_{x|\pi} \right) - \frac{\log m}{2} (|X| - 1) \, |\Pi| \,,$$

where $\pi \in \Pi$ (resp. $x \in X$) denotes an assignment of all variables in Π (resp. of variable X), $\hat{\theta}_{x|\pi}$ is the maximum likelihood estimate of the conditional probability $P(X = x \mid \Pi = \pi)$, $m_{x,\pi}$ denotes the number of data points consistent with $(X = x \ \wedge \ \Pi = \pi)$, and $|\Pi|$ (resp. $|X|$) represents the number of possible instantiations of variables in Π (resp. of variable X) with the convention that $|\varnothing| = 1$.

Given n variables, there are 2^{n-1} possible parent sets for each variable. Thus, the size of the solution space grows exponentially in the number of variables. It is therefore impractical to calculate local scores for all parent sets. The computation of this process can be sped up by adopting exact pruning approaches, which guarantees not to remove the optimal network from consideration. There are also other pruning strategies, e.g., restricting the cardinality of parent sets. However, they could eliminate parent sets that are in a globally optimal network. We utilize the following theorems that hold in particular for the BIC scoring function. The first theorem [3] is useful and can handle the issue of having to compute scores for all possible parent sets.

Theorem 1. *The optimal graph G has at most $O(\log m)$ parents per node.*

Therefore, there is no need to compute scores for any parent set with a size larger than $O(\log m)$, because these parent sets are guaranteed to be suboptimal.

This second theorem [3] provides a bound to discard parent sets without even inspecting them.

Theorem 2. *Let X_i be a variable with $\Pi_i \subset \Pi_i'$ two possible parent sets such that $t_i(\Pi_i') + \hat{s}_i(\Pi_i) > 0$, where $t_i(\Pi_i') = |\Pi_i'|\,(|X_i| - 1)$. Then Π_i' and all supersets $\Pi_i'' \supset \Pi_i'$ are not optimal parent set of X_i.*

The entropy of a parent set is also a useful measure for pruning. [2] gave a pruning rule that provides an upper bound on conditional entropy of candidate parent sets and their subsets. The entropy for a variable X_i and parent set Π_i are defined as follows, respectively.

$$H(X_i) = -\sum_{k=1}^{|X_i|} \frac{m_{ik}}{m} \log \frac{m_{ik}}{m} \tag{2}$$

$$H(\Pi_i) = -\sum_{j=1}^{|\Pi_i|} \frac{m_{ij}}{m} \log \frac{m_{ij}}{m}, \tag{3}$$

where m_{ik} and m_{ij} represent, respectively, the number of times $(X_i = x_{ik})$ and $(\Pi_i = \pi_{ij})$ appear in the dataset. The conditional information is defined as usual,

$$H(X \mid Y) = H(X \cup Y) - H(Y). \tag{4}$$

Theorem 3. *Let X_i be a variable, and Π_i be a parent set for X_i. Let $X_j \notin \Pi_i$ such that $m \cdot \min\{H(X_i \mid \Pi_i), H(X_j \mid \Pi_i)\} \geq (1 - |X_j|) \cdot t_i(\Pi_i)$. Then the parent set $\Pi_i' = \Pi_i \cup \{X_j\}$ and all its supersets can be safely ignored when building the list of parent sets for X_i.*

It can be used for pruning the search space of parent sets without having to compute their BIC scores.

After pruning, the remaining parent sets are defined as *potentially optimal parent sets* (POPS). We denote the set of POPS as $\mathcal{P}_i$ for variable X_i. Given POPS as the input, the BNSL problem can be converted into the following form.

$$G^* = \arg\max_{G \in \mathcal{G}} \sum_{i=1}^{n} \hat{s}_i(X_i, \Pi_i), \tag{5}$$

where Π_i is the parent set of X_i in G and $\Pi_i \in \mathcal{P}_i$. In practice, POPS are of course not computed, but the two previous theorems are used to stop the search for known suboptimal subsets.

