Esc: An Early-Stopping Checker for Budget-aware Index Tuning

Xiaoying Wang
Simon Fraser Universtiy
Burnaby, Canada
xiaoying_wang@sfu.ca

Wentao Wu
Microsoft Research
Redmond, USA
wentao.wu@microsoft.com

Vivek Narasayya
Microsoft Research
Redmond, USA
viveknar@microsoft.com

Surajit Chaudhuri
Microsoft Research
Redmond, USA
surajitc@microsoft.com

ABSTRACT

Index tuning is a time-consuming process. One major performance bottleneck in existing index tuning systems is the large amount of "what-if" query optimizer calls that estimate the cost of a given pair of query and index configuration without materializing the indexes. There has been recent work on *budget-aware* index tuning that limits the amount of what-if calls allowed in index tuning. Existing budget-aware index tuning algorithms, however, typically make fast progress early on in terms of the best configuration found but slow down when more and more what-if calls are allocated. This observation of "diminishing return" on index quality leads us to introduce *early stopping* for budget-aware index tuning, where user specifies a threshold on the tolerable loss of index quality and we stop index tuning if the projected loss with the remaining budget is below the threshold. We further propose *Esc*, a low-overhead early-stopping checker that realizes this new functionality. Experimental evaluation on top of both industrial benchmarks and real customer workloads demonstrates that *Esc* can significantly reduce the number of what-if calls made during budget-aware index tuning while incurring little or zero improvement loss and little extra computational overhead compared to the overall index tuning time.

PVLDB Reference Format:
Xiaoying Wang, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. Esc: An Early-Stopping Checker for Budget-aware Index Tuning. PVLDB, 18(5): 1278 - 1290, 2025.
doi:10.14778/3718057.3718059

1 INTRODUCTION

Index tuning is a time-consuming process that may take hours to finish for large and complex workloads. Existing index tuners typically adopt a cost-based tuning architecture [7, 41], as illustrated in Figure 1. It consists of three main components: (1) *workload parsing and analysis*, which parses each query in the workload and extracts *indexable columns*, e.g., columns that appear in selection and join predicates; (2) *candidate index generation*, which puts together the extracted indexable columns to generate a set of indexes that can potentially reduce the execution cost of the input workload; and (3) *configuration enumeration*, which looks for a subset (a.k.a., configuration) from the candidate indexes that meets the input constraints (e.g., maximum configuration size or amount of storage to be taken by the indexes) while minimizing the input workload cost. To evaluate the cost of a given query and configuration pair, index tuners

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 18, No. 5 ISSN 2150-8097.
doi:10.14778/3718057.3718059

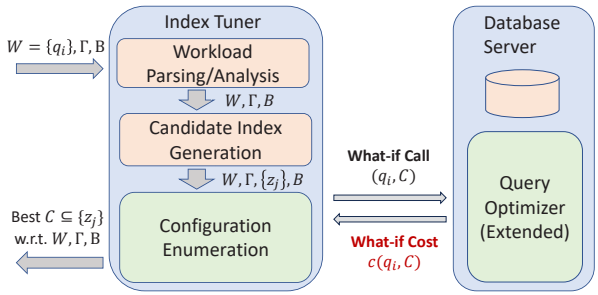
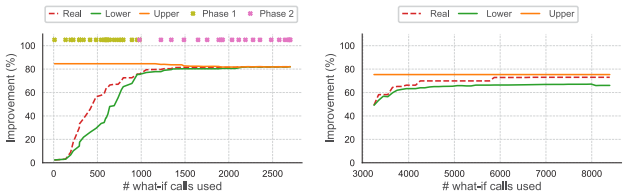


Figure 1: The architecture of cost-based index tuning using what-if optimizer calls, where W is the input workload and $q_i \in W$ is a single query, Γ is a set of tuning constraints, $\{z_j\}$ is the set of candidate indexes generated for W, and $C \subseteq \{z_j\}$ represents an index configuration during enumeration.

rely on the so-called "what-if" utility [8]. It is an extended API of the query optimizer that can estimate the cost by viewing the indexes contained by the configuration as "hypothetical indexes" instead of materializing them in a storage system, which would be much more costly. Nevertheless, what-if optimizer calls are not free—they are at least as expensive as a regular query optimizer call. As a result, they become the major bottleneck when tuning large and/or complex workloads [38].

