An Index Advisor Using Deep Reinforcement Learning

Hai Lan
RMIT University
hai.lan@rmit.edu.au

Zhifeng Bao
RMIT University
zhifeng.bao@rmit.edu.au

Yuwei Peng
Wuhan University
ywpeng@whu.edu.cn

ABSTRACT

We study the problem of index selection to maximize the workload performance, which is critical to database systems. In contrast to existing methods, we seamlessly integrate index recommendation rules and deep reinforcement learning, such that we can recommend single-attribute and multi-attribute indexes together for complex queries and meanwhile support *multiple-index access* to a table. Specifically, we first propose five heuristic rules to generate the index candidates. Then, we formulate the index selection problem as a reinforcement learning task and employ Deep Q Network (DQN) on it. Using the heuristic rules can significantly reduce the dimensions of the action space and state space in reinforcement learning. With the neural network used in DQN, we can model the interactions between indexes better than previous methods. We conduct experiments on various workloads to show its superiority.

KEYWORDS

database system; database configuration; index recommendation; deep reinforcement learning

ACM Reference Format:
Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3412106

1 INTRODUCTION

The index selection problem (ISP) is one of the central issues in database tuning. Informally, given a workload, a dataset, a set of index candidates, and some constraints on indexes to be built (e.g., maximum storage budget or maximum index number), ISP aims to select a subset of index candidates to maximize the performance of the workload while meeting the constraints. It has been proven to be an NP-hard problem [8]. In this paper, we propose an index advisor that integrates index recommendation rules and deep reinforcement learning. The superiority of our method can be exhibited from three aspects, which are also our main contributions.

Aspect 1: it can recommend single-attribute and multi-attribute indexes together for complex queries (e.g., TPC-H[1]).

Aspect 2: it can model the interactions between different indexes.

Aspect 3: it can support the case of *multiple-index access* to a table when recommending indexes (in Aspect 1), which means using more than one index to access a table.

Unfortunately, existing methods fail in one or more of the above aspects, as elaborated next. There are several traditional methods proposed to find appropriate indexes for a workload [1, 3, 7, 10, 13]. Chaudhuri et al. [1] and Valentin et al. [13] propose a two-stage greedy-based method. However, the interactions between different indexes are not considered in [1] (i.e., failing in Aspect 2). An interaction exists between an index a and an index b if the benefit of a is affected by the presence of b and vice-versa [11]. Although such an interaction is considered in [13], it just randomly shuffles indexes several times searching for potentially better indexes (i.e., failing in Aspect 2). ILP [7] and Cophy [3] formulate ISP as a binary integer problem (BIP) and employ a commercial BIP solver. However, these methods do not consider the interactions between different indexes and assume every table in a query can only use one index, which means they fail in Aspects 2 and 3. Accessing a table with multiple indexes is a special form of index interaction and most commercial and open source databases have supported this access method. Schlosser et al. [10] propose a one-stage approach without generating index candidates. At every step, a new single-attribute index is chosen or the existing one is extended by adding the new attribute to consistently maximize the additional performance per additional memory. However, they start from selecting single-attribute indexes, thereby missing greater performance improvement that multi-attribute indexes can bring; moreover, they only consider interactions based on built indexes (i.e., failing in Aspect 2).

Some learning-based methods are proposed to address ISP [4, 9, 12, 14]. Sadri et al. [9] employ Deep Reinforcement Learning (DRL) in ISP for a cluster database. However, they only recommend single-attribute indexes (i.e., failing in Aspect 1) and have not implemented or evaluated their method. The method in [14] does not consider the constraints into the model and only deals with a simple query that involves one table (i.e., failing in Aspect 1). The method in [12] only deals with the query with one table and builds single-attribute indexes (i.e., failing in Aspects 1 and 2). Ding et al. [4] use a neural network to compare the workload cost under different index configurations instead of the what-if caller. However, other parts of the algorithm still take the methods in [1, 13]. Thus it still fails to model the interactions well (i.e., failing in Aspect 2).

In this paper, we formulate the ISP as a DRL problem (Section 3.1) and propose a two-stage approach. Intuitively, DRL can explore more combinations that can better foresee the global impact of introducing new indexes than the traditional greedy based methods.