2.3 Order Graph

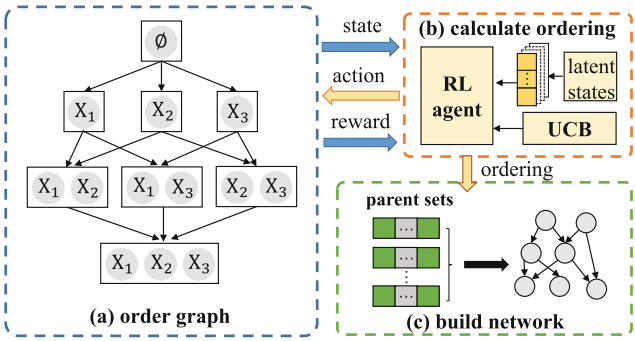

Fig. 1. RBNets framework. We have three components in our method. (a) Order graph is the environment of the agent, from which it can extract states and return rewards. This figure is an example order graph with three variables. (b) We use random walk to learn latent representations of states in the order graph. The agent explores order graph based on UCB search strategy and takes action in the environment. Finally it finds the ordering of variables. (c) From the ordering and POPS, we can calculate the parent sets and rebuild the Bayesian network structure.

Learning the structure of a Bayesian network can be seen as a search in a state-space graph (see Fig. 1(a) for an example with three variables). For a problem with n variables, this graph contains 2^n nodes. Each node represents a subset of variables. For ease of presentation, we identify nodes and subsets. They can be organized into $n+1$ layers. The top node corresponding to the empty set is the start node at layer 0, while the bottom node, which includes all variables, is the goal node at layer n. For any subset U and any variable $X_i \notin U$, an arc connects U to $U \cup \{X_i\}$. In our context, an arc corresponds to adding a new variable X_i

to a subnetwork whose variables are in U. The value of an arc is defined as

$$cost(U \rightarrow U \cup \{X_i\}) = BestCost(U, X_i) \\ = \max_{\Pi_i \subseteq U, \Pi_i \in \mathcal{P}_i} \hat{s}_i(X_i, \Pi_i), \quad (6)$$

where $BestCost(U, X_i)$ is the score of an optimal parent set for X_i in predecessor set U with the POPS constraint. For example, the edge from $\{X_1, X_3\}$ to $\{X_1, X_2, X_3\}$ has a cost equal to $BestCost(\{X_1, X_3\}, X_2)$, which is the score of the parent set of X_2 optimal in $\{X_1, X_3\}$.

With this definition of search graph, each path from the start node to the goal node represents an order of the variables; that is, each node can only find their parents in its predecessor. For example, the path $\varnothing, \{X_2\}, \{X_1, X_2\}, \{X_1, X_2, X_3\}$ denotes the variable ordering X_2, X_1, X_3, so this graph is called *order graph*. The value of a path is equal to the sum of the value of all the edges over the path. The longest path is then the path with the maximum total value in the order graph.

From a longest path from the start node to the goal node, we can reconstruct a Bayesian network structure by noting that each edge on the path encodes the choice of good parents for one of the variables out of the preceding variables. Therefore, we can generate a valid Bayesian network by putting together all the good parent choices.

3 Deep Reinforcement Learning-Based Bayesian Network Structure Learning

In this section, we present our proposed approach for solving BNSL problem. First, we formulate the BNSL problem by Reinforcement learning. Then we introduce Upper-Confidence Bound based exploration strategy for the agent. Finally, we present Deep Q-learning algorithm for our method. The framework is depicted in Fig. 1.

3.1 Reinforcement Learning Formulation

We propose to solve the previously-described highest-value path problem with a reinforcement learning (RL) approach. In RL, an agent interacts with its environment in order to learn a *policy* (i.e., which determines how to select actions in each state) in order to maximize an expected sum of rewards. As shown in Fig. 1(a), we use the order graph as the environment of the RL agent. Formally, the problem is defined as an episodic RL problem with a deterministic transition function, i.e.,

(a) state. Each node represents a state s of the agent. The initial state corresponds to the start node and the final state is the goal node.