To address this challenge, some technologies have been developed, such as cost derivation [7], caching/reusing what-if calls [26] that requires code changes to the query optimizer beyond the what-if API, or ML-based cost approximation [39]. Recent research has proposed *budget-aware* index tuning, which constrains the number of what-if calls allowed during configuration enumeration [51]. Here, the main challenge shifts from reducing the number of what-if calls in classic index tuning to prioritizing what-if calls w.r.t. the importance of query-configuration pairs in budget-aware index tuning. This problem is termed as *budget allocation*, and there has been recent work on optimizing budget allocation in a *dynamic* manner that skips *inessential* what-if calls at index tuning runtime by utilizing lower and upper bounds of what-if costs [43].

In practice, we have observed the following "diminishing return" behavior of existing budget-aware index tuning algorithms: they typically make fast progress at the beginning in terms of the best index configuration found, but their progress slows down as more budget on what-if calls is allocated. To put our discussion in context, Figure 2 presents examples of the *index tuning curve* (ITC) when using two state-of-the-art budget-aware index tuning algorithms (see Section 2), namely, *two-phase greedy search* and *Monte Carlo tree search* (*MCTS* for short), to tune the **TPC-H** benchmark workload and a real customer workload **Real-D** (see Section 7.1.1). We defer a formal discussion of ITC to Section 6.2. Roughly speaking, the ITC represents a *function* that maps from the number of what-if



(a) **TPC-H**, *two-phase greedy search* (b) **Real-D**, *MCTS*

Figure 2: Examples of index tuning curves of *two-phase greedy search* and *MCTS*, where we set the number of indexes allowed $K = 20$ and the budget on what-if calls $B = 20,000$.

calls made to the *percentage improvement* of the best configuration found, where the percentage improvement is defined as

$$\eta(W, C) = \frac{c(W, \emptyset) - c(W, C)}{c(W, \emptyset)} = 1 - \frac{c(W, C)}{c(W, \emptyset)}. \quad (1)$$

Here, W represents the input workload, C represents a configuration, and $\emptyset$ represents the *existing configuration* that index tuning starts from. $c(W, C) = \sum_{q \in W} c(q, C)$ represents the what-if cost of the workload W on top of the configuration C, which is the sum of the what-if costs of individual queries contained by W. In each plot of Figure 2, we use the red dashed line to represent the corresponding ITC. Intuitively, the ITC is a *profile* of the index tuner that characterizes its progress made so far with respect to the amount of budget on what-if calls being allocated.

This "diminishing return" behavior of existing budget-aware index tuning algorithms motivates us to introduce *early stopping*. Specifically, let ϵ (e.g., $\epsilon = 5\%$) be a user-given threshold that controls the *loss* on the percentage improvement, i.e., the gap between the percentage improvement of the best configuration found so far and the percentage improvement of the final best configuration with all budget allocated. If the projected improvement loss is below ϵ after certain amount of what-if calls are made, then we can safely terminate index tuning. Early stopping enables further savings on the number of what-if calls made in index tuning, and the savings can often be considerable. For example, as shown in Figure 2(a), *two-phase greedy search* requires making around 2,700 what-if calls to tune the **TPC-H** workload without early stopping. However, it actually makes no further progress (i.e., the best index configuration found does not change) after 1,000 what-if calls are made. Therefore, we would have saved 1,700 what-if calls, i.e., a reduction of 63%. While early stopping has been a well-known technique in the machine learning (ML) literature for preventing "overfitting" when training an ML model with an iterative method such as *gradient descent* [30, 31, 53], to the best of our knowledge we are the first to introduce it for index tuning with a very different goal of saving the amount of what-if calls.

Enabling early stopping for budget-aware index tuning, however, raises new challenges. First, to project the further improvement loss that is required by triggering early stopping, we need to know (1) the percentage improvement of the best configuration found so far and (2) the percentage improvement of the final best configuration *assuming that all budget were allocated*. Unfortunately, both are not available at the time point where the projection needs to be made. While it is clear that (2) is not available, one may wonder why (1) is also not available. Note that the best configuration found so far in budget-aware index tuning is based on *derived cost* (see Section 2.1) rather than the true what-if cost [51]. Technically, we can obtain (1) by making an extra what-if call for each query in the workload with the best configuration found. However, this is too expensive to be affordable in practice when tuning a large workload. Second, even if we know (1) and (2) so that we can compute the gap between (1) and (2) to verify whether the projected further improvement loss is below the threshold ϵ, it is unclear *when* this verification should be performed. Conducting this verification at the beginning of index tuning seems unnecessary, as the index tuner is expected to make fast progress; however, if this verification happens too late, then most of the savings given by early stopping will vanish.