Yuwei Peng is the corresponding author.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412106

[1] http://www.tpc.org/tpch/

First, we design five heuristic rules to generate the index candidates (Section 3.2). These rules can reduce the dimensions of the action space and the state space in DRL and support recommending single-attribute and multi-attribute indexes together. Second, we train a DRL model (Section 3.3) to select an appropriate subset of candidates as the recommended indexes. Thanks to the neural network used in DRL model, the interactions between indexes can be easily modeled as the interactions between neurons. Same as all existing work, we take the cost estimated by the optimizer of database system as the query's performance. Although we utilize a what-if interface to get the cost without building an index physically, it still needs to call the interface numerous times due to exploration in reinforcement learning (RL), which is time-consuming. Thus, we propose a cost cache to store the cost of query under a specific index configuration to avoid redundant interface calls.

To our best knowledge, this is the first work that integrates heuristic rules and DRL to recommend both single-attribute and multi-attribute indexes for complex queries over multiple tables, and supports the case of multiple-index access to a table in recommending indexes.

2 PROBLEM FORMULATION

A *database* D (of size n) is a set of tables in the form of $\{T_1, T_2, ..., T_n\}$, where T_i is a table. A *workload* W (of size m) is a set of pairs in the form of $\{(Q_1, f_1), (Q_2, f_2), ...(Q_m, f_m)\}$, where Q_i is a query and f_i is the frequency of Q_i. An *index configuration* X is a set of indexes. Every index in X can be a single-attribute or multi-attribute index. $Storage(X)$ denotes the storage space that X takes and $|X|$ denotes the number of indexes in X. A *constraint* on the index to be built is denoted by c. For example, c can be the maximum storage budget, or the maximum index number, etc. The *query cost* $Cost(Q, X)$ indicates the cost of a query Q under an index configuration X. We use the what-if caller [2] to acquire its value. The *workload cost* $Cost(W, X)$ denotes the cost of a workload W under an index configuration X and we can define $Cost(W, X) = \sum_{i=1}^{m} f_i Cost(Q_i, X)$. To this end, we define the ISP.

Definition 2.1 (Index Selection Problem (ISP)). Given a workload W, a database D, a set of constraints C and a set of candidate indexes $\mathcal{X}$, find an index configuration X^* such that X^*=arg $\min_{X \subseteq \mathcal{X}} Cost(W, X)$ subject to all constraints in C.

3 METHODOLOGY

In this section, we first transform the ISP into a reinforcement learning problem. Then, we define the rules used to generate the index candidates. Last, we present the training process in our method and discuss how to deal with the case in which the query cost estimated by the database optimizer is not accurate.

3.1 Formulation of ISP as a DRL Problem

In (D)RL, an agent interacts with environment by actions and rewards. At each step t, the agent uses a policy π to choose an action a_t according to the current state s_t and transitions to a new state s_{t+1}. Then the environment applies the action a_t and returns a reward r_t to the agent. The goal of (D)RL is to learn a policy π, a function that automatically takes an action based on the current state, with the maximum long-term reward (Equation 1). γ is a discount factor, which pays more attention to short-term (long-term) reward when approaching 0 (1).

$$\arg\max_{\pi} \mathbb{E}\left(\sum_{t=0}^{T} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right) \tag{1}$$

In ISP, an index configuration can be regarded as a result of several steps of index selection. At every step, we add one index from the index candidates into the current index configuration until the constraint is violated. Assume X_i represents the index configuration after the i^{th} step and X_0 is the index configuration at the beginning (before index tuning). Then, ISP can be regarded as finding an index selection policy π that selects an appropriate index at every step into index configuration to maximize the performance improvement of W:

$$\arg\max_{\pi} \sum_{t=0}^{T-1} (Cost(W, X_t) - Cost(W, X_{t+1})) \tag{2}$$

$$X_{t+1} = X_t \cup \pi(\mathcal{X}, X_t, W). \tag{3}$$

T is the maximum number of steps while satisfying the constraints.