(b) action. In time step t, the agent arrives at a state s_t from s_{t-1} and then selects an action a_t from a discrete action set A_t according to a policy π. In our formulation, A_t is the set of the neighbor states of s_t. Since each state has a varying number of actions, $|A_t|$ is set to the maximum number of actions of states. The illegal actions (i.e., there exists no corresponding arc in the order graph) are not considered.

(c) transition function. Following an action a_t in state s_t, a transition to s_{t+1} occurs with probability one if there exists an arc between the corresponding nodes in the order graph.

(d) reward. We use the value of each arc as the reward signal r_t at each time step t, that is, if the agent arrives at s_t from U to $U \cup \{X_i\}$, the reward is

$$r_t = \max_{\Pi_i \subseteq U, \Pi_i \in \mathcal{P}_i} \hat{s}_i(\Pi_i) \tag{7}$$

The RL agent interacts with the environment as follows: The agent starts in the initial state (i.e., start state). The agent repeatedly chooses an action in its current state, observes a reward and moves to a new state (i.e., adjacent node in the order graph). When the agent reaches the goal node, it returns automatically to the initial state.

Q-learning [24] is a standard algorithm for solving an RL problem. While interacting with the environment, it consists in learning the value $Q(s, a)$ of actions a in states s. It is updated as follows after an action a is performed in state s, observing reward r and moving to s':

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)],$$

where α is the learning rate ($0 < \alpha \leq 1$) determining to what extent newly acquired information overrides old information and $\gamma \in [0, 1]$ is a discount factor determining the importance of future rewards. In our episodic problem, γ is set to one so that sum of rewards corresponds to the value of a path.

When Q-learning is used to solve a path problem, $Q(s, a)$ has a simple interpretation: it is the estimated score of the best path found so far from s to the final state starting from the arc corresponding to a. When learning stops, the best found path can be recovered by choosing the actions that maximizes $Q(s, a)$ starting from the initial state. Using Q-learning makes the algorithm progressively find better and better solution. However, its efficiency depends on the exploration strategy used in the algorithm, which decides which action to try during learning.

3.2 Upper Confidence Bounds Based Strategy

The exploration strategy makes a trade-off between exploration (i.e., try actions that are currently considered sub-optimal but that may reveal to be good later on) and exploitation (i.e., select the best action found so far, which may in fact not be optimal).

Classically, Q-learning is run with an ϵ-greedy action selection: in current state s, select $\arg\max_a Q(s,a)$ with probability $1-\epsilon$ or choose a uniform random action otherwise. Running Q-learning amounts then to perform a local random search in each encountered state in order to find the best subpath to the final state. Although such strategy guarantees the convergence to an optimal solution (under some technical conditions) [24], it is clearly inefficient as actions will continue to be chosen with some non-negligible probability even if they revealed to be very bad. A better approach is to use an exploration based on an Upper-Confidence Bound (UCB), which is optimal (in terms of regret) in multi-armed bandits [1].

In every state and for each action, we define the following bonus $b(s,a)$ as

$$b(s,a) = \sqrt{\frac{2\ln \tau(s)}{\tau(s,a)}},$$

where $\tau(s)$ is the number of times s has been visited so far and $\tau(s,a)$ is the number of times action a has been tried in state s. Note that the bonus is higher for less-tried actions. A UCB-based exploration strategy chooses actions in a state s with $\arg\max_a [Q(s,a) + b(s,a)]$. This strategy automatically finds a good balance between exploration (i.e., high bonus) and exploitation (i.e., high Q value).