To address these challenges, in this paper we propose *Esc*, a low-overhead early-stopping checker for budget-aware index tuning. It is based on the following main ideas:

- Instead of measuring the gap between (1) and (2), which cannot be obtained in practice, we develop a *lower-bound* for (1) and an *upper-bound* for (2) and then measure the gap between the lower and upper bounds. Clearly, if this gap is below the threshold ϵ, then the gap between (1) and (2) is also below ϵ. Figure 2 also presents the lower and upper bounds of each index tuning curve.
- To avoid verifying early-stopping either too early or too late, we develop a general approach that performs early-stopping verification by monitoring *improvement rate* of the ITC. Specifically, we measure the degree of *convexity*/*concavity* of the ITC based on the variation observed in its improvement rate, and we only verify early stopping when the ITC becomes *concave*.

In more detail, we develop the lower and upper bounds of percentage improvement by piggybacking on the previous work [43]. While [43] lays the foundation of deriving lower and upper bounds for what-if cost, the bounds work only for individual what-if calls but not the entire workload. The extension to workload-level bounds is nontrivial—a straightforward approach that simply sums up call-level bounds would lead to workload-level bounds that are too conservative to be useful (Section 4.1). Following this observation, we develop new mechanisms to improve over the naive workload-level bounds: (i) a *simulated greedy search* procedure that is designed for optimizing the bounds in the context of greedy search, which has been leveraged by both *two-phase greedy search* and *MCTS* as a basic building block (Section 4.2) and (ii) a generic approach to refining the bounds by modeling *index interactions* [33] at workload-level (Section 5). On the other hand, there can be multiple *concave* stages of an ITC, and only the *final* concave stage is worth early-stopping verification. For instance, this final stage of the ITC shown by Figure 2(b) begins after 6,000 what-if calls are made. It is challenging to identify whether a *concave* stage is the final one, and we further propose techniques to address this challenge and therefore reduce the chance of unnecessary early-stopping verification.

To summarize, this paper makes the following contributions:

- We introduce early stopping for budget-aware index tuning as a new mechanism that can result in significant savings on the number of what-if calls made (Section 3).
- We propose *Esc*, a novel framework that enables early-stopping in budget-aware index tuning by developing lower/upper bounds of workload-level what-if cost (Section 4) with refinement by exploiting index interactions (Section 5) and lightweight verification schemes that leverage improvement rates and convexity/concavity properties of the index tuning curve (Section 6).

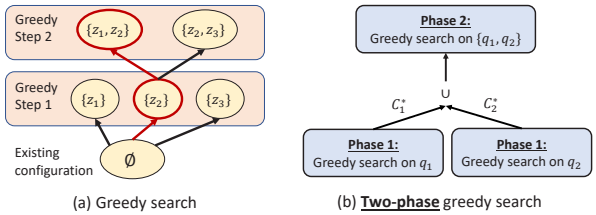


Figure 3: Example of budget-aware greedy search.

- We conduct extensive experimental evaluation using both industrial benchmarks and real workloads, and empirical results demonstrate that *Esc* can significantly reduce the number of what-if calls for state-of-the-art budget-aware tuning algorithms with little extra computational overhead and little or no improvement loss on the final configuration returned (Section 7).

Last but not least, while we focus on budget-aware index tuning algorithms in this work, early stopping can be applied to other index tuning algorithms such as (i) classic index tuning algorithms with *unlimited* budget of what-if calls [20, 43], which can be viewed as a special case of budget-aware index tuning and (ii) *anytime* index tuning algorithms [6], which are more sophisticated than budget-aware index tuning by constraining the overall index tuning time. Some of the technologies developed in this work, such as (a) the lower/upper bounds of workload-level what-if cost and (b) the general early-stopping verification scheme based on monitoring improvement rates of the index tuning curve, remain applicable, though their efficacy requires further investigation and evaluation. We leave this as an interesting direction for future work.

2 PRELIMINARIES

We present an overview of the problem of budget allocation and existing budget-aware index tuning algorithms.