To this end, we can formulate the ISP as a DRL problem, and the key elements of DRL are defined as follows. **Agent** is our tuning system, which receives the state and reward from the environment and updates the policy to select a suitable index at every step. **Environment** is the database system that needs indexes to be built. **State** means the current state of the agent. State records the information about current built indexes. State is represented using a binary one-hot encoding: a value 1 in a slot indicates the corresponding index has been built and 0 represents its absence. **Action** in our model is choosing an index to build. The goal of index tuning is to minimize the workload cost (i.e., maximize the performance). Thus, we refer the performance gain of the workload after applying the action as **Reward**:

$$r_t = \frac{Cost(W, X_{t-1}) - Cost(W, X_t))}{Cost(W, X_0)} \tag{4}$$

Policy is a function mapping from state to action. RL can be divided into two categories: value-based method (e.g., Q-Learning,DQN [6]) and policy-based method (e.g., DDPG [5]). The output of value-based method is the benefits of all actions, namely Q-value. The action with the largest Q-value is picked with a probability of $1 - \epsilon$ (or a random action is picked with a probability of ϵ), which enables the agent to exploit current knowledge (or explore unknown states).Differently, the output of the policy-based method is the actions.

We adopt DQN as our modelfor the following reasons: (1) In ISP, the action space is discrete, which is the same as DQN and Q-Learning. (2) The state space of ISP is quite large, while Q-Learning is only effective for small state space and thus cannot scale well. Moreover, the state of ISP is easily encoded into vector as the input of DQN. (3) It is difficult to map the actions generated by DDPG into the actions in ISP.

3.2 Rules for Index Candidates Generation

We classify the attributes of one table in a query into five sets:

- `J`: attributes that appear in JOIN conditions.
- `EQ`: attributes that appear in EQUAL conditions.
- `RANGE`: attributes that appear in RANGE conditions.

- O: attributes that appear in GROUP BY, ORDER BY clauses.
- USED: attributes that appear in this query.

With the above defined sets, we propose five rules to guide the index candidates generation:

Rule 1: Construct all single-attribute indexes by using the attributes in J, EQ, RANGE.

Rule 2: When the attributes in O come from the same table, generate the index by using all attributes in O.

Rule 3: If table *a* joins table *b* with multiple attributes, construct indexes by using all join attributes.

Rule 4: Denoting an attribute from J or an index generated by Rule 3 as *j* and an attribute from RANGE as *r*, construct indexes *j* + EQ + *r*, indexes EQ + *r*, indexes *j* + *r*, and indexes *j* + EQ.

Rule 5: If the attribute count in USED is fewer than the maximum number of attributes in indexes defined by users, construct indexes by appending the remaining attributes (attributes in USED but not in the indexes generated from rules above) into the created indexes.

3.3 Model Training

Every episode in RL is an index selection process. At every step in one episode, the model chooses the index having the largest Q-value with a probability of $1 - \epsilon$ or picks a random index. When the users required constraint is broken, the environment will return a *done* flag to finish the current episode.

During the training process, the model will call the what-if interface many times to compute the rewards, which is time-consuming. Thus, we use a cost cache to store the cost of a query under a certain index configuration. Before calling the interface, the model first checks whether the cost is in the cache. If the cost has been stored, we just take the cost from the cache.

3.4 Discussion

In this paper, we assume that the query cost estimated by the optimizer is accurate like many existing methods [1, 3, 7, 10] and take it as the metric for the performance of workload. Meanwhile, we can train our model with the execution time without any changes to get a more accurate model when the optimizer is not convincing.

4 EXPERIMENTS

We conduct experiments to investigate the following questions:

- Q1: How well does our method perform when recommending indexes for complex queries (i.e., Aspect 1 in Section 1)?
- Q2: How well does our method model the interactions between different indexes (i.e., Aspect 2 in Section 1)?
- Q3: How well does our method perform over the queries that get the best performance when accessing a table with multiple indexes (i.e., Aspect 3 in Section 1)?

4.1 Experimental Setting

Implementation and Environment. We implement our prototype in Python and models are written by PyTorch. The prototype is interfaced with PostgreSQL 10.5[2]. We adopt HypoPG[3] as part of the what-if caller implementation which we use to obtain the query cost in our method and compared methods. We employ CPLEX v12.9[4] to solve the BIP problem for ILP [7].

Data and Queries. We employ a 1GB TPC-H database with eight tables and two synthetic workloads, W^o and W^m. W^o is generated by the TPC-H query generator with 14 query templates from TPC-H and 13 templates involve multiple tables. We use W^o to answer Q1. W^m is generated by our generator with 50 query templates; the queries in W^m only fetch tuples from the *LINEITEM* table and most queries in W^m get the best performance when using multiple indexes together. The frequency of a query mentioned in Section 2 is generated randomly in the following experiments. All indexes generated by our proposed rules (Section 3.2) in our experiments are built on three attributes at most. In all experiments, we focus on the B-tree index but our approach can work for all indexes.