3.3 Deep Q-Learning Algorithm

When the number of Bayesian network variables becomes large, the number of states grows exponentially, which prevents the direct application of Q-learning. For large or continuous state-space, a function approximation scheme is needed to approximate the Q function. In this paper, we use the deep Q-network (DQN) algorithm [17] for its proven efficiency and performance. DQN learns $Q(s,a;\theta)$, an approximation of the Q values, with θ representing the parameter of the neural network, by minimizing the following loss function:

$$L(\theta) = E_{s,a,r,s'}[(\hat{y} - Q(s,a;\theta))^2], \tag{8}$$

where $\hat{y} = r + \gamma \max_{a'} Q(s',a';\theta^-)$, and θ^- represents the independent target network's parameters that is a copy of θ, which is updated at a lower frequency.

In addition, experience replay is adopted to improve the stability of the DQN training. It consists in (1) storing in a replay buffer the transitions (s,a,r,s') experienced by the agent during its interactions with the environment, then (2) minimizing the previous loss function on mini-batches uniformly sampled from the buffer.

Besides, both DQN and Q-learning are known to overestimate the Q values, as the max operator uses the same values to both select and evaluate an action. Correspondingly, the Deep Double Q-network (DDQN) [11] is proposed to solve this problem by redefining the target $\hat{y}$ with:

$$\hat{y} = r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta);\theta^-), \tag{9}$$

Algorithm 1: DDQN Algorithm with UCB

Input: empty replay buffer B, initial network parameters θ, θ^--copy of θ, replay buffer maximum size N_r, training batch size N_b, target network replacement frequency N^-

Output: the parameters of Q-network

for *episode* $e \in 1, 2, ..., M$ **do**
  Initialize node sequence $\mathbf{x} \leftarrow ()$;
  for $t \in 0, 1, ...$ **do**
    Set state $s \leftarrow \mathbf{x}$, take action a with $\arg\max_a [Q(s, a) + b(s, a)]$;
    Sample next node x^t from environment given (s, a) and receive reward r, and append x^t to $\mathbf{x}$;
    if $|\mathbf{x}| > N_r$ **then**
      delete oldest node $x_{t_{min}}$ from $\mathbf{x}$;
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to B, replacing the oldest tuple if $|B| \geq N_r$;
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim Uniform(B)$;
    Construct target values, one for each of the N_b tuples;
    if s' *is terminal* **then**
      $y_i = r$; break;
    else
      $y_i = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-)$;
    Do a gradient descent step with loss $\|y_i - Q(s, a; \theta)\|^2$;
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps;

while the other parts are identical to DQN. For simplicity, we base our work on DDQN. It would be straightforward to use instead other variants of DQN, such as prioritized experience replay [20], dueling network [27], or bootstrapped DQN [18].

We illustrate the DDQN algorithm with UCB in Algorithm 1. The whole process of BNSL in our framework RBNets is summarized in Algorithm 2.

4 Experimental Validation

In this section, we present extensive experiment results over the performance of our method, against state-of-the-art methods.

4.1 Experiment Setup

The experiments were performed on a PC with 2.10 GHz Intel Xeon E5-2620 processor, 64 GB of RAM, 1024 GB of hard disk space, and running Ubuntu 16.04. About parameter setting, the reward discount factor $\gamma = 1$ and the maximum episodes $M = 300$. We used RMSProp for learning parameters with the learning rate α of 0.001. We used a replay buffer size N_r of 2000, batch size N_b of 200, and target network replacement frequency N^- of 300.

Algorithm 2: RBNets Algorithm

Input: Dataset $\mathcal{D} = \{D_1, D_2, ..., D_m\}$ and variables $\mathcal{V} = \{X_1, X_2, ..., X_n\}$
Output: The Bayesian network structure G^*

1 Extract POPS $\mathcal{P}_i$ for each variable X_i based on Theorem 1, Theorem 2 and Theorem 3;
2 Build a search graph and calculate the value of each edge using Equation (6);
3 Find a good ordering of variables using Algorithm 1;
4 Obtain the parent sets Π_i of each variable X_i according to POPS given the ordering of variables;
5 Rebuild the network structure by the parent sets.

Table 1. The description of datasets sorted according to the number n of variables and the number m of instances. An asterisk indicates that the dataset is from BNR and we generate instances from it, otherwise it is from UCI.

