Regularized Cost-Model Oblivious Database Tuning with Reinforcement Learning

Debabrota Basu[1], Qian Lin[1], Weidong Chen[1], Hoang Tam Vo[3],
Zihong Yuan[1], Pierre Senellart[1,2], and Stéphane Bressan[1]

[1] School of Computing, National University of Singapore, Singapore
debabrota.basu@u.nus.edu
[2] LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay, Paris, France
[3] SAP Research and Innovation, Singapore

Abstract. In this paper, we propose a learning approach to adaptive performance tuning of database applications. The objective is to validate the opportunity to devise a tuning strategy that does not need prior knowledge of a cost model. Instead, the cost model is learned through reinforcement learning. We instantiate our approach to the use case of index tuning. We model the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. During the reinforcement learning process, we face two important challenges: the unavailability of a cost model and the size of the state space. To address the former, we iteratively learn the cost model, in a principled manner, using regularization to avoid overfitting. To address the latter, we devise strategies to prune the state space, both in the general case and for the use case of index tuning. We empirically and comparatively evaluate our approach on a standard OLTP dataset. We show that our approach is competitive with state-of-the-art adaptive index tuning, which is dependent on a cost model.

1 Introduction

In a recent SIGMOD blog entry [23], Guy Lohman asked "Is query optimization a 'solved' problem?". He argued that current query optimizers and their cost models can be critically wrong. Instead of relying on wrong cost models, Stillger et al. have proposed LEO-DB2, a *learning* optimizer [41]; its enhanced performance with respect to classical query optimizers strengthens the claim of discrepancies introduced by the predetermined cost models. Stillger et al. have proposed LEO-DB2, a *learning* optimizer [41]; its enhanced performance with respect to classical query optimizers strengthens the claim of discrepancies introduced by the predetermined cost models

This is our perspective in this article: we propose a learning approach to performance tuning of database applications. By performance tuning, we mean selection of an optimal physical database configuration in view of the workload. In general, configurations differ in the indexes, materialized views, partitions, replicas, and other parameters. While most existing tuning systems and literature [10,39,40] rely on a predefined cost model, the objective of this work is to validate the opportunity for a tuning strategy to do without.

To achieve this, we propose a formulation of database tuning as a *reinforcement learning* problem (see Section 3). The execution of queries and updates is modelled as a Markov decision process whose states are database configurations, whose actions are configuration changes, and whose rewards are functions of the cost of configuration change and query/update evaluation. This formulation does not rely on a preexisting cost model, rather it learns it.

We present a solution to the reinforcement learning formulation that tackles the curse of dimensionality (Section 4). To do this, we reduce the search space by exploiting the quasi-metric properties of the configuration change cost, and we approximate the cumulative cost with a linear model. We formally prove that, assuming such a linear approximation is sound, our approach converges to an optimal policy for estimating the cost.

We then tackle in Section 5 the problem of overfitting: to avoid instability while learning the cost model, we add a regularization term in learning the cost model. We formally derive a bound on the total regret of the regularized estimation, that is logarithmic in the time step (i.e., in the size of the workload).

We instantiate our approach to the use case of index tuning (Section 6), developing in particular optimizations specific to this use case to reduce the search space. The approaches of Sections 4 and 5 provide us with two algorithms COREIL and rCOREIL to solve the index tuning problem.

We use this case to demonstrate the validity of a cost-model oblivious database tuning with reinforcement learning, through experimental evaluation on a TPC-C workload [31] (see Section 7). We compare the performance with the *Work Function Index Tuning* (WFIT) algorithm [40]. Results show that our approach is competitive yet does not need knowledge of a cost model. While the comparison of WFIT with COREIL establishes reinforcement learning as an effective approach to automatize the index tuning problem, performance of rCOREIL with respect to COREIL demonstrates that the learning performance is significantly enhanced by a crisp estimation of the cost model.

Related work is discussed in Section 2.

This article extends a conference publication [6]. In addition to minor edits and precisions added throughout the paper, the following material is novel: the discussion of related work on reinforcement learning (Section 2.2); the result of convergence to an optimal policy (Proposition 2) and its proof; the introduction of regularization (Section 5), including bounds on regrets (Theorem 1 and Section 6.4), the experimental comparison between regularized and non-regularized versions of our approach (Section 7.4) and the study of the quality of their cost estimators (Section 7.5).

2 Related Work

We now review the related work in two areas of relevance: self-tuning databases, and the use of reinforcement learning for data management applications.

2.1 Automated Database Configuration

Table 1 provides a brief classification of related research in automated database configuration, in terms of various dimensions: the offline or online nature of the algorithm (see further), and the physical design aspects being considered by these works (index selection, vertical partitioning, or mixed design together with horizontal partitioning and replication).

Table 1. Automated physical database design

	Index selection	Vert. partitioning	Mixed
Offline	[1,20,46]	[17,22,34]	Stand-alone: [11,13,27] Parallel DBs: [2,33]
Online	[10,24,25,40]	[3,21,37]	

Offline Algorithms Traditionally, automated database configuration is conducted in an offline manner. In that approach, database administrators (DBAs) identify representative workloads from the trace of historical database queries and updates. That task can be done either manually or with the help of sophisticated tools provided by database vendors. Based on these representative workloads, new database configurations are realized: for example, new beneficial indexes to be created [1,20,46], smart vertical partitioning for reducing I/O costs [17,22,34], or possibly a combination of index selection, partitioning and replication for both stand-alone databases [11,13,27] and parallel databases [2,33].

Online Algorithms Given the increasing complication and agility of database applications, coupled with the introduction of modern database environments such as database-as-a-service, the aforementioned manual task of DBAs, though it can be done at regular times in an offline fashion, becomes even more tedious and problematic. Therefore, it is desirable to design more automated solutions to the database design problem that are able to continuously monitor changes in the workload and react in a timely manner by adapting the database configuration to the new workload. In fact, the problem of online index selection has been well-studied in the past [10, 12, 24, 25, 40]. Generally, these techniques adopt the same working model in which the system continuously tracks the incoming queries for identifying candidate indexes, profiles the benefit of the indexes, and realizes the ones that are most useful for query execution. Specifically, an online approach to physical design tuning (index selection) was proposed in [10]. The essence of the algorithm is to progressively choose the optimal plan at each step by using a case-by-case analysis on the potential benefits that we may lose by not implementing relevant candidate indexes. That is, each new database configuration is selected only when a physical change, i.e., creating or deleting an index, would be helpful in improving system performance. Similarly, a framework for continuous online physical tuning was proposed in [39] where effective indexes are created and deleted in response to the shifting workload. Furthermore, the framework is able to self-regulate its performance by providing explicit mechanism for controlling the overhead of profiling the benefit of indexes.

One of the key components of an index selection algorithm is profiling indexes' benefit, i.e., how to evaluate the cost of executing a query workload with the new indexes as well as the cost of configuration transition, i.e., creating and deleting indexes. To realize this function, most of the aforementioned online algorithms exploit a *what-if optimizer* [14] which returns such estimated costs. For examples, the what-if optimizer of DB2 was used in [40], and the what-if optimizer of SQL Server was employed in [10], while the classical optimizer of PostgreSQL was extended to support what-if analysis in [39]. However, it is well-known that invoking the optimizer for estimating the cost of each query under different configurations is expensive [28]. In this work, we propose an algorithm that does not require the use of a what-if optimizer while being able to adaptively provide a better database configuration in the end, as reported in our experimental results.

More recently, as column-oriented databases have attracted a great deal of attention in both academia and industry, online algorithms for automated vertical partitioning becomes critical for such an emerging breed of database systems [3, 21, 37]. Specifically, a storage advisor for SAP's HANA in-memory database system was proposed in [37] in order to take advantage of both columnar and row-oriented storage layouts. At the core of that storage advisor is a cost model which is used to estimate and compare query execution times for different stores. Similarly, a continuous layout adaptation has been recently been introduced in [3] with the aim to support multiple storage layouts in a single engine which is able to adapt to changing data access patterns. The adaptive store monitors the access patterns of incoming queries through a dynamic window of N queries and devises cost models for evaluating the workload and layout transformation cost. Furthermore, in order to efficiently find a near optimal data layout for a given workload, the hybrid store exploits proper heuristic techniques to prune the immense search space of alternative data layouts. On the contrary, the algorithm introduced in [21] uses data mining techniques for vertical partitioning in database systems. The technique is based on closed item sets mining from a query set and system statistic information at run-time, and hence is able to automatically detect changing workloads and perform a re-partitioning action without the need of interaction from DBAs.

Recently, [9] and [5] have aimed at solving the problem of online index selection for large join queries and data warehouses. Since both approaches use a predefined cost model, similarly to [40], it renders them susceptible to erroneous estimations of the cost model. In our work, we are removing the effects of errors made by the cost model by learning it. This gives our approach more robustness and flexibility than cost-model–dependent ones. Moreover, [9] uses genetic algorithms to optimize the selection of multi-table indexes incrementally. But genetic algorithms generally performs worse than reinforcement learning [35] in this kind of dynamic optimization tasks due to its more exploratory nature. In addition, reinforcement learning agents have a more real-time behaviour than genetic behaviour. In [5], authors use a heuristics approach where they incrementally look for frequent itemsets in a query workload. With the knowledge base acquired from there updates, indexes are generated for the frequent itemsets while eliminating the ones generated for infrequent itemsets. Due to such greedy behaviour, higher variance and instability is expected than for reinforcement learning approaches where a trade-off between exploration and exploitation is reached through learning.

Discussion Compared to the state-of-the-art in online automated database design, our proposed overall approach is more general and has the potential to be applied for various problems such as index selection, horizontal/vertical partitioning design, and in combination with replication as well; we note however that we experiment solely with index tuning in this article. More importantly, as our proposed online algorithm is able to learn the estimated cost of queries gradually through subsequent iterations, it does not need the what-if optimizer for estimating query cost. Therefore, our proposed algorithm is applicable to a wider range of database management systems which may not implement a what-if optimizer or expose its interface to users.

2.2 Reinforcement Learning in Data Management

Reinforcement learning [42] is about determining *the best thing to do next* under an evolving knowledge of the world, in order to reach a goal. This goal is commonly represented as maximization of the cumulative reward obtained while performing some actions. Here each action leads to an individual reward and to a new state, usually in a stochastic manner. *Markov decision processes* (MDPs) [30] are a common model for reinforcement learning scenarios. In this model each action leads to a new state and to a given reward according to a probability distribution that must be learned. This implies an inherent trade-off between exploration (trying out new actions leading to new states and to potentially high rewards) and exploitation (performing actions already known to yield high rewards), a compromise explored in depth in, e.g., the stateless model of multi-armed bandits [4]. Despite being well-adapted to the modelling of uncertain environments, the use of MDPs in data management applications has been limited so far. The use of MDP for modelling data cleaning tasks has been raised in [7]. In that paper, the authors discussed the absence of a straightforward technique to do that because of the huge state space. More generally, the following reasons may explain the difficulties in using reinforcement learning in data management applications:

(i) As in [7], the state space is typically huge, representing all possible partial knowledge of the world. This can be phrased as the *curse of dimensionality*.

(ii) States have complex structures, namely that of the data, or, in our case, of the database configuration.

(iii) Rewards may be delayed, obtained after a long sequence of state transitions. This is for example the case in focused Web crawling, which is a data-centric application domain. Still multi-armed bandits have been successfully applied [16] to this problem.

(iv) Because of data uncertainty, there may be only *partial observability* of the current state. That changes the problem to a partially observable Markov decision process [44].

The last two issues are fortunately not prevalent in the online tuning problem discussed here. This lets us formulate online tuning problem as an MDP and focus on a solution to the first two problems.

3 Problem Definition

Let R be a logical database schema. We can consider R to be the set of its possible database instances. Let S be the set of corresponding physical database configurations of instances of R. For a given database instance, two configurations s and s' may differ in the indexes, materialized views, partitions, replicas, and other parameters. For a given instance, two configurations will be logically equivalent if they yield the same results for all queries and updates.

The cost of changing configuration from $s \in S$ to $s' \in S$ is denoted by the function $\delta(s, s')$. The function $\delta(s, s')$ is not necessarily symmetric, i.e., we may have $\delta(s, s') \neq \delta(s', s)$. This property emerges as the cost of changing configuration from s to s' and the reverse may not be the same. On the other hand, it is a non-negative function and also verifies the identity of indiscernibles: formally, $\delta(s, s') \geqslant 0$ and the equality holds if and only if $s = s'$. Physically this means that there is no free configuration change. As it is always cheaper to do a direct configuration change, we get

$$\forall s, s', s'' \in S \quad \delta(s, s'') \leqslant \delta(s, s') + \delta(s', s'').$$

This is simply the triangle inequality. As δ exhibits the aforementioned properties, it is a quasi-metric on S.