Competitors: (1) ILP [7], a traditional approach for index selection problem. Cophy [3] takes the same idea with ILP and has similar performance [3]. Thus, we just compare with ILP. (2) ISRM (Index Selection using Recursive Method) [10], a state-of-the-art traditional greedy-based method, where we choose the index with the largest cost-decrease at every step.

Evaluation Metrics. We use the workload cost $Cost(W, X)$ for effectiveness evaluation, and the recommended index configuration X with a lower value is better.

4.2 Performance Comparison

4.2.1 Index Selection on All Tables (Q1). We compare the performance of W^o under: (1) indexes recommended by our method (DQN) which selects a subset of the candidates generated by the proposed rules in Section 3.2; (2) all indexes in the candidates generated by the proposed rules (ALL-C), which provide the optimal value for DQN; (3) indexes recommended by applying our DQN model to select a subset of single-attribute indexes built on all tables (DQN-S); (4) all single-attribute indexes (ALL-S), which provide the optimal value for DQN-S; (5) indexes recommended by ISMR. There are 61 indexes in ALL-S and 81 indexes in ALL-C. Due to the scalability of ILP, we exclude it in this part.

The result is shown in Figure 1. When the index number equals 1, the costs of W^o under the indexes recommended by DQN-S and ISMR are 25.44. When index number is fewer than 4, the costs of W^o under DQN-S and ISMR are the same. Because they recommend the same single-attribute indexes.

We can observe from Figure 1: (1) W^o cannot get the best performance if only recommending single-attribute indexes by comparing ALL-S and ALL-C. (2) When index number equals 1, the cost of W^o under DQN is much lower than DQN-S and ISMR. Because our model generates the multiple-attribute indexes at the beginning, which supports more queries. (3) When index number is 2 or 3, the cost of W^o under indexes recommended by DQN-S is close to DQN. Because the performance improvement of W^o is from the queries generated from $Q20$ in TPC-H which only needs single-attribute index. (4) When index number is greater than 3, the costs of W^o under DQN-S are higher than ISMR. Because ISMR starts to recommend some multi-attribute indexes after 3. (5) DQN-S and DQN get the optimal performance when index number is 7 and 10 separately. The storage sizes used in ALL-S and ALL-C are 28.03 units and 53.15

[2] https://www.postgresql.org/

[3] https://github.com/HypoPG/hypopg

[4] https://www.ibm.com/analytics/cplex-optimizer

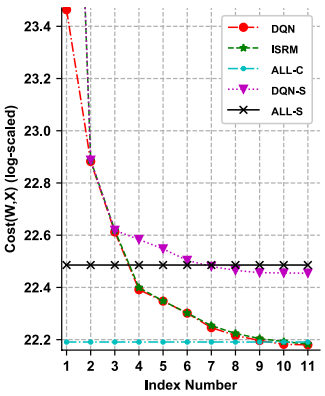


Figure 1: W^o with indexes on all tables

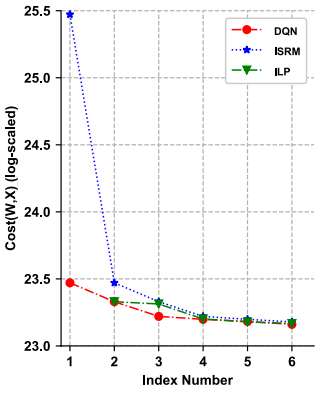
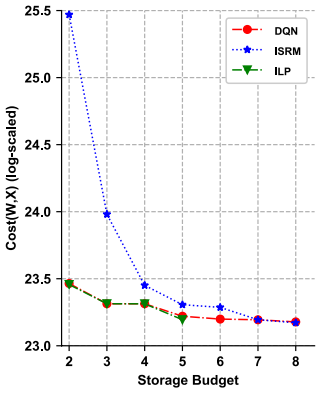