Small			Medium			Large			Very large			Very Large and Massive		
Dataset	n	m	Dataset	n	m	Dataset	n	m	Dataset	n	m	Dataset	n	m
shuttle	9	58000	horse colic	27	368	hailfinder*	56	500	pathfinder*	109	1000	isolet	617	7797
adult	14	48842	water*	32	500	hepar*	72	1000	gas sensor	128	13910	parkinson	754	756
voting	16	435	alarm*	37	1000	ozone	73	2536	semeion	256	1593	androgen	1024	1687
segment	19	2310	sponge	45	76	insurance	86	9000	madelon	500	4400	wikipedia	1068	731

4.2 Datasets

The datasets are from UCI repository[1] and Bayesian Network Repository (BNR)[2]. We removed the lines with missing data and discretized continuous variables into two states using the mean values. The BNR classifies networks as small (less than 20 variables), medium (20–50 variables), large (50–100 variables), very large (100–1000 variables), and massive (more than 1000 variables). The description is depicted in Table 1.

4.3 Baseline Methods

Many existing techniques in heuristic search and exact solver can be used to handle the problem of Bayesian network structure learning. We compare our method with the following two exact methods (A* and GOBNILP), three heuristic methods (GHC, OBS, and ASOBS), and one reinforcement learning based method (RL-BIC).

– **A*** [28] is developed based on the dynamic programming recurrences to learn optimal network structures. It formulates learning optimal Bayesian network as a shortest path finding problem. With the guidance of a consistent heuristic, the algorithm learns an optimal Bayesian network. We use the version 2017 from URLearning[3].

[1] http://archive.ics.uci.edu/ml/.
[2] https://www.bnlearn.com/bnrepository/.
[3] http://www.urlearning.org/.

Table 2. The running time (in seconds) of different methods. Note that the extraction of POPS was computed in a preprocessing step and the running time does not include it. We use the same extracted POPS in different methods. Running time of RBNets includes model training time. Resource limits of 12 h of CPU time and 64 GB of memory were imposed: OT = out of time; OM = out of memory. Bold indicates that the time is the best result among all tested methods. An asterisk indicates that the dataset is generated from BNR.

Dataset	Time (s)						
	A*	GOBNILP	GHC	OBS	ASOBS	RL-BIC	RBNets
shuttle	**0.8**	3.6	5.6	3.2	3.0	2.8	1.2
adult	9.6	**0.9**	28.4	12.8	6.4	3.6	2.3
voting	4.4	3.1	19.2	18.6	12.5	8.3	**2.6**
segment	**2.8**	4.0	26.7	10.2	5.8	6.1	3.5
horse colic	**8.5**	11.8	121.6	64.0	42.3	29.6	12.4
water*	6.2	**4.7**	239.4	43.9	26.6	25.7	15.8
alarm*	70.1	**6.9**	1385.1	482.3	320.7	105.4	80.5
sponge	138.4	20.6	OT	215.6	68.5	29.8	**19.7**
hailfinder*	OM	129.3	OT	673.3	104.3	105.6	**71.1**
hepar*	OM	OT	OT	1204.7	382.4	309.1	**209.2**
ozone	OM	OT	OT	3420.0	715.7	769.2	**526.4**
insurance	OM	OT	OT	3257.8	2802.5	2035.7	**1953.0**
pathfinder*	OM	OM	OT	3070.6	1961.0	1544.9	**1194.1**
gas sensor	OM	OM	OT	3641.2	3409.4	3284.8	**3026.5**
semeion	OM	OM	OT	18530.6	13298.5	12037.0	**10573.2**
madelon	OM	OM	OT	30268.9	22746.6	22453.7	**19281.7**
isolet	OM	OM	OT	38211.4	29682.3	28444.8	**26318.6**
parkinson	OM	OM	OT	OT	31842.1	28373.5	**27805.9**
androgen	OM	OM	OT	OT	39467.4	37009.2	**36256.0**
wikipedia	OM	OM	OT	OT	40369.2	38675.1	**38096.8**