Let Q be a workload, defined as a schedule of queries and updates. For brevity, we refer to both of them as *queries*. To simplify, we consider the schedule to be sequential and the issue of concurrency control orthogonal to the current presentation. Thus, query q_t represents the t^{th} query in the schedule, which is executed at time t.

We model a query q_t as a random variable, whose generating distribution may not be known *a priori*. It means that q_t is only observable at time t. The cost of executing query $q \in Q$ on configuration $s \in S$ is denoted by the function

$cost(s, q)$. For a given query, the *cost* function is always positive as the system have to pay some cost to execute a query.

Let s_0 be the initial configuration of the database. At any time t the configuration is changed from s_{t-1} to s_t with the following events in order:

1. Arrival of query q_t. We call $\hat{q}_t$ the observation of q_t at time t.
2. Choice of the configuration $s_t \in S$ based on $\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_t$ and s_{t-1}.
3. Change of configuration from s_{t-1} to s_t. If no configuration change occurs at time t, then $s_t = s_{t-1}$.
4. Execution of query $\hat{q}_t$ under the configuration s_t.

Thus, the system has to pay the sum of the cost of configuration change and that of query execution during each transition. Now, we define *per-stage cost* as

$$C(s_{t-1}, s_t, \hat{q}_t) \coloneqq \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t).$$

We can phrase in other words the stochastic decision process of choosing the configuration changes as a *Markov decision process* (MDP) [30] where states are database configurations, actions are configuration changes, and penalties (negative rewards) are the per-stage cost of the action. Note that in contrast to the general framework of MDPs, transitions from one state to another on an action are deterministic. Indeed, in this process there is no uncertainty associated with the new configuration when a configuration change is decided. On the other hand, penalties are *stochastic*, as they depend on the query which is a random variable. In the absence of a reliable cost model, the cost of a query in a configuration is not known in advance. This makes penalties *uncertain*.

Ideally, the problem would be to find the sequence of configurations that minimizes the sum of future per-stage costs. We assume an infinite horizon [42], which means an action will affect all the future states and actions of the system. But it makes the cumulative sum of future per-stage costs infinite. One practical way to circumvent this problem is to introduce a *discount factor* $\gamma \in [0, 1)$. Mathematically, it makes the cumulative sum of per-stage costs convergent. Physically, it gives more importance to immediate costs than to costs distant in the future, which is a practical intuition. Now, the problem translates into finding the sequence of configurations that minimize a *discounted cumulative cost* defined with γ. Under Markov assumption, a sequence of configuration changes is represented by a function, called *policy* $\pi \colon S \times Q \to S$. Given the current configuration s_{t-1} and a query $\hat{q}_t$, a policy π determines the next configuration $s_t \coloneqq \pi(s_{t-1}, \hat{q}_t)$.

We define the *cost-to-go* function V^π for a policy π as:

$$V^\pi(s) \coloneqq \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t, \hat{q}_t)\right] \text{ such that } \begin{cases} s_0 = s \\ s_t = \pi(s_{t-1}, \hat{q}_t), t \geqslant 1 \end{cases} \tag{1}$$

where $0 < \gamma < 1$ is the discount factor. The value of $V^\pi(s)$ represents the expected cumulative cost for the following policy π from the current configuration s.

Let $\mathcal{U}$ be the set of all policies for a given database schema. Our problem can now be formally phrased as to minimize the expected cumulative cost, i.e., to find an optimal policy π^* such that

$$\pi^* := \arg\min_{\pi \in \mathfrak{U}} V^{\pi}(s_0)$$

where the initial state s_0 is given.

4 Adaptive Database Tuning

4.1 Algorithm Framework

In order to find the optimal policy π^*, we start from an arbitrary policy π, compute an estimation of its cost-to-go function, and incrementally attempt to improve it using the current estimate of the cost-to-go function $\overline{V}$ for each $s \in S$. This strategy is known as *policy iteration* [42] in the reinforcement learning literature.

Traditionally, policy iteration functions as follows. Assuming the probability distribution of q_t is known in advance, we improve the cost-to-go function $\overline{V}^{\pi_t}$ of the policy π_t at iteration t using

$$\overline{V}^{\pi_t}(s) = \min_{s' \in S} \Big(\delta(s, s') + \mathbb{E}\left[cost(s', q)\right] + \gamma \overline{V}^{\pi_{t-1}}(s') \Big) \tag{2}$$

We obtain the updated policy as $\arg\min_{\pi_t \in \mathfrak{U}} \overline{V}^{\pi_t}(s)$. The algorithm terminates when there is no change in the policy. The proof of optimality and convergence of policy iteration can be found in [29].

Unfortunately, policy iteration suffers from several problems. First, there may not be any proper model available beforehand for the cost function $cost(s, q)$. Second, the curse of dimensionality [29] makes the direct computation of $\overline{V}$ hard. Third, the probability distribution of queries is not assumed to be known *a priori*, making it impossible to compute the expected cost of query execution $\mathbb{E}\left[cost(s', q)\right]$.

Algorithm 1 Algorithm Framework

```
1: Initialization: an arbitrary policy $\pi_0$ and a cost model $C_0$
2: Repeat till convergence
3:     $\overline{V}^{\pi_{t-1}}$ ← approximate using a linear projection over $\phi(s)$
4:     $C^{t-1}$ ← approximate using a linear projection over $\eta(s, \hat{q}_t)$
5:     $\pi_t$ ← $\arg\min_{s \in S'} \left(C^{t-1} + \gamma \overline{V}^{\pi_{t-1}}(s)\right)$
6: End
```

Instead, we apply the basic framework shown in Algorithm 1. The initial policy π_0 and cost model C_0 can be initialized arbitrarily or using some intelligent heuristics. In line 5 of Algorithm 1, we have tried to overcome the issues at the root of the curse of dimensionality by juxtaposing the original problem with approximated per-stage cost and cost-to-go function. Firstly, we map a configuration to a vector of associated feature $\phi(s)$. Then, we approximate the cost-to-go function by a linear model $\theta^T\phi(s)$ with parameter θ. It is extracted from a reduced subspace S' of configuration space S that makes the search for optimal policy computationally cheaper. Finally, we learn the per-stage cost $C(s, s', \hat{q})$ by a linear model $\zeta^T\eta(s, \hat{q})$ with parameter ζ. This method does not need any prior knowledge of the cost model, rather it learns the model iteratively. Thus, we have resolved shortcomings of policy iteration and the need of predefined cost model for the performance tuning problem in our algorithm. These methods are depicted and analysed in the following sections.

4.2 Reducing the Search Space

In order to reduce the size of search space in line 5 of Algorithm 1, we filter the configurations that satisfy certain necessary conditions deduced from an optimal policy.

Proposition 1. *Let s be any configuration and $\hat{q}$ be any observed query. Let π^* be an optimal policy. If $\pi^*(s, \hat{q}) = s'$, then $cost(s, \hat{q}) - cost(s', \hat{q}) \geqslant 0$. Furthermore, if $\delta(s, s') > 0$, i.e., if the configurations certainly change after a query, then $cost(s, \hat{q}) - cost(s', \hat{q}) > 0$.*

Proof. Since $\pi^*(s, \hat{q}) = s'$, we have

$$\begin{aligned}
&\delta(s, s') + cost(s', \hat{q}) + \gamma V(s') \\
&\leqslant cost(s, \hat{q}) + \gamma V(s) \\
&= cost(s, \hat{q}) + \gamma \mathbb{E}\left[\min_{s''} \left(\delta(s, s'') + cost(s'', \hat{q}) + \gamma V(s'')\right)\right] \\
&\leqslant cost(s, \hat{q}) + \gamma\delta(s, s') + \gamma V(s'),
\end{aligned}$$

where the second inequality is obtained by exploiting triangle inequality $\delta(s, s'') \leqslant \delta(s, s') + \delta(s', s'')$, as δ is a quasi-metric on S.

This infers that

$$cost(s, \hat{q}) - cost(s', \hat{q}) \geqslant (1 - \gamma)\delta(s, s') \geqslant 0.$$

The assertion follows. □

By Proposition 1, if π^* is an optimal policy and $s' = \pi^*(s, \hat{q}) \neq s$, then $cost(s, \hat{q}) > cost(s', \hat{q})$. Thus, we can define a reduced subspace as

$$S_{s,\hat{q}} = \{s' \in S \mid cost(s, \hat{q}) > cost(s', \hat{q})\}.$$

Hence, at each time t, we can solve

$$\pi_t = \underset{s \in S_{s_{t-1}, \hat{q}_t}}{\arg\min} \left(\delta(s_{t-1}, s) + cost(s, \hat{q}_t) + \gamma \overline{V}^{\pi_{t-1}}(s)\right). \tag{3}$$

Next, we design an algorithm that converges to an optimal policy through searching in the reduced set $S_{s,\hat{q}}$.

4.3 Modified Policy Iteration with Cost Model Learning

We calculate the optimal policy using the *least square policy iteration* (LSPI) [18]. If for any policy π, there exists a vector $\boldsymbol{\theta}$ such that we can approximate $V^{\pi}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for any configuration s, then LSPI converges to the optimal policy. This mathematical guarantee makes LSPI a useful tool to solve the MDP as defined in Section 3. But the LSPI algorithm needs a predefined cost model to update the policy and evaluate the cost-to-go function. It is not obvious that any form of cost model would be available and as mentioned in Section 1, pre-defined cost models may be critically wrong. This motivates us to develop another form of the algorithm, where the cost model can be equivalently obtained through learning.

Assume that there exists a feature mapping $\boldsymbol{\eta}$ such that $cost(s, q) \approx \boldsymbol{\zeta}^T \boldsymbol{\eta}(s, q)$ for some vector $\boldsymbol{\zeta}$. Changing the configuration from s to s' can be considered as executing a special query $q(s, s')$. Therefore we approximate

$$\delta(s, s') = cost(s, q(s, s')) \approx \boldsymbol{\zeta}^T \boldsymbol{\eta}(s, q(s, s')).$$

The vector $\boldsymbol{\zeta}$ can be updated iteratively using the well-known *recursive least squares estimation* (RLSE) [45] as shown in Algorithm 2, where $\boldsymbol{\eta}^t = \eta(s_{t-1}, \hat{q}_t)$ and $\hat{\epsilon}^t = (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta^t} - cost(s_{t-1}, \hat{q}_t)$ is the prediction error. Combining RLSE with LSPI, we get our cost-model oblivious algorithm as shown in Algorithm 3.

Algorithm 2 Recursive least squares estimation.

```
procedure RLSE($\hat{\epsilon}^t, \overline{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t$)
    $\gamma^t \leftarrow 1 + (\boldsymbol{\eta}^t)^T \overline{B}^{t-1} \boldsymbol{\eta}^t$
    $\overline{B}^t \leftarrow \overline{B}^{t-1} - \frac{1}{\gamma^t}(\overline{B}^{t-1} \boldsymbol{\eta}^t (\boldsymbol{\eta}^t)^T \overline{B}^{t-1})$
    $\boldsymbol{\zeta}^t \leftarrow \boldsymbol{\zeta}^{t-1} - \frac{1}{\gamma^t} \overline{B}^{t-1} \boldsymbol{\eta}^t \hat{\epsilon}^t$
    return $\overline{B}^t, \boldsymbol{\zeta}^t$.
end procedure
```

In Algorithm 3, the vector $\boldsymbol{\theta}$ determines the current policy. We can make a decision by solving the equation in line 6. The values of $\delta(s_{t-1}, s)$ and $cost(s, \hat{q}_t)$ are obtained from the approximation of the cost model. The vector $\boldsymbol{\theta}^t$ is used to approximate the cost-to-go function following the current policy. If $\boldsymbol{\theta}^t$ converges, then we update the current policy (line 14–16).

Instead of using any heuristics we initialize policy π_0 as initial configuration s_0 and the cost-model C_0 as 0, as shown in the lines 1–3 of Algorithm 3.

Proposition 2. *If for any policy π, there exists a vector $\boldsymbol{\theta}$ such that $V^{\pi}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for any configuration s, Algorithm 3 will converge to an optimal policy.*

Proof. Let $\mathcal{V}\colon S \to \mathbb{R}$ be a set of bounded, real-valued functions. Then $\mathcal{V}$ is a Banach space with the norm $\|v\| = \|v\|_{\infty} = \sup |v(s)|$ for any $v \in \mathcal{V}$.