2.1 Budget-aware Index Tuning

Budget-aware index tuning aims to constrain the amount of what-if calls that can be made during index tuning, in particular, during index configuration enumeration. An essential problem of budget-aware index tuning is *budget allocation*, i.e., determining on which query-configuration pairs to make what-if calls. For any query-configuration pair without making what-if call, we use the *derived cost* from *cost derivation* [7], defined by

$$d(q, C) = \min_{S \subseteq C} c(q, S), \tag{2}$$

as an approximation of its true what-if cost. There are two existing algorithms that address this budget allocation problem: (1) *two-phase greedy search* and (2) *Monte Carlo tree search* (MCTS). Based on the empirical study in [43], the gap between derived cost and the true what-if cost is below 5% for 80% to 90% of the what-if calls made by these two budget-aware index tuning algorithms.

2.1.1 Two-phase Greedy Search. A classic configuration enumeration algorithm is *greedy search* [7], as illustrated in Figure 3(a). It is a step-by-step procedure where it selects the next best candidate index in each *greedy step* that minimizes the workload cost, until the selected index configuration meets the given constraints. An improved version is the so-called *two-phase greedy search* [7], which first runs greedy search on top of each query to find its best candidate indexes and then runs greedy search again for the entire workload by taking the union of the best candidate indexes found for the individual queries. Figure 3(b) presents an example of *two-phase greedy search* with two queries in the workload. What-if calls are allocated in a "first come first serve" manner. *Two-phase greedy search* can achieve state-of-the-art performance [7, 20, 43, 51] in terms of the final index configuration found and has also been integrated into commercial database tuning software such as the Database Tuning Advisor (DTA) developed for Microsoft SQL Server [6],

Notation	Description
$c(q, C)$	The what-if cost of a QCP (q, C)
$c(W, C)$	The what-if cost of a WCP (W, C)
$\eta(W, C)$	The percentage improvement of a WCP (W, C)
$d(q, C)$	The derived cost of a QCP (q, C)
$d(W, C)$	The derived cost of a WCP (W, C)
$L(q, C)$	The lower bound of $c(q, C)$
$L(W, C)$	The lower bound of $c(W, C)$
$U(q, C)$	The upper bound of $c(q, C)$
$U(W, C)$	The upper bound of $c(W, C)$
$\Delta(q, C)$	The CI of q given C
$\delta(q, z, C)$	The MCI of an index z w.r.t. C and q
$u(q, z)$	The MCI upper bound of an index z w.r.t. q

Table 1: Notation and terminology (QCP: query-configuration pair; WCP: workload-configuration pair; CI: cost improvement; MCI: marginal cost improvement; q: a query; W: a workload; z: an index; C: an index configuration).

2.1.2 Monte Carlo Tree Search. To better tackle the trade-off between exploration and exploitation in budget allocation, previous work [51] proposed a budget-aware index tuning algorithm based on Monte Carlo tree search (*MCTS*). It models budget allocation as a Markov decision process (MDP) and allocates what-if calls with the goal of maximizing the "reward" that is defined by the percentage improvement (ref. Equation 1). After budget allocation is done, it runs greedy search again to find the best index configuration with the lowest derived cost (ref. Equation 2). It has been shown that *MCTS* outperforms *two-phase greedy search* under limited budget on the number of what-if calls [51].

2.2 What-if Call Interception

The two budget-aware index tuning algorithms discussed above allocate what-if calls at a *macro* level by treating each what-if call as a black box. That is, they use the what-if cost (or its approximation, e.g., derived cost) as the only signal to decide the next what-if call to be made. This results in wasted budget on *inessential* what-if calls that can be accurately approximated by their derived costs without affecting the result of index tuning. To skip these inessential what-if calls, previous work developed Wii [43], a what-if call *interception* mechanism that enables *dynamic* budget allocation in index tuning. The main idea there is to use lower/upper bounds of what-if cost: a what-if call can be skipped if the gap between the lower and upper bounds is sufficiently small. We present more details in Section 3.2. In this paper, we will build on top of these call-level lower/upper bounds to develop *Esc* that enables *early stopping* at workload-level index tuning. Moreover, in budget-constrained index tuning, skipping these inessential what-if calls can sharpen the efficacy of budget allocation by *reallocating* the budget to what-if calls that cannot be skipped. This results in improved versions of *two-phase greedy search* and *MCTS* algorithms with Wii integrated.