- **GOBNILP** [7] is based on the integer programming (IP) for exact BN learning, which learns Bayesian networks from complete discrete data or from local scores. It adds acyclicity constraints to the ILP during solving in the form of cutting planes. We use the version 1.6.1 with SCIP 3.2.1 of GOBNILP[4].
- **Greedy Hill Climbing (GHC)** [9] examines all possible local changes (edge addition, edge deletion, and edge reversal) in each step and apply the one that leads to the biggest improvement in score and optimizes the network structure. We use the implementation of bnlearn[5] package with version 4.5.
- **OBS** [25] makes use of the topological orderings of variables as a search space, selecting for each ordering the best network consistent with it. This search space is much smaller, makes more global search steps, has a lower branching factor, and avoids costly acyclicity checks.

[4] https://www.cs.york.ac.uk/aig/sw/gobnilp/.
[5] https://www.bnlearn.com/.

- **ASOBS** [19] performs approximated structure learning without constraints on the in-degree. It is made of two parts: parent set identification for exploring the space of possible parent sets of a node; structure optimization for maximizing the score of the resulting structure.
- **RL-BIC** [29] proposes an encoder-decoder model, which takes observable data as input and generates graph adjacency matrices that are used to compute rewards. The reward incorporates both the predefined score function and two penalty terms for enforcing acyclicity.

4.4 Evaluation Metrics

The explicit goal of Bayesian network structure learning is to maximize the BIC score. Therefore, in addition to the running time, we evaluate the performance of methods by the BIC score. The difference in BIC scores between the two alternative networks is an asymptotic approximation of the logarithm of the Bayesian factor, which is the ratio of two posterior probabilities [19]. $\Delta BIC_{1,2} = BIC_1 - BIC_2$ represents the difference between the BIC scores of network net_1 and network net_2. If $\Delta BIC_{1,2} > 0$, it means that net_1 is better than net_2. In order to quantify this metric, the evidence in favor of net_1 is respectively {neutral, positive, strongly positive, very strong} if $\Delta BIC_{1,2}$ is between {0 and 2; 2 and 6; 6 and 10; beyond 10} [19]. In the same way, the evidence in favor of net_2 is respectively {neutral, negative, strongly negative, very negative} if $\Delta BIC_{1,2}$ is between $\{-2$ and 0; -6 and -2; -10 and -6; smaller than $-10\}$.

Table 3. The comparison of RBNets with four heuristic baselines in unknown network structures.

RBNets vs	GHC	OBS	ASOBS	RL-BIC
$\Delta BIC(K)$				
Very positive ($K > 10$)	24	21	15	12
Strongly positive ($6 < K < 10$)	3	4	3	6
Positive ($2 < K < 6$)	2	2	6	5
Neutral ($-2 < K < 2$)	1	3	5	6
Negative ($-6 < K < -2$)	0	0	1	1
Strongly negative ($-10 < K < -6$)	0	0	0	0
Very negative ($K < -10$)	0	0	0	0

4.5 Performance Evaluation of Time

We first tested the running time of different methods in solving the benchmark datasets. We terminate a method early if it runs for more than 12 h on a dataset, which means out of time in our scenario. The result is presented in Table 2, from

Table 4. The comparison of RBNets with four heuristic baselines in known network structures.

RBNets vs	GHC	OBS	ASOBS	RL-BIC
$\Delta BIC(K)$				
Very positive ($K > 10$)	29	23	19	14
Strongly positive ($6 < K < 10$)	0	1	0	5
Positive ($2 < K < 6$)	1	3	2	3
Neutral ($-2 < K < 2$)	0	3	5	4
Negative ($-6 < K < -2$)	0	0	2	2
Strongly negative ($-10 < K < -6$)	0	0	1	2
Very negative ($K < -10$)	0	0	1	0

which we can draw that the running time mainly depends on the scale of the datasets, including the number of variables and instances. The conclusions are as follows.