Algorithm 3 Least squares policy iteration with RLSE.

```
Initialize the configuration $s_0$.
Initialize $\boldsymbol{\theta}^0 = \boldsymbol{\theta} = \mathbf{0}$ and $B^0 = \epsilon I$.
Initialize $\boldsymbol{\zeta}^0 = \mathbf{0}$ and $\overline{B}^0 = \varepsilon I$.
for t=1,2,3,... do
    Let $\hat{q}_t$ be the just received query.
    $s_t \leftarrow \underset{s \in S_{s_{t-1},\hat{q}_t}}{\arg\min} \; (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, q(s_{t-1}, s)) + (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s, \hat{q}_t) + \gamma \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$
    Change the configuration to $s_t$.
    Execute query $\hat{q}_t$.
    $\hat{C}^t \leftarrow \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t)$.
    $\hat{\epsilon}^t \leftarrow (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, \hat{q}_t) - cost(s_{t-1}, \hat{q}_t)$
    $B^t \leftarrow B^{t-1} - \frac{B^{t-1}\boldsymbol{\phi}(s_{t-1})(\boldsymbol{\phi}(s_{t-1}) - \gamma\boldsymbol{\phi}(s_t))^T B^{t-1}}{1 + (\boldsymbol{\phi}(s_{t-1}) - \gamma\boldsymbol{\phi}(s_t))^T B^{t-1}\boldsymbol{\phi}(s_{t-1})}$.
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \frac{(\hat{C}^t - (\boldsymbol{\phi}(s_{t-1}) - \gamma\boldsymbol{\phi}(s_t)^T \boldsymbol{\theta}^{t-1}) B^{t-1}\boldsymbol{\phi}(s_{t-1})}{1 + (\boldsymbol{\phi}(s_{t-1}) - \gamma\boldsymbol{\phi}(s_t))^T B^{t-1}\boldsymbol{\phi}(s_{t-1})}$.
    $(\overline{B}^t, \boldsymbol{\zeta}^t) \leftarrow RLSE(\hat{\epsilon}^t, \overline{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t)$
    if $(\boldsymbol{\theta}^t)$ converges then
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\boldsymbol{t}}$.
    end if
end for
```

If we redefine our problem in the reduced search space, we get:

$$\begin{aligned} &\underset{\pi \in \mathcal{U}}{\arg\min}\, \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \left(\delta(s_{t-1}, s_t) + cost(s_t, q)\right)\right] \\ &\text{such that:}\;\; s_t = \pi(s_{t-1}, q), \quad s_t \in S_{s_{t-1},q}, \quad \text{for } t \geqslant 1 \end{aligned} \tag{4}$$

Then Algorithm 3 is analogous to LSPI over the reduced search space. For this new problem given by Equation (4), Algorithm 3 converges to a unique cost-to-go function $\tilde{V} \in \mathcal{V}$. We need to show that $V^* = \tilde{V}$. That means we need to prove the cost-to-go function estimate by Algorithm 3 is the optimal one.

Let us define the process of updating policy as a mapping $\mathcal{M}\colon \mathcal{V} \to \mathcal{V}$. Now based on Equation (2), it can be expressed as

$$\mathcal{M}v(s) = \mathbb{E}\big[\min_{s' \in S_{s,q}} \left(\delta(s, s') + cost(s', q) + \gamma v(s')\right)\big].$$

For a particular configuration s and query q, let

$$a^*_{s,q}(v) = \arg\min_{s' \in S_{s,q}} \left(\delta(s, s') + cost(s', q) + \gamma v(s')\right).$$

Assume that $\mathcal{M}v(s) \geqslant \mathcal{M}u(s)$. Then

$$\begin{aligned}
0 &\leqslant \mathcal{M}v(s) - \mathcal{M}u(s) \\
&= \mathbb{E}\left[\delta(s, a^*_{s,q}(v)) + \mathit{cost}(a^*_{s,q}(v), q) + \gamma v(a^*_{s,q}(v))\right] \\
&\quad - \mathbb{E}\left[\delta(s, a^*_{s,q}(u)) + \mathit{cost}(a^*_{s,q}(u), q) + \gamma u(a^*_{s,q}(u))\right] \\
&\leqslant \mathbb{E}\left[\delta(s, a^*_{s,q}(u)) + \mathit{cost}(a^*_{s,q}(u), q) + \gamma v(a^*_{s,q}(u))\right] \\
&\quad - \mathbb{E}\left[\delta(s, a^*_{s,q}(u)) + \mathit{cost}(a^*_{s,q}(u), q) + \gamma u(a^*_{s,q}(u))\right] \\
&= \gamma \mathbb{E}\left[v(a^*_{s,q}(u)) - u(a^*_{s,q}(u))\right] \\
&\leqslant \gamma \mathbb{E}\left[\|v - u\|\right] = \gamma \|v - u\|.
\end{aligned}$$

Thus we can conclude, $|\mathcal{M}v(s) - \mathcal{M}u(s)| \leqslant \gamma |v(s) - u(s)|$ for all configuration $s \in S$. From the definition of our norm, we can write

$$\sup_{s \in S} |\mathcal{M}v(s) - \mathcal{M}u(s)| = \|\mathcal{M}v - \mathcal{M}u\| \leqslant \gamma \|v - u\|.$$

This means that if $0 \leqslant \gamma < 1$, $\mathcal{M}$ is a contraction mapping. By [29, Proposition 3.10.2], there exists a unique v^* such that $\mathcal{M}v^* = v^*$, such that for an arbitrary v^0, the sequence v^n generated by $v^{n+1} = \mathcal{M}v^n$ converges to v^*. By the property of convergence of LSPI [18], $v^* = \tilde{V}$. From Proposition 1, the optimal cost-to-go function V^* also satisfies $\mathcal{M}V^* = V^*$. Hence $V^* = \tilde{V}$ and the property of convergence of LSPI is preserved in Algorithm 3. □

5 Adaptive Database Tuning with Regularized Cost-Model Learning

In the results that we will present in Section 7.3, we will observe a higher variance of Algorithm 3 for index tuning than that of the state-of-art WFIT algorithm [40]. This high variance is caused mainly due to the absence of the cost model. As Algorithm 3 decides the policy depending on the estimated cost model, any error in the cost model causes instability in its outcome.

The process of cost-model estimation by accumulating information of incoming queries is analogous to approximating a function online from its incoming samples. Here, the function is the per-stage cost model $C\colon\ S \times S \times \widetilde{Q} \to \mathbb{R}$. Here, $\widetilde{Q}$ is the extended set of queries given by $Q \cup \{q(s, s') \mid s, s' \in S\}$. We obtain this $\widetilde{Q}$ by considering configuration updates as special queries, as explained in Section 4.3. Now, the per-stage cost function can be defined as

$$C(s_{t-1}, s_t, \hat{q}_t) = \mathit{cost}(s_{t-1}, q(s_{t-1}, s_t)) + \mathit{cost}(s_t, \hat{q}_t)$$

This equation shows that if we consider changing the configuration from s to s' as executing a special query $q(s, s')$, approximating the function $\mathit{cost}\colon\ S \times \widetilde{Q} \to \mathbb{R}$ in turn approximates the per-stage cost.

As explained in the previous section, we approximate *cost* online using linear projection to the feature space of the observed state and query. At each step we obtain some vector $\boldsymbol{\zeta}$ such that $cost(s,q) \approx \boldsymbol{\zeta}^T \boldsymbol{\eta}(s,q)$. Here, $\eta(s,q)$ is the feature vector corresponding to state s and query q. In order to obtain the optimal approximation, we initialize with an arbitrary ζ and then recursively improve our estimate of $\boldsymbol{\zeta}$ using recursive least squares estimation (RLSE) algorithm [45]. But the issues with RLSE are:

i. It tries to minimize the square error per step

$$\hat{\epsilon}_t^2 = \left((\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta^t} - cost(s_{t-1}, \hat{q}_t)\right)^2$$

which is highly sensitive to outliers. If RLSE faces some query or configuration update which is very different from the previously observed queries, the estimation of $\boldsymbol{\zeta}$ can change drastically.

ii. The algorithm becomes unstable if the components of $\boldsymbol{\eta}(s,q)$ are highly correlated. This may happen when the algorithm passes through a series of related queries.

As the reinforcement learning algorithm uses the estimated cost model to decide policies and to evaluate them, error or instability in the estimated cost model at any step affects its performance. Specifically, large deviations arise in the estimated cost model due to the queries which are far from the previously learned distribution. This costs the learning algorithm some time to adapt. It also affects the policy and evaluation of the present state and action. We thus propose to use a *regularized* cost-model estimator instead of RLSE, which is less sensitive to outliers and relatively stable, so as to improve the performance of Algorithm 3 and decrease its variance.

5.1 Regularized Cost-Model Estimator

In order to avoid the effect of outliers, we can penalize high variance of $\boldsymbol{\zeta}$ by adding a regularization term with the squared prediction error of RLSE. Thus at time step t, the new estimator will try to find

$$\boldsymbol{\zeta}^t = \arg\min_{\zeta} P^t \qquad \text{given} \qquad \hat{\epsilon}_t, \overline{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t \tag{5}$$

such that:

$$\begin{aligned} P^t &\coloneqq \hat{\epsilon}_t^2 + \lambda \|\boldsymbol{\zeta}^{t-1}\|_2^2 \\ &= \left(\langle \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2 + \lambda \langle \boldsymbol{\zeta}^{t-1}, \boldsymbol{\zeta}^{t-1} \rangle. \end{aligned}$$

Here, $\lambda > 0$ is the regularization parameter. Square of L_2-norm, $\|\boldsymbol{\zeta}\|_2^2$, is the regularization function. $\boldsymbol{\eta^t} \coloneqq \boldsymbol{\eta}(s_{t-1}, \hat{q}_t)$ is the feature vector of state s_{t-1} and query $\hat{q}_t$. We call this squared error the *loss function* L_t defined at time t for a given choice of $\boldsymbol{\zeta}^t$. Thus,

$$L_t(\boldsymbol{\zeta}^t) \coloneqq \left(\langle \boldsymbol{\zeta}^t, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2.$$

The dual of this problem can be considered as picking up such an $\boldsymbol{\zeta}^t$ inside an n-dimensional Euclidean ball $\mathbb{B}_\lambda^n$ of radius $s(\lambda)$ that minimizes the error $\hat{\epsilon}_t^2$. From an optimization point of view, we choose $\boldsymbol{\zeta}^t$ inside $\mathbb{B}_\lambda^n := \left\{\boldsymbol{\zeta} \mid \|\boldsymbol{\zeta}\|_2^2 \leqslant s(\lambda) \text{ and } \boldsymbol{\zeta} \in \mathbb{R}^n\right\}$ rather than searching for it in the whole space $\mathbb{R}^n$. This estimator penalizes any drastic change in the cost model due to some outlier query. If some query tries to pull $\boldsymbol{\zeta}^t$ out of $\mathbb{B}_\lambda^n$, this estimator regularizes $\boldsymbol{\zeta}^t$ at the boundary. It also induces sparsity in the components of estimating vector $\boldsymbol{\zeta}$ that eliminates the instability due to highly correlated queries.

Algorithm 4 Regularized cost-model estimation.

```
Initialize $\boldsymbol{\zeta}^0 = \mathbf{0}$ and $R^0 = \varepsilon I$.
for t=1,2,3,... do
    $\hat{\epsilon}_t \leftarrow (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}^t - cost(s_{t-1}, \hat{q}_t)$
    $\gamma^t \leftarrow \lambda + (\boldsymbol{\eta}^t)^T R^{t-1} \boldsymbol{\eta}^t$
    $R^t \leftarrow R^{t-1} - \frac{1}{\gamma^t}(R^{t-1}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T R^{t-1})$
    $\boldsymbol{\zeta}^t \leftarrow \boldsymbol{\zeta}^{t-1} - \frac{1}{\gamma^t} R^{t-1}\boldsymbol{\eta}^t \hat{\epsilon}_t$
    return $R^t, \boldsymbol{\zeta}^t$
end for
```

The online penalized cost-model estimation algorithm obtained from this formulation is shown in Algorithm 4. Generally, the optimal values of ε and λ are decided using a cross-validation procedure. In Section 6.4, we are going to derive the optimal value of ε and a probable estimation for λ for the index tuning problem. This will decide optimal values of the hyper-parameters for a given set of workload with theoretical performance bounds.

5.2 Performance Bound

We can depict this online cost-model estimation task as a simple game between a decision maker and an adversary [36]. In database tuning, the decision maker is our cost-model estimating algorithm and the adversary is the workload providing an uncertain sequence of queries. Then, we can formulate the game as Algorithm 5.