3 EARLY STOPPING IN INDEX TUNING

We start with the problem formulation of early stopping in budget-aware index tuning and then present an overview of the solution

that is based on lower/upper bounds of what-if cost. Table 1 summarizes the notation and terminology that will be used.

3.1 Problem Formulation

Let B be the budget on the number of what-if calls. At time t, i.e., when t what-if calls have been allocated, we want to decide if it is safe to skip allocating the remaining $B - t$ what-if calls without much loss on the improvement of the final index configuration returned. Formally, let C_t^* be the configuration found with $t \leq B$ what-if calls allocated. *That is, after t what-if calls we can only use derived cost when running the remaining part of configuration search.* Under this notation, C_B^* is the configuration found with all B what-if calls allocated. We stop index tuning if

$$\eta(W, C_B^*) - \eta(W, C_t^*) \leq \epsilon, \tag{3}$$

where $0 < \epsilon < 1$ is a user-defined threshold. By Equation 1,

$$c(W, C_t^*) - c(W, C_B^*) \leq \epsilon \cdot c(W, \emptyset). \tag{4}$$

Unfortunately, computing the left side of Equation 4 is impossible since $c(W, C_B^*)$ would only be known when all the B what-if calls were allocated, which negates the very purpose of *early stopping*. Moreover, the computation of $c(W, C_t^*)$ would require making $|W|$ extra what-if calls for each time point t, which would be prohibitively expensive for large workloads. As a result, we need a different approach instead of utilizing Equation 4 directly.

3.2 A Framework by Lower/Upper Bounds

We develop a lower bound $\eta_L(W, C_t^*)$ for $\eta(W, C_t^*)$ and an upper bound $\eta_U(W, C_B^*)$ for $\eta(W, C_B^*)$. That is, $\eta_L(W, C_t^*) \leq \eta(W, C_t^*)$ and $\eta(W, C_B^*) \leq \eta_U(W, C_B^*)$. As a result, if $\eta_U(W, C_B^*) - \eta_L(W, C_t^*) \leq \epsilon$, it then implies $\eta(W, C_B^*) - \eta(W, C_t^*) \leq \epsilon$ (i.e., Equation 3).

Figure 4 illustrates this framework in detail. The x-axis represents the number of what-if calls allocated, whereas the y-axis represents the percentage improvement of the corresponding best configuration found. Ideally, we should compare the *true* percentage improvements $\eta(W, C_t^*)$ and $\eta(W, C_B^*)$; however, since the true improvements are not observable, we instead compare the lower and upper bounds $\eta_L(W, C_t^*)$ and $\eta_U(W, C_B^*)$.

3.2.1 Conversion to Lower/Upper Bounds on What-if Costs. Our problem is equivalent to developing an upper bound $U(W, C_t^*) \geq c(W, C_t^*)$ and a lower bound $L(W, C_B^*) \leq c(W, C_B^*)$. As a result, $\eta_L(W, C_t^*) \leq \eta(W, C_t^*)$ and $\eta_U(W, C_B^*) \geq \eta(W, C_B^*)$.

To derive $L(W, C_B^*)$ and $U(W, C_t^*)$, we consider a more fundamental problem: Given an arbitrary configuration C, derive a lower bound $L(W, C)$ and an upper bound $U(W, C)$ such that $L(W, C) \leq c(W, C) \leq U(W, C)$. Since $c(W, C) = \sum_{q \in W} c(q, C)$, it is natural to first consider *call-level* lower and upper bounds $L(q, C)$ and $U(q, C)$ for a given query q such that $L(q, C) \leq c(q, C) \leq U(q, C)$. For this purpose, we reuse the results developed in previous work [43]. Below we provide a summary of the call-level lower/upper bounds. We will discuss extensions to workload-level bounds in Section 4.

3.2.2 Call-level Upper Bound. We assume the following *monotonicity* property of the what-if cost:

Assumption 1 (Monotonicity). *Let C_1 and C_2 be two index configurations where $C_1 \subseteq C_2$. Then $c(q, C_2) \leq c(q, C_1)$.*

That is, including more indexes into a configuration does not increase its what-if cost. We then have the derived cost $d(q, C) \geq c(q, C)$, which is a valid upper bound, i.e., $U(q, C) = d(q, C)$.