- Small networks and medium networks datasets are easy for exact algorithms, A* and GOBNILP, while heuristic methods including GHC, OBS, ASOBS, and RL-BIC need more time to find a solution. However, our proposed RBNets achieves satisfactory results and are even better than exact methods in voting and sponge datasets.
- In large networks, it can be challenging for both A* and GOBNILP, either out of memory or out of time. As for the heuristic methods, GHC does not work when number of variables is larger than 40, which is understandable because it examines all possible local changes in each step. ASOBS performs better than OBS because it extends the ordering-based algorithm and provides an effective approach for model selection with reduced computational cost. RL-BIC achieves good performance among baseline methods. Our RBNets has the best performance compared with all baselines.
- In very large and massive networks, A*, GOBNILP, and GHC fail to complete the finding process due to out of memory or out of time. OBS, ASOBS, and RL-BIC can be applied to very large networks, but is still slower than RBNets.
- A* method can easily exceed the memory limit in large datasets, the reason is that it requires all the search information, such as parent and order graphs, to be stored in memory during the search process.

To sum up, the improvement of RBNets is significant in running time when compared with baselines, especially in large, very large, and massive networks.

4.6 Learning Performance from Datasets

In addition to running time, we measure the quality of networks that are learned from datasets of different methods. Based on the datasets mentioned in Table 1,

there are 15 datasets from UCI with unknown network structures and 5 datasets from BNR with known network structures. For the former, we randomly divide each dataset into two subsets of instances, which forms 30 datasets with unknown network structures. For the latter, we generated instances from these networks using logic sampling. Each instance corresponds to a value assignment for all nodes. Then we run each method when given the first 200, 500, 1000, 2000, 3000, and 5000 instances from each dataset, i.e., overall we consider 30 datasets (5 original datasets multiplied by 6 different number of instances) with known network structures.

Then we compare our RBNets with four heuristic methods GHC, OBS, ASOBS, and RL-BIC respectively. It should be noted that exact methods A* and GOBNILP are not appropriate here since they aim to learn the optimal Bayesian networks. Besides, A* and GOBNILP can not work in very large networks unless the in-degree is restricted. A positive ΔBIC means that RBNets yields a network with higher BIC score than the network obtained using other approaches; vice versa for negative values of ΔBIC. The comparison results are shown in Table 3 and Table 4, from which we draw following conclusions.

- The ΔBIC of the learned network is larger than 10 in most cases, implying very effective calculation for the networks learned by RBNets.
- Especially comparing with GHC and OBS, RBNets acts much better than them whether the network structure is known or not. $\Delta BIC > 10$ is obtained in 24/30 cases and 29/30 cases in unknown and known structures, respectively.
- ASOBS and RL-BIC have good performance in a few datasets, e.g., ASOBS leads to $\Delta BIC < -2$ in 1/30 cases in unknown structures and 4/30 cases in known structures. However, they still perform worse than RBNets in most cases.
- RL-BIC yields $\Delta BIC < -2$ in 1/30 cases and 4/30 cases in different data scenarios. However, RBNets performs better than R-RBNets in most cases.