Algorithm 5 Cost-model Estimation Game.

```
Initialize $\boldsymbol{\zeta}^0 = \mathbf{0}$.
for t=1,2,3,..., T do
    Algorithm 4 picks $\boldsymbol{\zeta}^t \in \mathbb{B}_\lambda^n$ according to Equation (5)
    Adversary picks $(\boldsymbol{\eta}_t, c_t)$
    Algorithm suffers loss $L_t(\boldsymbol{\zeta}^t)$
end for
```

We can define the regret of this game after time step T as,

$$Reg_T \coloneqq \sum_{t=1}^{T} L_t(\boldsymbol{\zeta}^t) - \sum_{t=1}^{T} L_t(\boldsymbol{\zeta}^{\text{OPT}}) \tag{6}$$

where $\boldsymbol{\zeta}^{\mathbf{OPT}}$ is the solution picked up by an offline expert that minimizes the cumulative loss after time step T. Reg_T is the difference between cumulative sum of errors up to time T obtained using Algorithm 4 and the optimal offline algorithm. This regret term captures deviation of the cost-model estimated by the Algorithm 5 from the computable optimal cost model.

As the loss function $L(\boldsymbol{\zeta})$ is the square of the error between estimated and actual values of cost at time t, it is a convex function over the set of $\boldsymbol{\zeta}$. According to the analysis given in [36], we can canonically describe our estimation model as a scheme to develop a Legendre potential function $\boldsymbol{\Phi}(\boldsymbol{\zeta}^{\text{OPT}})$ with time t for the given workload, where the initial value of potential is given by:

$$\boldsymbol{\Phi}_0(\boldsymbol{\zeta}^{\text{OPT}}) \coloneqq \|\boldsymbol{\zeta}^{\text{OPT}}\|^2$$

and its value at time t is updated as

$$\boldsymbol{\Phi}_t(\boldsymbol{\zeta}^{\text{OPT}}) \coloneqq \boldsymbol{\Phi}_{t-1}(\boldsymbol{\zeta}^{\text{OPT}}) + \frac{1}{\lambda} L_t(\boldsymbol{\zeta}^t).$$

Now, we can re-write Equation (5) as:

$$\boldsymbol{\zeta}^t = \underset{\boldsymbol{\zeta} \in \mathbb{B}_\lambda^n}{\arg\min} \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\text{OPT}}, \boldsymbol{\zeta}^{t-1}) + \frac{1}{\lambda} \left(\boldsymbol{\nabla} L_{t-1}(\boldsymbol{\zeta}_{t-1})\right)^T \boldsymbol{\zeta}^{t-1} \right] \tag{7}$$

Here, $D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\text{OPT}}, \boldsymbol{\zeta}^{t-1})$ is the Bregman divergence [26] between $\boldsymbol{\zeta}^{\text{OPT}}$ and $\boldsymbol{\zeta}^{t-1}$ along the potential field $\boldsymbol{\Phi}_0(\boldsymbol{\zeta}^{\text{OPT}})$. This term in Equation (7) inclines Algorithm 4 to choose such a $\boldsymbol{\zeta}$ which is nearest to optimal $\boldsymbol{\zeta}^{\text{OPT}}$ on the $\|\boldsymbol{\zeta}\|^2$ manifold. Also, $\boldsymbol{\nabla} L_{t-1}(\boldsymbol{\zeta}_{t-1})^T \boldsymbol{\zeta}^{t-1}$ is the change of the loss function in the direction of $\boldsymbol{\zeta}^{t-1}$. Minimization of this term is equivalent to selection of such a $\boldsymbol{\zeta}^t$ that minimizes the corresponding loss. Thus, the $\boldsymbol{\zeta}^t$ picked up by the Algorithm is the one that minimizes a linear combination of these two terms weighted by λ. From this formulation we can obtain the following lemma for the regret bound.

Lemma 1. *After time step T, the upper bound of the regret of Algorithm 4 can be given by*

$$Reg_T \leqslant \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{1}{\lambda} \sum_{t=1}^{T} \hat{\epsilon_t}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t. \tag{8}$$

Proof. Applying Theorem 1 of [43] on Equation (7) we get the inequalities,

$$\begin{aligned} Reg_T &\leqslant \lambda \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\text{OPT}}, \boldsymbol{\zeta}^0) - D_{\boldsymbol{\Phi}_T}(\boldsymbol{\zeta}^{\text{OPT}}, \boldsymbol{\zeta}^{T+1}) + \sum_{t=1}^{T} D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) \right] \\ &\leqslant \lambda \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\text{OPT}}, \boldsymbol{\zeta}^0) + \sum_{t=1}^{T} D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) \right]. \end{aligned}$$

From the definition of the Legendre potential we get:

$$
\begin{aligned}
\boldsymbol{\Phi}_t(\boldsymbol{\zeta}) &= \boldsymbol{\Phi}_{t-1}(\boldsymbol{\zeta}) + \frac{1}{\lambda} L_t(\boldsymbol{\zeta}) \\
&= \|\boldsymbol{\zeta}\|^2 + \sum_{t=1}^{T} \left(\langle \boldsymbol{\zeta}, \boldsymbol{\eta}^{\boldsymbol{t}} \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2 \\
&= \boldsymbol{\zeta}^T \left(I + \frac{1}{\lambda} \sum_{t=1}^{T} \boldsymbol{\eta}^t (\boldsymbol{\eta}^t)^T \right) \boldsymbol{\zeta} - \boldsymbol{\zeta}^T \left(\frac{1}{\lambda} \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t) \boldsymbol{\eta}^t \right) + \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t)^2 \\
&= \boldsymbol{\zeta}^T (R^T)^{-1} \boldsymbol{\zeta} - \boldsymbol{\zeta}^T b_T + C_T
\end{aligned}
$$

where $b_T = \sum_{t=1}^{T} c^t \boldsymbol{\eta}^t$ and $C_T = \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t))^2$. Thus, the dual of the potential can be given by

$$
\boldsymbol{\Phi}_t^*(\boldsymbol{\zeta}) = \boldsymbol{\zeta}^T R^T \boldsymbol{\zeta} - 2\boldsymbol{\zeta}^T R^T b_T + (b_T)^T R^T b_T
$$

Now, from the definition of $\boldsymbol{\Phi}_0$ and properties of Bregman divergence,

$$
\begin{aligned}
D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) &= D_{\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) \\
&= \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2
\end{aligned}
$$

and

$$
\begin{aligned}
D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) &= D_{\boldsymbol{\Phi}_t^*}\left(\boldsymbol{\nabla}\boldsymbol{\Phi}_t(\boldsymbol{\zeta}^{t+1}), \boldsymbol{\nabla}\boldsymbol{\Phi}_t(\boldsymbol{\zeta}^t)\right) \\
&= D_{\boldsymbol{\Phi}_t^*}\left(\mathbf{0}, \boldsymbol{\nabla}\boldsymbol{\Phi}_t(\boldsymbol{\zeta}^t)\right) \\
&= D_{\boldsymbol{\Phi}_t^*}\left(\mathbf{0}, \frac{1}{\lambda}\boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t)\right) \\
&= \frac{1}{\lambda^2}\left(\boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t)\right)^T R^t \left(\boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t)\right) \\
&= \frac{1}{\lambda^2}\left(\langle \boldsymbol{\zeta}, \boldsymbol{\eta}^t \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t \\
&= \frac{1}{\lambda^2} \hat{\epsilon_t}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t.
\end{aligned}
$$

By replacing these results in the aforementioned inequality we get:

$$
Reg_T \leqslant \lambda \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{1}{\lambda} \sum_{t=1}^{T} \hat{\epsilon_t}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t. \qquad \square
$$

Lemma 2. *If $R^0 \in \mathbb{R}^{n \times n}$ and invertible,*

$$
(\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t = 1 - \frac{\det(R^t)}{\det(R^{t-1})} \quad \forall t = 1, 2, \ldots, T \tag{9}
$$

Proof. From [19], we get if there exists an invertible matrix $B \in \mathbb{R}^{n \times n}$ such that $A = B + \boldsymbol{x}\boldsymbol{x}^T$, where $\boldsymbol{x} \in \mathbb{R}^n$, then

$$
\boldsymbol{x}^T A^{-1} \boldsymbol{x} = 1 - \frac{\det(B)}{\det(A)} \tag{10}
$$

As, per Algorithm 4, $R^0 = \varepsilon I$, it is invertible. Since $(R^t)^{-1} = (R^{t-1})^{-1} + \boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T$, by the Sherman–Morrison formula, all R^t's are invertible for $t \geqslant 0$. Thus, simply replacing A by $(R^t)^{-1}$ and B by $(R^{t-1})^{-1}$ in Equation (10), we obtain

$$(\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t = 1 - \frac{\det((R^{t-1})^{-1})}{\det((R^t)^{-1})} = 1 - \frac{\det(R^t)}{\det(R^{t-1})}$$

since, $\det((R^t)^{-1}) = \frac{1}{\det(R^t)}$. □

Using Lemmas 1 and 2, we finally derive the regret bound for the regularized cost-model estimator in the following theorem.

Theorem 1. *If we consider the error as a bounded function such that* $0 \leqslant \hat{\epsilon_t}^2 \leqslant E_{\max}$ *and* $\|\boldsymbol{\eta}^t\|_\infty \leqslant \delta$,

$$Reg_T \leqslant \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[n\ln\left(1 + \frac{\varepsilon\delta^2 T}{n}\right) - (n-1)\ln(\varepsilon)\right] \tag{11}$$

where $R^0 = \varepsilon I$.

Proof. Let us assume the squared error has an upper bound $E_{\max}$ for a given workload. Under this assumption, we get from Equations (8) and (9),

$$\begin{aligned}
Reg_T &\leqslant \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\sum_{t=1}^{T}\left(1 - \frac{\det(R^t)}{\det(R^{t-1})}\right)\\
&\leqslant \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 - \frac{E_{\max}}{\lambda}\sum_{t=1}^{T}\ln\left(\frac{\det(R^t)}{\det(R^{t-1})}\right)\\
&= \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\ln\left(\frac{\det(R^0)}{\det(R^T)}\right)\\
&= \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\ln(\varepsilon) - \ln(\det(R^T))\right]\\
&= \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\ln(\varepsilon) + \ln\left(\det\left(\frac{1}{\varepsilon}I + \sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right)\right)\right]\\
&= \lambda \|\boldsymbol{\zeta}^{\text{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\sum_{k=1}^{n}\ln(1 + \varepsilon\lambda_k) - (n-1)\ln(\varepsilon)\right].
\end{aligned}$$

Because

$$\det\left(\frac{1}{\varepsilon}I + \sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right) = \varepsilon^{-n}\det\left(I + \varepsilon\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right) \quad = \varepsilon^{-n}\prod_{k=1}^{n}(1 + \varepsilon\lambda_k)$$

where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of the matrix $\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T$. As the eigenvalues of $\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T$ are equal to the eigenvalues of its Gram matrix $G_{ij} = (\boldsymbol{\eta}^i)^T\boldsymbol{\eta}^j$, we can write

$$\sum_{k=1}^{n} \lambda_k = Trace(G) = \sum_{t=1}^{T} (\boldsymbol{\eta}^t)^T \boldsymbol{\eta}^t \leqslant \delta^2 T$$

where $\|\boldsymbol{\eta}^t\|_\infty \leqslant \delta$, that is, the maximum value of any component of $\boldsymbol{\eta}$ is bounded by δ. In the above inequality, the equality holds if and only if $\lambda_1 = \lambda_2 = \ldots = \lambda_n = \frac{\delta^2 T}{n}$. By applying this condition, we get the regret bound as

$$Reg_T \leqslant \lambda \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda} \left[n \ln \left(1 + \frac{\varepsilon \delta^2 T}{n} \right) - (n-1) \ln(\varepsilon) \right]. \qquad \square$$

This theorem shows that our estimation of the cost model using Algorithm 4 is always upper bounded by a constant value depending on the optimal solution added with a term that increases with time logarithmically. This shows that the regret, which is the cumulative deviation of the cost model computed by Algorithm 4 with respect to the optimal one, increases very slowly with time. That means the error of estimation in each and every time step is considerably small.