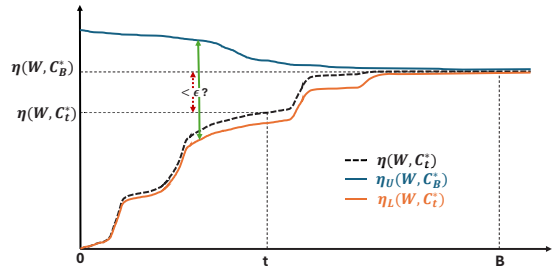


Figure 4: A framework for early-stopping in budget-aware index tuning based on workload-level bounds of what-if cost.

3.2.3 Call-level Lower Bound. We define the *cost improvement* of the query q given the configuration C as $\Delta(q, C) = c(q, \emptyset) - c(q, C)$. Moreover, we define the *marginal cost improvement* (MCI) of an index z with respect to a configuration C as $\delta(q, z, C) = c(q, C) - c(q, C \cup \{z\})$. Let $C = \{z_1, ..., z_m\}$. We can rewrite CI in terms of the MCI's, i.e., $\Delta(q, C) = \sum_{j=1}^{m} \delta(q, z_j, C_{j-1}) \leq \sum_{j=1}^{m} u(q, z_j)$, where $C_0 = \emptyset, C_j = C_{j-1} \cup \{z_j\}$, and $u(q, z_j)$ is an *upper bound* of the MCI $\delta(q, z_j, C_{j-1})$, for $1 \leq j \leq m$. Hence, we can set the lower bound

$$L(q, C) = c(q, \emptyset) - \sum_{j=1}^{m} u(q, z_j) \leq c(q, C). \tag{5}$$

3.2.4 MCI Upper Bounds. We further assume the following *submodularity* property of the what-if cost:

Assumption 2 (Submodularity). *Given two configurations X and Y s.t. $X \subseteq Y$ and an index $z \notin Y$, we have $c(q, Y) - c(q, Y \cup \{z\}) \leq c(q, X) - c(q, X \cup \{z\})$. Or equivalently, $\delta(q, z, Y) \leq \delta(q, z, X)$.*

That is, the MCI of an index z diminishes when z is included into a *larger* configuration with more indexes.

Assume monotonicity and submodularity of the cost function $c(q, X)$. Let Ω_q be the *best* possible configuration for q assuming that all candidate indexes have been created. We can set

$$u(q, z) = \min\{c(q, \emptyset), \Delta(q, \Omega_q), \Delta(q, \{z\})\}. \tag{6}$$

In practice, there are situations where we do not know $c(q, \{z\})$ and thus $\Delta(q, \{z\})$. In previous work [43], the authors proposed a lightweight approach to estimate $c(q, \{z\})$ based on the *coverage* of $\{z\}$ with respect to Ω_q, assuming that $c(q, \Omega_q)$ is known.

4 WORKLOAD-LEVEL BOUNDS

We now discuss how to leverage the call-level lower and upper bounds on what-if cost to establish lower/upper bounds that can be used at workload-level. We discuss both general-purpose bounds as well as optimized bounds for greedy search, which has been an essential step in state-of-the-art budget-aware index tuning algorithms such as *two-phase greedy search* and *MCTS*.

4.1 General-Purpose Bounds

4.1.1 Upper Bound of Workload Cost. The upper bound $U(W, C_t^*)$ can just be set to the derived cost $d(W, C_t^*)$, since we can show

$$d(W, C) = \sum_{q \in W} d(q, C) \geq \sum_{q \in W} c(q, C) = c(W, C)$$

for an arbitrary index configuration C. To obtain C_t^*, however, we need to continue with the index tuning algorithm on top of the current best configuration C_t found *without making more what-if calls*. As an example, we will illustrate this *simulation* process for greedy search in Section 4.2.1.