5 Conclusion

In this paper, we discuss the problem of learning Bayesian network structures from a given dataset and scoring function, which has been shown to be NP-hard. The running time and learning performance of traditional methods are not satisfactory, which calls for further research. In this paper, we propose a novel deep Reinforcement learning based Bayesian Network structure learning approach (RBNets). We formulate this problem as a highest-value path problem and calculate the ordering of variables using the Deep Double Q-network algorithm with Upper Confidence Bound based exploration strategy. Then we can reconstruct the structure of Bayesian networks from the potential optimal parent sets and the ordering of variables. Substantial experiments on real and synthetic datasets show the efficiency and effectiveness of our method against baseline methods. RBNets has better performance over running time and BIC score when compared with state-of-the-art methods, especially in large networks.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. (ML) **47**(2–3), 235–256 (2002)
2. de Campos, C.P., Scanagatta, M., Corani, G., Zaffalon, M.: Entropy-based pruning for learning Bayesian networks using BIC. Artif. Intell. (AI) **260**, 42–50 (2018)
3. Campos, C.P.D., Ji, Q.: Efficient structure learning of Bayesian networks using constraints. J. Mach. Learn. Res. (JMLR) **12**, 663–689 (2011)
4. de Campos, L.M., Fernández-Luna, J.M., Gámez, J.A., Puerta, J.M.: Ant colony optimization for learning Bayesian networks. Int. J. Approx. Reason. **31**(3), 291–311 (2002)
5. Chen, C., Yuan, C.: Learning diverse Bayesian networks. In: AAAI Conference on Artificial Intelligence (AAAI), pp. 7793–7800 (2019)
6. Chickering, D.M.: Learning Bayesian networks is NP-complete. Networks **112**(2), 121–130 (1996)
7. Cussens, J.: Bayesian network learning with cutting planes. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 153–160 (2011)
8. Cussens, J., Bartlett, M.: Advances in Bayesian network learning using integer programming. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 182–191 (2013)
9. Friedman, N., Nachman, I., Peér, D.: Learning Bayesian network structure from massive datasets: the "sparse candidate" algorithm. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 206–215 (1999)
10. Gasse, M., Aussem, A., Elghazel, H.: An experimental comparison of hybrid algorithms for Bayesian network structure learning. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012. LNCS (LNAI), vol. 7523, pp. 58–73. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33460-3_9
11. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI Conference on Artificial Intelligence (AAAI), pp. 2094–2100 (2016)
12. Heckerman, D.: A tutorial on learning with Bayesian networks. In: NATO Advanced Study Institute on Learning in Graphical Models, pp. 301–354 (1998)
13. Jaakkola, T., Sontag, D., Globerson, A., Meila, M.: Learning Bayesian network structure using LP relaxations. J. Mach. Learn. Res. (JMLR) **9**, 358–365 (2010)
14. Lee, C., van Beek, P.: Metaheuristics for score-and-search Bayesian network structure learning. In: Canadian Conference on Artificial Intelligence (Canadian AI), pp. 129–141 (2017)
15. Liao, Z.A., Sharma, C., Cussens, J., van Beek, P.: Finding all Bayesian network structures within a factor of optimal. In: AAAI Conference on Artificial Intelligence (AAAI), pp. 7892–7899 (2019)
16. Malone, B., Yuan, C., Hansen, E.A., Bridges, S.: Improving the scalability of optimal Bayesian network learning with external-memory frontier breadth-first branch and bound search. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 479–488 (2011)
17. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
18. Osband, I., Blundell, C., Pritzel, A., Roy, B.V.: Deep exploration via bootstrapped DQN. In: Neural Information Processing Systems (NeurIPS), pp. 4026–4034 (2016)
19. Scanagatta, M., de Campos, C.P., Corani, G., Zaffalon, M.: Learning Bayesian networks with thousands of variables. In: Neural Information Processing Systems (NeurIPS), pp. 1864–1872 (2015)

20. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. In: International Conference on Learning Representations (ICLR) (2016)
21. Silander, T., Myllymaki, P.: A simple approach for finding the globally optimal Bayesian network structure. In: Conference on Uncertainty in Artificial Intelligence (UAI) (2006)
22. Silver, D., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
23. Singh, A.P., Moore, A.W.: Finding optimal Bayesian networks by dynamic programming. In: USENIX Annual Technical Conference (USENIX ATC) (2005)
24. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
25. Teyssier, M., Koller, D.: Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 548–549 (2005)
26. Wang, X., et al.: Ordering-based causal discovery with reinforcement learning. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 3566–3573 (2021)
27. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., de Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning (ICML), pp. 1995–2003 (2016)
28. Yuan, C., Malone, B.M., Wu, X.: Learning optimal Bayesian networks using A* search. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 2186–2191 (2011)
29. Zhu, S., Ng, I., Chen, Z.: Causal discovery with reinforcement learning. In: International Conference on Learning Representations (ICLR) (2020)