6 Case Study: Index Tuning

In this section we present *COREIL* (for *Cost-model Oblivious REInforcement Learning algorithm*) and its regularized version, *rCOREIL*. COREIL and rCOREIL instantiate Algorithm 3 taking as cost-model estimators Algorithms 2 and 4 respectively. Both of them tune the configurations differing in their secondary indexes and handle the configuration changes corresponding to the creation and deletion of indexes. COREIL uses reinforcement learning approach to solve the index tuning problem *on-the fly*. It projects index tuning as an MDP and applies Algorithm 3 to solve it. On the other hand, rCOREIL uses the regularized cost-model estimator, described in Section 5.1; rCOREIL's regularized estimator affords it to leverage the fact that if we serve the learning algorithm with a better cost-model to evaluate its policy better, it will perform better. In this section, we also define the feature mappings $\boldsymbol{\phi}$ and $\boldsymbol{\eta}$ for both COREIL and rCOREIL. They are used to approximate the cost-to-go function V and the *cost* function respectively. At the end of this section we prove tighter performance bounds for Algorithm 4 in case of index tuning. We also derive optimal values of the parameters λ and ε for a given workload.

6.1 Reducing the Search Space

Let I be the set of indexes that can be created. Each configuration $s \in S$ is an element of the power set 2^I. For example, 7 attributes in a schema of R yield a total of 13699 indexes and a total of 2^{13699} possible configurations. Such a large search space invalidates a naive brute-force search for the optimal policy.

For any query $\hat{q}$, let $r(\hat{q})$ be a function that returns a set of recommended indexes. This function may be already provided by the database system (e.g., as with IBM DB2), or it can be implemented externally [1]. Let $d(\hat{q}) \subseteq I$ be the set of indexes being modified (update, insertion or deletion) by $\hat{q}$. We can define the reduced search space as

$$S_{s,\hat{q}} = \{s' \in S \mid (s - d(\hat{q})) \subseteq s' \subseteq (s \cup r(\hat{q}))\}. \tag{12}$$

Deleting indexes in $d(\hat{q})$ will reduce the index maintenance overhead and creating indexes in $r(q)$ will reduce the query execution cost. Note that the definition of $S_{s,\hat{q}}$ here is a subset of the one defined in Section 4.2 which deals with the general configurations.

Note that for tree-structured indexes (e.g., B$^+$-tree), we could further consider the *prefix closure* of indexes for optimization. For any configuration $s \in 2^I$, define the prefix closure of s as

$$\langle s \rangle = \{i \in I \mid i \text{ is a prefix of an index } j \text{ for some } j \in s\}. \tag{13}$$

Thus in Equation (12), we use $\langle r(\hat{q}) \rangle$ to replace $r(\hat{q})$ for better approximation. The intuition is that in case of $i \notin s$ but $i \subseteq \langle s \rangle$ we can leverage the prefix index to answer the query.

6.2 Defining the Feature Mapping $\boldsymbol{\phi}$

Let V be the cost-to-go function following a policy. As mentioned earlier, Algorithm 3 relies on a proper feature mapping $\boldsymbol{\phi}$ that approximates the cost-to-go function as $V(s) \approx \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for some vector $\boldsymbol{\theta}$. The challenge lies in how to define $\boldsymbol{\phi}$ under the scenario of index tuning. Both in COREIL and rCOREIL, we define it as

$$\phi_{s'}(s) := \begin{cases} 1, & \text{if } s' \subseteq s \\ -1, & \text{otherwise} \end{cases}$$

for each $s, s' \in S$. Let $\boldsymbol{\phi} = (\phi_{s'})_{s' \in S}$. Note that $\phi_\emptyset$ is an intercept term since $\phi_\emptyset(s) = 1$ for all $s \in S$. The following proposition shows the effectiveness of $\boldsymbol{\phi}$ for capturing the values of the cost-to-go function V.

Proposition 3. *There exists a unique $\boldsymbol{\theta} = (\theta_{s'})_{s' \in S}$ which approximates the value function as*

$$V(s) = \sum_{s' \in S} \theta_{s'} \phi_{s'}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s). \tag{14}$$

Proof. Suppose $S = \{s^1, s^2, \ldots, s^{|S|}\}$. Note that we use superscripts to denote the ordering of elements in S.

Let $\boldsymbol{V} = (V(s))^T_{s \in S}$ and M be a $|S| \times |S|$ matrix such that

$$M_{i,j} = \phi_{s^j}(s^i).$$

Let $\boldsymbol{\theta}$ be a $|S|$-dimension column vector such that $M\boldsymbol{\theta} = \boldsymbol{V}$. If M is invertible then $\boldsymbol{\theta} = M^{-1}V$ and thus Equation (14) holds.

We now show that M is invertible. Let $\boldsymbol{\psi}$ be a $|S| \times |S|$ matrix such that

$$\boldsymbol{\psi}_{i,j} = M_{i,j} + 1.$$

We claim that $\boldsymbol{\psi}$ is invertible and its inverse is the matrix $\boldsymbol{\tau}$ such that

$$\boldsymbol{\tau}_{i,j} = (-1)^{|s^i|-|s^j|}\boldsymbol{\psi}_{i,j}.$$

To see this, consider

$$\begin{aligned}(\boldsymbol{\tau}\boldsymbol{\psi})_{i,j} &= \sum_{1\leqslant k\leqslant |S|} (-1)^{|s^i|-|s^k|}\boldsymbol{\psi}_{i,k}\boldsymbol{\psi}_{k,j} \\ &= \sum_{s_j\subseteq s_k\subseteq s_i} (-1)^{|s^i|-|s^k|}.\end{aligned}$$

Therefore $(\boldsymbol{\tau}\boldsymbol{\psi})_{i,j} = 1$ if and only if $i = j$. By the Sherman-Morrison formula, M is also invertible. □

However, for any configuration s, $\boldsymbol{\theta}(s)$ is a $|2^I|$-dimensional vector. In order to reduce the dimensionality, the cost-to-go function can be approximated by $V(s) \approx \sum_{s'\in S, |s'|\leqslant N} \theta_{s'}\phi_{s'}(s)$ for some integer N. Here we assume that the collaborative benefit among indexes could be negligible if the number of indexes exceeds N. In particular when $N = 1$, we have

$$V(s) \approx \theta_0 + \sum_{i\in I} \theta_i\phi_i(s). \tag{15}$$

where we ignore all the collaborative benefits among indexes in a configuration. This is reasonable since any index in a database management system is often of individual contribution for answering queries [32]. Therefore, we derive $\boldsymbol{\phi}$ from Equation (15) as $\boldsymbol{\phi}(s) = (1, (\phi_i(s))_{i\in I}^T)^T$. By using this feature mapping $\boldsymbol{\phi}$, both COREIL and rCOREIL approximate the cost-to-go function $V(s) \approx \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$ for some vector $\boldsymbol{\theta}$.

6.3 Defining the Feature Mapping $\boldsymbol{\eta}$

A good feature mapping for approximating functions δ and *cost* must take into account both the benefit from the current configuration and the maintenance overhead of the configuration.

To capture the difference between the index set recommended by the database system and that of the current configuration, we define a function $\boldsymbol{\beta}(s, \hat{q}) = (1, (\boldsymbol{\beta}_i(s, \hat{q}))_{i\in I}^T)^T$, where

$$\boldsymbol{\beta}_i(s, \hat{q}) := \begin{cases} 0, & i \notin r(\hat{q}) \\ 1, & i \in r(\hat{q}) \text{ and } i \in s \\ -1, & i \in r(\hat{q}) \text{ and } i \notin s. \end{cases}$$

If the execution of query $\hat{q}$ cannot benefit from index i then $\boldsymbol{\beta}_i(s, \hat{q})$ always equals zero; otherwise, $\boldsymbol{\beta}_i(s, \hat{q})$ equals 1 or -1 depending on whether s contains i or not. For tree-structured indexes, we could further consider the prefix closure of indexes as defined in Equation (13) for optimization.

On the other hand, to capture whether a query (update, insertion or deletion) modifies any index in the current configuration, we define a function $\boldsymbol{\alpha}(s, \hat{q}) = (\alpha_i(s, \hat{q}))_{i \in I}$ where

$$\alpha_i(s, \hat{q}) = \begin{cases} 1, & \text{if } i \in s \text{ and } \hat{q} \text{ modify } i \\ 0, & \text{otherwise.} \end{cases}$$

Note that if $\hat{q}$ is a selection query, $\boldsymbol{\alpha}$ trivially returns $\mathbf{0}$.

By combining $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$, we get the feature mapping $\boldsymbol{\eta} = (\boldsymbol{\beta}^T, \boldsymbol{\alpha}^T)^T$ used in both of the algorithms. It can be used to approximate the functions δ and *cost* as described in Section 4.3.

6.4 Performance Bounds for Regularized COREIL

rCOREIL applies Algorithm 4 for cost-model estimation, while COREIL uses RLSE for this. If we follow Algorithm 3, on line 13 rCOREIL calls the regularized cost-model estimator with arguments $\hat{\epsilon}^t, R^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t$ instead of RLSE. Following Theorem 1 and the construction of the feature map in Section 6.3, Proposition 4 gives a tighter regret bound for the cost-model estimation of rCOREIL.

Proposition 4. *If we consider the error as a bounded function such that* $0 \leqslant \hat{\epsilon_t}^2 \leqslant E_{\max}$*:*

$$Reg_T^{rCOREIL} \leqslant \lambda \|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda} \left[2n\text{ln}T - n\text{ln}n\right] \tag{16}$$

and the optimal value for ε *is given by:*

$$\varepsilon^* = \frac{n^2 - n}{T}.$$

Proof. From Section 6.3, $\|\boldsymbol{\eta}^t\|_\infty \leqslant 1$. Equation (11) transforms into

$$Reg_T^{rCOREIL} \leqslant \lambda \|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda} \left[n\text{ln}\left(1 + \frac{\epsilon T}{n}\right) - (n-1)\text{ln}(\varepsilon)\right].$$

Now, we determine the optimal value of ε by minimizing the RHS of above inequality as this will impose tighter limit on the bound. Thus,

$$\left[\frac{\partial(\text{RHS})}{\partial \varepsilon}\right]_{\varepsilon^*=0} = 0.$$

By solving this, we get $\varepsilon^* = \frac{n^2-n}{T}$. Substituting this value in the previous inequality gives us the regret bound for regularized COREIL algorithm as

$$Reg_T^{rCOREIL} \leqslant \lambda \|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda} \left[2n\text{ln}(T) - n\text{ln}(n)\right]. \qquad \square$$

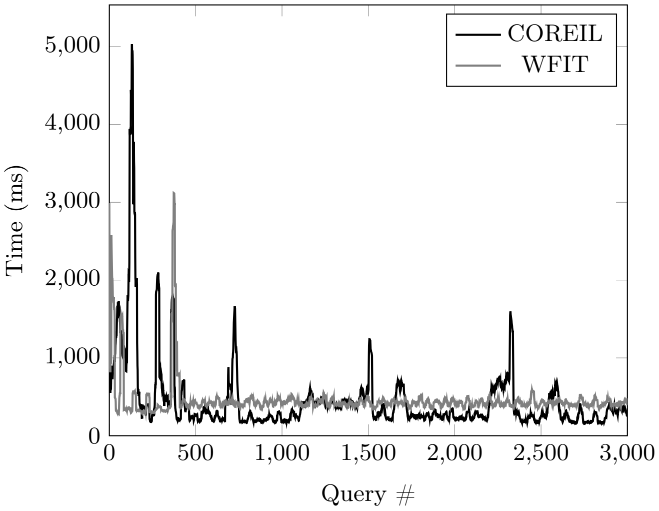

Fig. 1. Evolution of the efficiency (total time per query) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20)

Similarly, we can also find out the optimal value of λ that will make the upper bound tightest.

Corollary 1. *If the value of optimal solution $\boldsymbol{\zeta}^{OPT}$ can be predicted beforehand, the optimal value of λ is given by*

$$\lambda^{*} = \frac{E_{\max}}{\|\boldsymbol{\zeta}^{OPT}\|^{2}} \left[2n\ln(T) - n\ln(n)\right]$$

where the stopping time T is given.

Proof. As an optimal λ will minimize the RHS of Equation (16), we get it by setting the partial derivative of the RHS with respect to λ as zero. This simply gives us, $\lambda^{*} = \frac{E_{\max}}{\|\boldsymbol{\zeta}^{OPT}\|^{2}} \left[2n\ln(T) - n\ln(n)\right]$. □

Substituting the optimal value of λ in Equation (16) for a given T and $\boldsymbol{\zeta}^{OPT}$, we get

$$Reg_{T}^{rCOREIL} \leqslant \|\boldsymbol{\zeta}^{OPT}\|^{2} + E_{\max}\left[2n\ln(T) - n\ln(n)\right].$$

For large n and comparatively smaller T, $[2n\ln(T) - n\ln(n)]$ is a negative number that makes the plausible error in cost-model estimation much smaller than even the magnitude of the optimal $\boldsymbol{\zeta}$ vector. This shows the guarantee on the quality of the cost-model estimated by rCOREIL once the parameters are properly set.