4.1.2 Lower Bound of Workload Cost. Let $C_B^* = \{z_1, ..., z_k\}$ for some $k \leq K$. By Equation 5, we could have set

$$L(W, C_B^*) = \sum_{q \in W} L(q, C_B^*) = \sum_{q \in W} \Big(c(q, \emptyset) - \sum_{i=1}^{k} u(q, z_i) \Big).$$

Unfortunately, this lower bound cannot be computed, because we do not know C_B^* and therefore the $\{z_1, ..., z_k\}$ at time $t < B$. However, for each query $q \in W$, if we order all candidate indexes z decreasingly with respect to their $u(q, z)$ and then take the top K candidate indexes in this ranking, it is easy to show that

$$\sum_{i=1}^{k} u(q, z_i) \leq \sum_{z \in \mathcal{U}(q,K)} u(q, z),$$

where $\mathcal{U}(q, K)$ represents the set of candidate indexes of q with the top-K largest MCI upper bounds. Therefore, we can instead set

$$L(W, C_B^*) = \sum_{q \in W} \Big(c(q, \emptyset) - \sum_{z \in \mathcal{U}(q,K)} u(q, z) \Big). \tag{7}$$

However, while this lower bound can be used for any budget-aware tuning algorithm, it may be too conservative. We next present optimizations of this lower bound for greedy search.

4.2 Optimizations for Greedy Search

Now let $C_B^* = \{z_1, ..., z_k\}$ for some $k \leq K$ where z_i represents the index selected by greedy search at the i-th step ($1 \leq i \leq k$) with B what-if calls allocated. The lower bound by applying Equation 5,

$$L(W, C_B^*) = \sum_{q \in W} \Big(c(q, \emptyset) - \sum_{i=1}^{k} u^{(i)}(q, z_i) \Big), \tag{8}$$

cannot be computed. Here, $u^{(i)}(q, z)$ is the $u(q, z)$ after the greedy step i and we use Procedure 1 to update the MCI upper bounds [43]:

Procedure 1. *For each index z that has not been selected by greedy search, we update $u(q, z)$ as follows:*

(a) Initialize $u(q, z) = \min\{c(q, \emptyset), \Delta(q, \Omega_q)\}$ for each index z.
(b) During each greedy step $1 \leq k \leq K$, update

$$u(q, z) = c(q, C_{k-1}) - c(q, C_{k-1} \cup \{z\}) = \delta(q, z, C_{k-1})$$

if both $c(q, C_{k-1})$ and $c(q, C_{k-1} \cup \{z\})$ are available, where C_k is the configuration selected by the greedy step k and $C_0 = \emptyset$.

Our idea is to further develop an upper bound for $\sum_{i=1}^{k} u^{(i)}(q, z_i)$ by running a *simulated greedy search* procedure described below.

4.2.1 Simulated Greedy Search. For ease of exposition, consider tuning a workload with a single query q using greedy search.

Procedure 2. *At time t (i.e., when $t < B$ what-if calls have been made), run greedy search to get up to K indexes in total, where each greedy step j selects the index z'_j with the maximum $u^{(j)}(q, z'_j) > 0$.*

Let the configuration found by Procedure 2 be $C_t^u = \{z'_1, z'_2, ..., z'_l\}$ where $l \leq K$. If $l < K$, then it means that any remaining index z satisfies $u(q, z) = 0$. As a result, we can assume $l = K$.

Theorem 1. *$\sum_{j=1}^{K} u^{(j)}(q, z'_j) \geq \sum_{i=1}^{k} u^{(i)}(q, z_i)$. As a result,*

$$L(q, C_B^*) = c(q, \emptyset) - \sum_{j=1}^{K} u^{(j)}(q, z'_j) \tag{9}$$

is a lower bound of the what-if cost $c(q, C_B^*)$ for greedy search.

Table: R (a, b, c, d)	**Queries**
Indexes	q_1 : SELECT a, b FROM R WHERE R.b = 10
z_1 : [R.b; R.a]	q_2 : SELECT a, b FROM R WHERE
z_2 : [R.b, R.a, R.c]	R.b > 10 AND R.a > 20 AND R.c > 30

Figure 5: An example of index interaction

Due to space constraints, all proofs are deferred to the full version of this paper [42]. We next generalize this result to multi-query workload with the understanding that the index z'_j is selected for the entire workload W with the maximum $u^{(j)}(W, z'_j) > 0$, i.e.,

$$L(W, C_B^*) = c(W, \emptyset) - \sum_{j=1}^{K} u^{(j)}(W, z'_j), \tag{10}$$

where $u(W, z) = \sum_{q \in W} u(q, z)$.

Moreover, as we mentioned in Section 4.1.1, the simulated greedy search outlined in Procedure 2 can be reused for computing the upper bound $U(W, C_t^*)$ with slight modification. Details of this revised simulated greedy search are included in the full version [42].