Figure 2: W^o with indexes on two tables



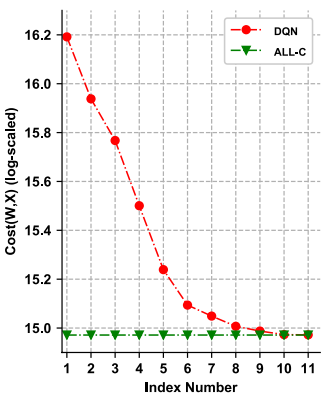


Figure 3: W^m with indexes on one table

units respectively, where each unit indicates 128.57M. However, they are 5.26 units and 10.2 units when DQN-S and DQN achieve the optimal performance. Even the costs of W^o under DQN-S and DQN can be lower than the optimal values. This is because more indexes built will increase the the difficulty for the optimizer to find the optimal physical plan, which means the optimizer may return a suboptimal plan and the cost is higher. (6) When the index number is greater than 1, ISMR is competitive with our method in the current experiment. Acturally, Our method has a lower $Cost(W, X)$ than ISMR under every index number (slightly better by 0.1% ∼ 1%). Due to space limit, we do not report the result w.r.t. the storage budget constraint. However, the result is similar with Figure 1.

4.2.2 Index Selection on Two Tables. Figure 2 presents the comparison result when selecting indexes on two tables, *LINEITEM* and *ORDERS*, under the index number constraint and the storage budget constraint respectively. Our method, ILP, and ISMR all can recommend single/multi-attribute indexes. The storage budget varies from 2 units to 8 units where each unit indicates 128.57M. Note that in Figure 2, ILP does not work when the index number constraint is 1, which is determinded by the idea of atomic configuration [7]. When storage budget is greater than 5, ILP will take more than half an hour to get the results. We can find that our DQN is as competitive as ILP and even better than ILP w.r.t. the constraint of index number, and DQN consistently outperforms ISRM.

4.2.3 Index Selection on one Table for queries benefiting from multiple-index (Q2, Q3). Figure 3 illustrates the comparison when recommeding indexes on table *LINEITEM* under W^m. We compare with the cost of W^m under all indexes in our candidate set (ALL-C), which consists of 69 indexes. As mentioned in Section 1, *multiple-index access* to a table is also a special form of index interaction. This experiment also answers Q2. In ILP, the authors assume every table in a query can only be acccessed with one index at most. Thus, it cannot recommend indexes for W^m and we thereby exclude it. ISMR is sensitive to the order of attributes added in the algorithm. If choosing an attribute that cannot improve the performance at first several steps, ISMR cannot recommend any indexes and stops. However, if choosing a right attribute, the performance of ISMR is similar to ours, which means it only models the interactions based on built indexes.

5 CONCLUSION

We proposed a novel method to solve the ISP by integrating heuristic rules and DRL together. Unlike previous RL based methods, our method can recommend single and multi-attributes index together and support complex queries; moreover, our method can model the interaction between different indexes in a fine-grained manner.

ACKNOWLEDGMENTS

Zhifeng Bao is supported by ARC (DP200102611, DP180102050) and a Google Faculty Research Award. Yuwei Peng is supported by the Key Research and Development Program of China (2016YFB1000701).

REFERENCES

[1] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *VLDB 1997*. 146–155.
[2] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *SIGMOD 1998*. 367–378.
[3] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *Proc. VLDB Endow.* 4, 6 (2011), 362–372.
[4] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *SIGMOD 2019*. 1241–1258.
[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR 2016*.
[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013).
[7] Stratos Papadomanolakis and Anastassia Ailamaki. 2007. An Integer Linear Programming Approach to Database Design. In *ICDE 2007*. 442–449.
[8] Gregory Piatetsky-Shapiro. 1983. The Optimal Selection of Secondary Indices is NP-Complete. *SIGMOD Rec.* 13, 2 (1983), 72–75.
[9] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In *ICDEW*. 158–161.
[10] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *ICDE 2019*.
[11] Karl Schnaitter, Neoklis Polyzotis, and Lise Getoor. 2009. Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. *Proc. VLDB Endow.* 2, 1 (2009), 1234–1245.
[12] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration using Deep Reinforcement Learning. *CoRR* abs/1801.05643 (2018).
[13] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *ICDE 2000*. 101–110.
[14] Jeremy Welborn, Michael Schaarschmidt, and Eiko Yoneki. 2019. Learning Index Selection with Structured Action Spaces. *CoRR* abs/1909.07440 (2019).