7 Performance Evaluation

In this section, we present an empirical evaluation of COREIL and rCOREIL through two sets of experiments. In the first set of experiments, we implement

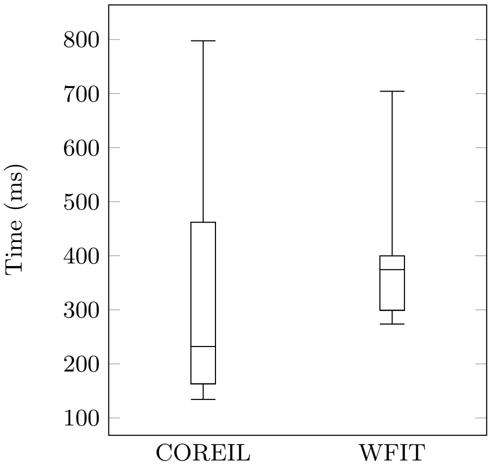

Fig. 2. Box chart of the efficiency (total time per query) of the two systems. We show in both cases the 9th and 91th percentiles (whiskers), first and third quartiles (box) and median (horizontal rule).

a prototype of COREIL in Java. We compare its performance with that of the state-of-the-art WFIT algorithm [40] (briefly described in Section 7.2). In the results, we can see that COREIL shows competitive performance with WFIT but has higher variance. This validates the efficiency of the reinforcement learning approach to solve the index tuning problem *on the fly*. This shows that, even without any assumption of a pre-determined cost model, it is possible to perform at the level of the state-of-the-art.

In the second set of experiments, we evaluate the performance of rCOREIL with respect to COREIL. The results show enhancements in performance by rCOREIL as reasoned in Section 5. This validates the claim in Section 5 that the higher variance of COREIL is due to suboptimal use of the RLSE algorithm. It also establishes the fact that if we serve the learning algorithm with an enhanced estimation of cost-model, it improves the performance substantially. In these experiments, we also check the sensitivity of rCOREIL with respect to the parameter λ and cross-validate the optimal value for the given workload.

7.1 Dataset and Workload

The dataset and workload is conforming to the TPC-C specification [31] and generated by the OLTP-Bench tool [15]. The 5 types of transactions in TPC-C are distributed as NewOrder 45%, Payment 43%, OrderStatus 4%, Delivery 4% and StockLevel 4%. Each of these transactions are associated with $3 \sim 5$ SQL statements (query/update). The scale factor used throughout the experiments is 2. We do not leverage any repetition or periodicity of the workload in our approach; still for robustness there may be up to 10% of repetition of queries.

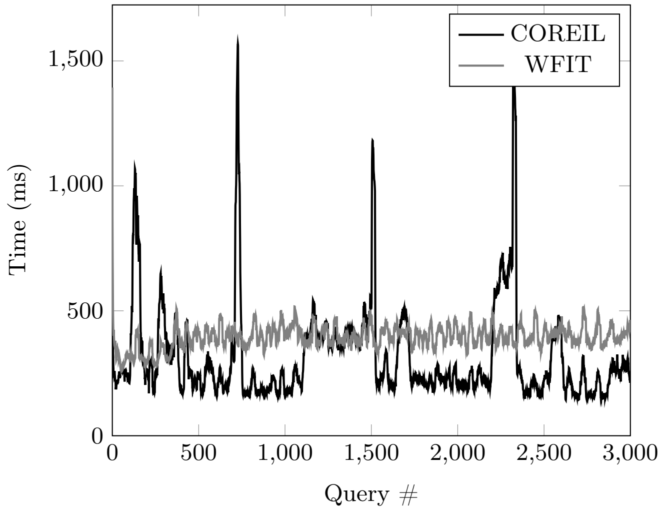

Fig. 3. Evolution of the overhead (time of the optimization itself) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20)

Note that [40] additionally uses the dataset NREF in its experiments. However, this dataset and workload are not publicly available.

7.2 WFIT: Brief Description

WFIT is proposed in [40] as a method of semi-automatic index tuning. This algorithm keeps the database administrator "in the loop" by generating recommendations. These recommendations are generated through a feedback loop originating from the administrator's preferences. This process is based on the Work Function Algorithm [8]. In order to determine the change of configuration, WFIT considers all the queries observed in the past. Then it solves a deterministic problem of minimizing the total processing cost. However, while doing so, it assumes the existence of a pre-determined cost model served by the database system or administrator. Due to use of a pre-defined cost model for all the datasets and workloads it faces the problems discussed in the Introduction. Results documented in the following sections will show the importance of a reinforcement learning approach to make the process generic and cost-model oblivious.

7.3 COREIL: Experiments and Results

Experimental Set-up We conduct all the experiments on a server running IBM DB2 10.5. The server is equipped with Intel i7-2600 Quad-Core @ 3.40 GHz and 4 GB RAM. We measure wall-clock times for execution of all components.

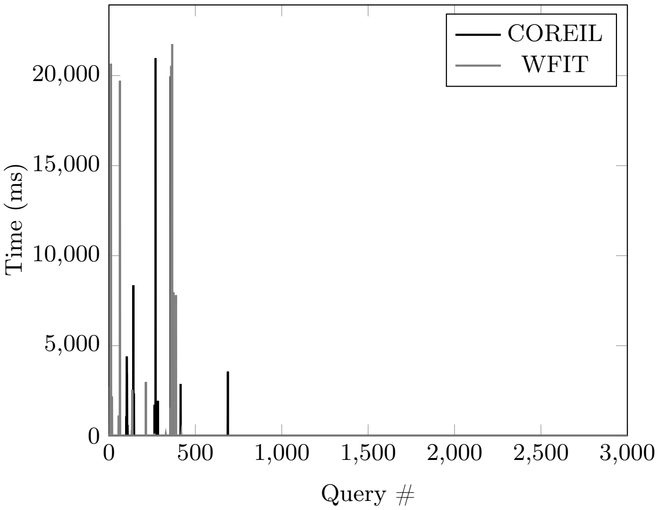

Fig. 4. Evolution of the time taken by configuration change (index creation and destruction) of the two systems from the beginning of the workload; no configuration change happens past query #700. All values except the vertical lines shown in the figure are zero.

Specially, for execution of workload queries or index creating/dropping, we measure the response time of processing corresponding SQL statement in DB2. Additionally, WFIT uses the what-if optimizer of DB2 to evaluate the cost. In this setup, each query is executed only once and all the queries were generated from one execution history.

Efficiency Figure 1 shows the total cost of processing TPC-C queries for online index tuning of COREIL and WFIT. Total cost consists of the overhead of corresponding tuning algorithm, cost of configuration change and that of query execution. Results show that, after convergence, COREIL has lower processing cost most of the time. But COREIL converges slower than WFIT, which is expected since it does not rely on the what-if optimizer to guide the index creations. [4] With respect to the whole execution set, the average processing cost of COREIL (451 ms) is competitive to that of WFIT (452 ms). However, if we calculate the average processing cost of the 500^{th} query forwards, the average performance of COREIL (357 ms) outperforms that of WFIT (423 ms). To obtain further insight from these data, we study the distribution of the processing time per query, as shown in Figure 2. As can be seen, although COREIL exhibits larger variance in the processing cost, its median is significantly lower that that

[4] By convergence we mean the first stable patch in Figure 1 after the series of high spikes, around the 500^{th} query. The convergence point is qualitatively chosen by observing characteristics of the curve.

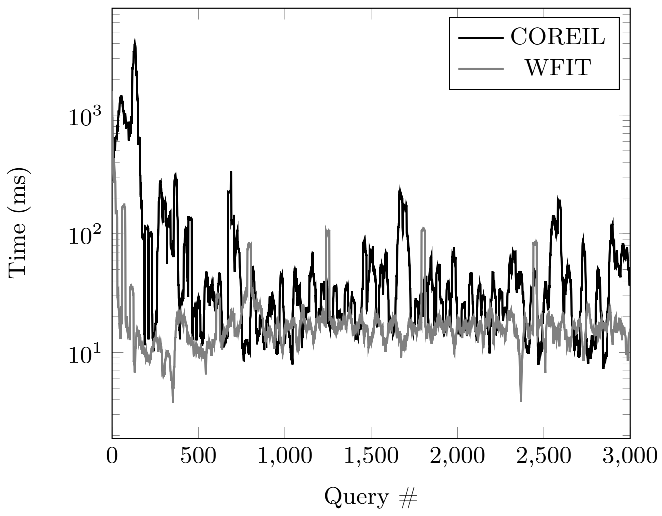

Fig. 5. Evolution of the effectiveness (query execution time in the DBMS alone) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

of WFIT. All these results confirms that COREIL has better efficiency than WFIT under a long term execution.

Figures 3 and 4 show analysis of the overhead of corresponding tuning algorithm and cost of configuration change respectively. By comparing Figure 1 with Figure 3, we can see that the overhead of the tuning algorithm dominates the total cost and the overhead of COREIL is significantly lower than that of WFIT. In addition, WFIT tends to make costlier configuration changes than COREIL, which is reflected in a higher time for configuration change. This would be discussed further in the micro-analysis. Note that both methods converge rather quickly and no configuration change happens beyond the 700th query.

A possible reason for the comparatively smaller overhead of COREIL with respect to WFIT, in addition to not relying on a possibly costly what-if optimizer, is the MDP structure. In MDPs, all the history of the system is assumed to be summarized in the present state and the cost-function. Thus, COREIL has to do less book-keeping than WFIT.

Effectiveness To verify the effectiveness of indexes created by the tuning algorithms, we extract the cost of query execution from the total cost. Figure 5 (note the logarithmic y-axis) indicates that the set of indexes created by COREIL shows competitive effectiveness with that created by WFIT, though WFIT is more effective in general and exhibits less variance after convergence. Again, this is to be expected since COREIL does not have access to any cost model for the queries. As previously noted, the total running time is lower for COREIL than

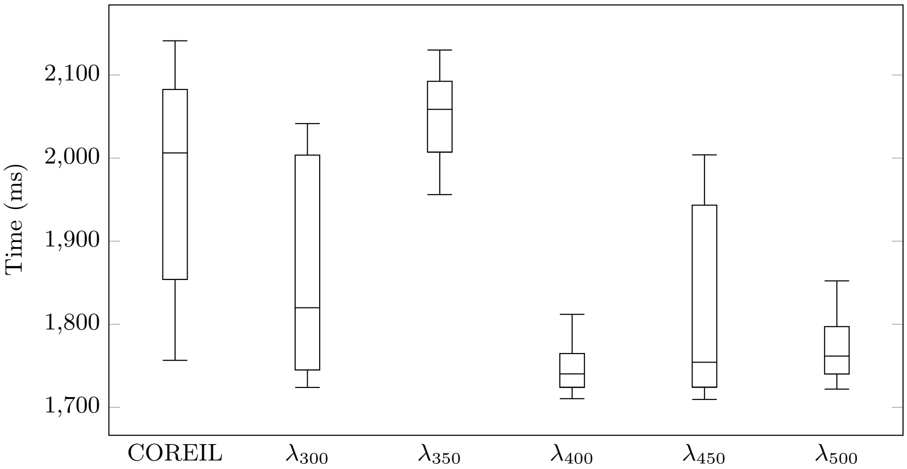

Fig. 6. Box chart of the efficiency (total time per query) of COREIL and its improved version with different values of λ. We show in both cases the 9th and 91st percentile (whiskers), first and third quartiles (box) and median (horizontal rule).

WFIT, as overhead rather than query execution dominates running time for both systems.

We have also performed a micro-analysis to check whether the indexes created by the algorithms are reasonable. We observe that WFIT creates more indexes with longer compound attributes, whereas COREIL is more parsimonious in creating indexes. For instance, WFIT creates a 14-attribute index as shown below.

```
[S_W_ID, S_I_ID, S_DIST_10, S_DIST_09, S_DIST_08, S_DIST_07,
 S_DIST_06, S_DIST_05, S_DIST_04, S_DIST_03, S_DIST_02,
 S_DIST_01, S_DATA, S_QUANTITY]
```

The reason of WFIT creating such a complex index is probably due to multiple queries with the following pattern.

```
SELECT S_QUANTITY, S_DATA, S_DIST_01, S_DIST_02, S_DIST_03,
       S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
       S_DIST_09, S_DIST_10
FROM STOCK
WHERE S_I_ID = 69082 AND S_W_ID = 1;
```

In contrast, COREIL tends to create shorter compound-attribute indexes. For example, COREIL created an index `[S_I_ID, S_W_ID]` which is definitely beneficial to answer the query above and is competitive in performance compared with the one created by WFIT.

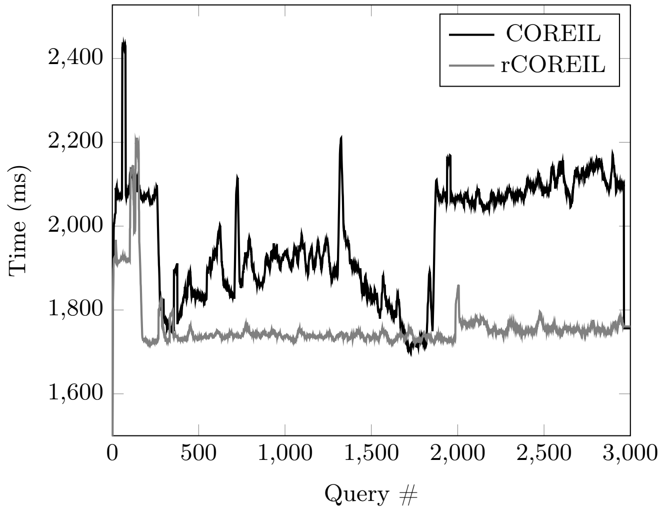

Fig. 7. Evolution of the efficiency (total time per query) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20)

7.4 rCOREIL: Experiments and Results

Experimental Set-up We run COREIL and rCOREIL, with a set of λ values $300, 350, 400, 450$, and 500. The previous set of experiments have already established competitive performance of COREIL with WFIT. In this set we evaluate the basic idea of rCOREIL: providing regularized estimation of cost-model enhances the performance of COREIL and also stabilizes it. We conduct all the experiments on a server running IBM DB2 10.5 with scale factor and time measure, mentioned in the previous set of experiments. But here the server is installed on a 64 bit Windows virtual box with dual-core 2-GB hard disk. It operates in an Ubuntu machine with Intel i7-2600 Quad-Core @ 3.40 GHz and 4 GB RAM. This eventually makes both version of algorithms slower in comparison to the previous physical machine installation.

Efficiency As the offline optimal outcome for this workload is unavailable beforehand, we set an expected range of λ as $[300, 600]$ depending on the other parameters like the number of queries and the size of state space. Figure 6 shows efficiency of COREIL and rCOREIL with different values of λ. As promised by Algorithm 4, variations of rCOREIL are always showing lesser median and variance of total cost. We can also observe from the boxplot, the efficiency is maximum as well as the variance is minimum for $\lambda = 400$. As efficiency is the final measure that controls runtime performance of the algorithm, we have considered this as optimal value of λ for further analysis. This process is analogous to cross-validation of parameter λ, where the proved bounds help us to set a range

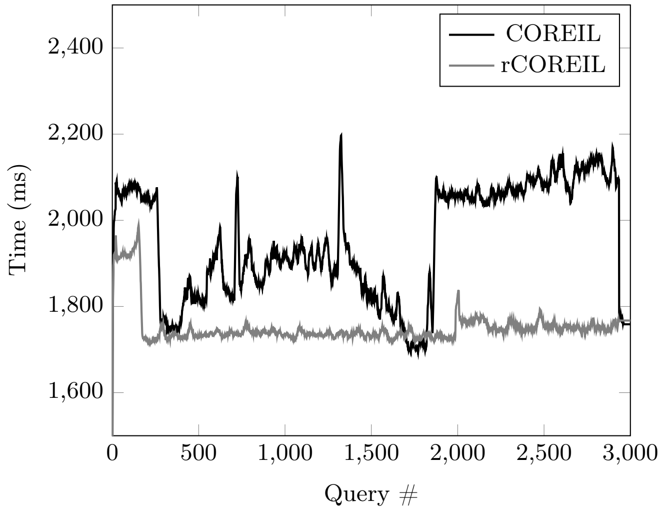

Fig. 8. Evolution of the overhead (time of the optimization itself) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20)

of values for searching it instead of going through an arbitrary large range of values. Though here we are validating depending upon the result obtained from the whole run of 3,000 queries in the workload, the optimal λ would typically be set, in a realistic scenario, after running first 500 queries of the workload with different parameter values and then choosing the optimal one. Figure 7 shows that rCOREIL with $\lambda = 400$ outperforms COREIL. With respect to the whole execution set, the average processing cost of rCOREIL is 1758 ms which is significantly less than that of COREIL (1975 ms). Also the standard deviation of rCOREIL is 90ms which is half of that of COREIL, 180ms. This enhanced performance and low variance establishes the claim that if we serve the learning algorithm with a better estimation of cost-model it will improve.

Figures 8 and 9 show analysis of the overhead of corresponding tuning algorithms and cost of configuration change respectively. In this set of experiments also, we can see that the overhead of the tuning algorithms dominates their total cost. Here, the overhead of rCOREIL for each query is on an average 207ms lower than that of COREIL. This is more than 10% improvement over the average overhead of COREIL. In addition, rCOREIL (mean: 644ms) also makes cheaper configuration changes than COREIL (mean: 858ms). rCOREIL also converges faster than COREIL as the last configuration update made by rCOREIL occurs at the 335^{th}query but the last two updates for COREIL occur at the 358^{th} and 1940^{th} queries respectively. If we look closely, the 358^{th} and 1940^{th} queries in this particular experiment are:

```
SELECT COUNT(DISTINCT (S_I_ID)) AS STOCK_COUNT
```

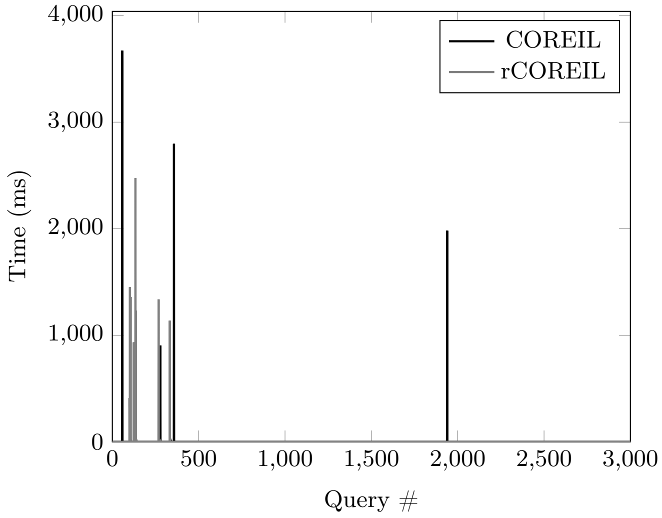

Fig. 9. Evolution of the time taken by configuration change (index creation and destruction) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload; no configuration change happens past query #2000. All values except the vertical lines shown in the figure are zero.

```
FROM ORDER_LINE, STOCK
WHERE OL_W_ID = 2 AND OL_D_ID = 10 AND OL_O_ID < 3509
  AND OL_O_ID >= 3509 - 20 AND S_W_ID = 2
  AND S_I_ID = OL_I_ID AND S_QUANTITY < 20;
```

and

```
SELECT COUNT(DISTINCT (S_I_ID)) AS STOCK_COUNT
FROM ORDER_LINE, STOCK
WHERE OL_W_ID = 1 AND OL_D_ID = 8 AND OL_O_ID < 3438
      AND OL_O_ID >= 3438 - 20 AND S_W_ID = 1
      AND S_I_ID = OL_I_ID AND S_QUANTITY < 11;
```

In reaction to this, COREIL creates indexes `[ORDER_LINE.OL_D_ID,ORDER_LINE.OL_W_ID]` and `[STOCK.S_W_ID, STOCK.S_QUANTITY]` respectively. It turns out that such indexes are not of much use for most other queries (only 6 out of 3000 queries benefit of one of these indexes). COREIL makes configuration updates to tune the indexes for such queries, while the regularized cost model of rCOREIL does not make configuration updates due to rare and complex events, because it regularizes any big change due to such an outlier. Instead, rCOREIL has a slightly higher the overhead to find out the optimal indexes. For example, in the window consisting of 10 queries after the 359^{th} query average overhead of rCOREIL increases from 1724ms to 1748ms.

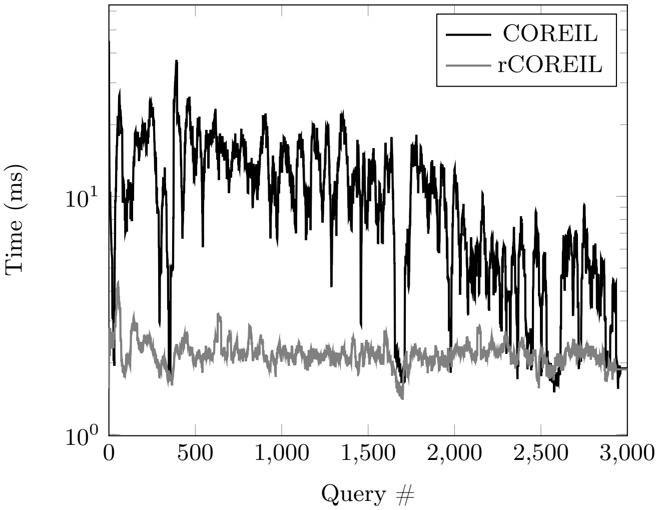

Fig. 10. Evolution of the effectiveness (query execution time in the DBMS alone) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

Effectiveness Like Section 7.3, here also we extract the cost of query execution to verify the effectiveness of indexes created by the tuning algorithms. Figure 10 indicates that the set of indexes created by rCOREIL are significantly more effective than those created by COREIL. We can see the average query execution time of rCOREIL is less than that of COREIL almost by a factor of 10.

At a micro-analysis level, we observe rCOREIL creates only one index with two combined attributes, all other indexes being single-attribute. On the other hand, COREIL creates only one index with a single attribute whereas all other indexes have two attributes. This observation shows that though COREIL creates parsimonious and efficient indexes, rCOREIL shows even better specificity and effectiveness in doing so.

7.5 Analysis of Cost Estimator

In order to examine the quality of the three cost estimators used by WFIT, COREIL, and rCOREIL to predict the actual cost of query executions or configuration updates, we observe the actual execution time, the estimated cost, and that returned by the what-if optimizer during every run of experiments for COREIL and rCOREIL, respectively. The scatter plot of Figure 11 shows that the what-if cost has significantly less correlation (0.013) with the actual execution time than COREIL (0.1539) Again, the scatter plot of Figure 12 shows the regularized cost estimated by rCOREIL has significantly higher positive correlation (0.1558) than that predicted by the what-if optimizer. This proves that the execution time estimated by COREIL and rCOREIL are significantly more reliable than the

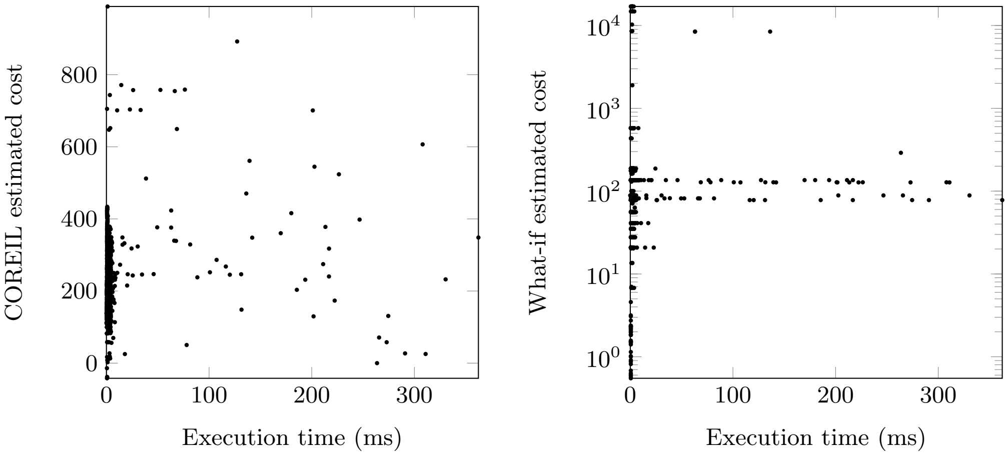

Fig. 11. Scatter plot of the estimated cost by COREIL and the what-if optimizer vs execution time. Left shows correlation between cost estimated by COREIL and actual execution time (in ms). Right shows (on a log y-axis) correlation between the cost estimated by the what-if optimizer and the actual execution time (in ms) in the same run.

ones estimated by what-if optimizer. It can also been observed that rCOREIL provides better estimations: visually, there are many more points at the middle of Figure 12 (left) with positive inclination.

Finally, Figure 13 shows that the regularized cost model estimator of rCOREIL gives a more stable estimation of the cost model than that of COREIL, as the cost model estimated by COREIL (averaged over 20 queries) shows higher variance and also sensitivity to changes in types of queries.

8 Conclusion

We have presented a cost-model oblivious solution to the problem of performance tuning. We first formalized the problem as a Markov decision process. Then we devised and presented a solution, which addresses both issues of the curse of dimensionality and of over-fitting. We instantiated the problem to the case of index tuning. For this case, we implemented and evaluated the COREIL and rCOREIL algorithms, with and without regularization, respectively. Experiments show competitive performance with respect to the state-of-the-art WFIT algorithm, despite our approach being cost-model oblivious. We also show that as our cost-model estimation becomes crisp and stable the performance of learner improves significantly. Beyond the material presented in this paper, we continue studying the universality and robustness of the COREIL and rCOREIL approaches.

Specially for rCOREIL, it is an interesting problem to determine the optimal regularization parameter on the go or to adapt it with the dynamics of workload. Though now this process causes us only a one-time up-front cost, following the

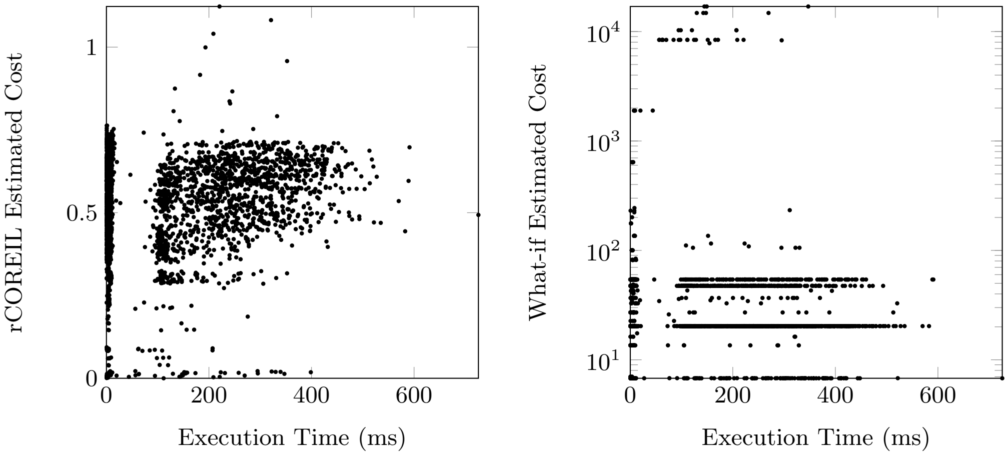

Fig. 12. Scatter plot of the estimated cost by rCOREIL and the what-if optimizer vs execution time. Left shows correlation between cost estimated by rCOREIL and actual execution time (in ms). Right shows (on a log y-axis) correlation between the cost estimated by the what-if optimizer and the actual execution time (in ms) in the same run.

flavour of our approach we would like to perform it online. One possible method is to run COREIL for the first 500 queries and to calculate the costs for different set of regularization parameter values simultaneously for that period. Following that, we can choose the parameter value that causes minimum average estimation of the cost function.

We are now running further empirical performance evaluation tests with other datasets such as TPC-E, TPC-H and dedicated benchmarks for online index tuning [38]. For completeness from an engineering perspective, we are considering concurrent access, which was ignored in the algorithm and experiments presented in this paper for the sake of simplicity. We are also going to look at the favourable case of predictable workload such as periodic transactions. Furthermore, we are extending the solution to other aspects of database configuration, including partitioning and replication. For each of these aspects, we need to devise specific and non-trivial heuristics that help curb the combinatorial explosion of the configuration space as well as specific intelligent initialization techniques.

Finally, note that a critical assumption in our approach is that queries arrive sequentially and that nothing is known ahead of time about the workload. Both assumptions do not held in a number of realistic settings: queries can be submitted concurrently to the database, and a workload may often be predictable (such as when it consists of similar transactions, repeated on different data items). We leave for further work the adaptation of rCOREIL to such settings.

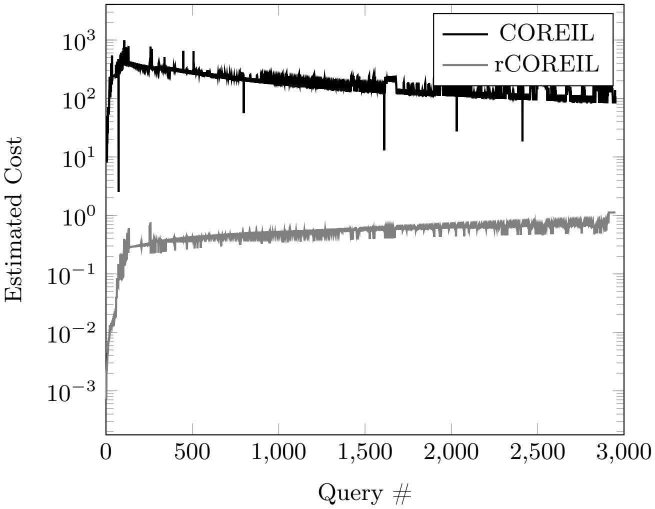

Fig. 13. Evolution of the estimated costs of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

Acknowledgement

We thank Prof. Haibo Chen for valuable feedback on this work. This research is funded by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme with the SP2 project of the Energy and Environmental Sustainability Solutions for Megacities – E2S2 programme.

References

1. Agrawal, S., Chaudhuri, S., Narasayya, V.R.: Automated selection of materialized views and indexes in sql databases. In: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00). pp. 496–505 (2000)
2. Agrawal, S., Narasayya, V., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04). pp. 359–370 (2004)
3. Alagiannis, I., Idreos, S., Ailamaki, A.: H2o: A hands-free adaptive store. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14) (2014)
4. Audibert, J.Y., Munos, R., Szepesvári, C.: Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. Theoretical Computer Science 410(19) (2009)
5. Azefack, S., Aouiche, K., Darmont, J.: Dynamic index selection in data warehouses. CoRR abs/0809.1965 (2008), `http://arxiv.org/abs/0809.1965`

6. Basu, D., Lin, Q., Chen, W., Vo, H.T., Yuan, Z., Senellart, P., Bressan, S.: Cost-model oblivious database tuning with reinforcement learning. In: Proc. DEXA. pp. 253–268. Valencia, Spain (Sep 2015)
7. Benedikt, M., Bohannon, P., Bruns, G.: Data cleaning for decision support. In: Proceedings of the 1st International VLDB Workshop on Clean Databases (CleanDB'06) (2006)
8. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
9. Bouchakri, R., Bellatreche, L., Hidouci, K.W.: Advances in Databases and Information Systems: 16th East European Conference, ADBIS 2012, Poznań, Poland, September 18-21, 2012. Proceedings, chap. Static and Incremental Selection of Multi-table Indexes for Very Large Join Queries, pp. 43–56. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), `http://dx.doi.org/10.1007/978-3-642-33074-2_4`
10. Bruno, N., Chaudhuri, S.: An online approach to physical design tuning. In: Proceedings of the 23th IEEE International Conference on Data Engineering (ICDE'07). pp. 826–835 (2007)
11. Bruno, N., Chaudhuri, S.: Constrained physical design tuning. Proceedings of the VLDB Endowment 1(1), 4–15 (2008)
12. Bruno, N., Chaudhuri, S.: Interactive physical design tuning. In: Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE'10). pp. 1161–1164 (2010)
13. Bruno, N., Nehme, R.V.: Configuration-parametric query optimization for physical design tuning. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08). pp. 941–952 (2008)
14. Chaudhuri, S., Narasayya, V.: Autoadmin: What-if index analysis utility. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98). pp. 367–378 (1998)
15. Difallah, D.E., Pavlo, A., Curino, C., Cudre-Mauroux, P.: Oltp-bench: An extensible testbed for benchmarking relational databases. Proceedings of the VLDB Endowment 7(4), 277–288 (2013)
16. Gouriten, G., Maniu, S., Senellart, P.: Scalable, generic, and adaptive systems for focused crawling. In: Proceedings of the 25th ACM Conference on Hypertext and Social Media (HT'14). pp. 35–45 (2014)
17. Hammer, M., Niamir, B.: A heuristic approach to attribute partitioning. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD'79). pp. 93–101 (1979)
18. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. The Journal of Machine Learning Research 4, 1107–1149 (2003)
19. Lai, T.L., Wei, C.Z.: Least squares estimates in stochastic regression models with applications to identification and control of dynamic systems. The Annals of Statistics pp. 154–166 (1982)
20. LeFevre, F., Sankaranarayanan, J., Hacigumus, H., Tatemura, J., Polyzotis, N., Carey, M.J.: Exploiting opportunistic physical design in large-scale data analytics. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14) (2014)
21. Li, L., Gruenwald, L.: Self-managing online partitioner for databases (smopd): A vertical database partitioning system with a fully automatic online approach. In: Proceedings of the 17th International Database Engineering and Applications Symposium (IDEAS'13). pp. 168–173 (2013)

22. Lightstone, S., Bhattacharjee, B.: Automated design of multidimensional clustering tables for relational databases. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04). pp. 1170–1181 (2004)
23. Lohman, G.M.: Is query optimization a "solved" problem? http://wp.sigmod.org/?p=1075 (2014)
24. Luhring, M., Sattler, K.U., Schmidt, K., Schallehn, E.: Autonomous management of soft indexes. In: Proceedings of the 2nd International Workshop on Self-Managing Data Bases (SMDB'07). pp. 450–458 (2007)
25. Malik, T., Wang, X., Dash, D., Chaudhary, A., Ailamaki, A., Burns, R.: Adaptive physical design for curated archives. In: Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM'09). pp. 148–166 (2009)
26. Nielsen, F., Bhatia, R.: Matrix information geometry. Springer (2013)
27. Papadomanolakis, S., Dash, D., Ailamaki, A.: Efficient use of the query optimizer for automated physical design. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07). pp. 1093–1104 (2007)
28. Papadomanolakis, S., Dash, D., Ailamaki, A.: Efficient use of the query optimizer for automated physical design. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07). pp. 1093–1104 (2007)
29. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley-Interscience (2007)
30. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming, vol. 414. John Wiley & Sons (2009)
31. Raab, F.: TPC-C - the standard benchmark for online transaction processing (OLTP). In: Gray, J. (ed.) The Benchmark Handbook. Morgan Kaufmann (1993)
32. Ramakrishnan, R., Gehrke, J., Gehrke, J.: Database management systems, vol. 3. McGraw-Hill New York (2003)
33. Rao, J., Zhang, C., Megiddo, N., Lohman, G.: Automating physical database design in a parallel database. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD'02). pp. 558–569 (2002)
34. Rasin, A., Zdonik, S.: An automatic physical design tool for clustered column-stores. In: Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13). pp. 203–214 (2013)
35. Rieser, V., Robinson, D.T., Murray-Rust, D., Rounsevell, M.: A comparison of genetic algorithms and reinforcement learning for optimising sustainable forest management. GeoComputation (2011)
36. Rockafellar, R.T.: Convex analysis. Princeton university press (2015)
37. Rösch, P., Dannecker, L., Färber, F., Hackenbroich, G.: A storage advisor for hybrid-store databases. Proceedings of the VLDB Endowment 5(12), 1748–1758 (2012)
38. Schnaitter, K., Polyzotis, N.: A benchmark for online index selection. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 1701–1708 (March 2009)
39. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: On-line index selection for shifting workloads. In: Proceedings of the 2nd International Workshop on Self-Managing Data Bases (SMDB'07). pp. 459–468 (2007)
40. Schnaitter, K., Polyzotis, N.: Semi-automatic index tuning: Keeping dbas in the loop. Proceedings of the VLDB Endowment 5(5), 478–489 (2012)
41. Stillger, M., Lohman, G.M., Markl, V., Kandil, M.: LEO – DB2's LEarning Optimizer. In: VLDB (2001)
42. Sutton, R.S., Barto, A.G.: Reinforcement Learning. MIT Press (1998)

43. Warmuth, M.K., Jagota, A.K.: Continuous and discrete-time nonlinear gradient descent: Relative loss bounds and convergence. In: Electronic proceedings of the 5th International Symposium on Artificial Intelligence and Mathematics. Citeseer (1997)
44. White, D.J.: Markov decision processes. John Wiley & Sons New York, NY (1993)
45. Young, P.: Recursive least squares estimation. In: Recursive Estimation and Time-Series Analysis, pp. 29–46. Springer Berlin Heidelberg (2011)
46. Zilio, D.C., Zuzarte, C., Lightstone, S., Ma, W., Lohman, G.M., Cochrane, R., Pirahesh, H., Colby, L.S., Gryz, J., Alton, E., Liang, D., Valentin, G.: Recommending materialized views and indexes with IBM DB2 design advisor. In: Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04). pp. 180–188 (2004)