4.2.2 Lower Bound for Two-phase Greedy Search. We update the MCI upper-bounds for *two-phase greedy search* as follows:

Procedure 3. *For index z and query q, update $u(q, z)$ as follows:*

(a) Initialize $u(q, z) = \min\{c(q, \emptyset), \Delta(q, \Omega_q)\}$ for each index z.
(b) In Phase 1, update $u(q, z)$ based on Equation 6.
(c) In Phase 2, during each greedy step $1 \leq k \leq K$, update

$$u(q, z) = c(q, C_{k-1}) - c(q, C_{k-1} \cup \{z\}) = \delta(q, z, C_{k-1})$$

if both $c(W, C_{k-1})$ and $c(q, C_{k-1} \cup \{z\})$ are available, where C_k is the configuration selected by greedy search in step k ($C_0 = \emptyset$) and z has not been included in C_k.

The update step (c) excludes pathological cases where $c(W, C_k)$ is unknown but both $c(q, C_k)$ and $c(q, C_k \cup \{z\})$ are known for a particular query q (due to Phase 1).

Theorem 2. *The $L(W, C_B^*)$ defined in Equation 10 remains a lower bound of $c(W, C_B^*)$ for* two-phase greedy search *if we maintain the MCI upper-bounds by following Procedure 3.*

4.2.3 Lower Bound for Monte Carlo Tree Search. We can use the same simulated greedy search to obtain $L(W, C_B^*)$, given that there is a final greedy search stage in *MCTS* after all budget allocation is done. However, we are only able to use Equation 6 for maintaining the MCI upper bounds—we can prove that it is safe to do so using the same argument as in *two-phase greedy search* when t is in Phase 1 (see the full version [42]). It remains future work to investigate further improvement over Equation 6 for *MCTS*.

5 REFINEMENT WITH INDEX INTERACTION

Our approach of computing the lower bounds $L(q, C_B^*)$ and $L(W, C_B^*)$ in Equations 9 and 10 basically sums up the MCI Upper-bounds of individual indexes. This ignores potential *index interactions*, as illustrated by the following example.

Example 1 (Index Interaction). *As shown in Figure 5, let R be a table with four columns a, b, c, and d. Let z_1 and z_2 be two indexes on R, where z_1 has a single key column b with a as an included column, and z_2 has a compound key with three columns b, a, and c in order. Consider the SQL query q_1 in Figure 5. Both z_1 and z_2 have very similar, if not the same, cost improvement for q_1, as one can use an index scan on top of either z_1 and z_2 to evaluate q_1 without consulting*

the table R. As a result, if z_1 (resp. z_2) has been included in some configuration, including z_2 (resp. z_1) cannot further improve the cost of q_1. In other words, we have roughly the same cost improvements for z_1, z_2, and $\{z_1, z_2\}$, i.e., $\Delta(q_1, \{z_1\}) \approx \Delta(q_1, \{z_2\}) \approx \Delta(q_1, \{z_1, z_2\})$.

Note that index interaction is *query-dependent*. To see this, consider the same z_1 and z_2 in Example 1 but a different SQL query q_2 in Figure 5. Since z_1 can hardly be used for evaluating q_2, we have $\Delta(q_2, \{z_1\}) \approx 0$ (see [42] for details). As a result, in the presence of both z_1 and z_2, the query optimizer will pick z_2 over z_1; hence, we have $\Delta(q_2, \{z_1, z_2\}) = \Delta(q_2, \{z_2\}) \approx \Delta(q_2, \{z_1\}) + \Delta(q_2, \{z_2\})$. Therefore, z_1 and z_2 *do not interact* in the case of q_2.

5.1 Index Interaction

Motivated by Example 1, given two indexes z_1, z_2 and a query q, we define the *index interaction* between z_1 and z_2 w.r.t. q as

$$\mathcal{I}(z_1, z_2|q) = \frac{\Delta_U(q, \{z_1, z_2\}) - \Delta(q, \{z_1, z_2\})}{\Delta_U(q, \{z_1, z_2\}) - \Delta_L(q, \{z_1, z_2\})}.$$

Here, $\Delta_L(q, \{z_1, z_2\}) = \max\{\Delta(q, \{z_1\}), \Delta(q, \{z_2\})\}$ is a lower bound of $\Delta(q, \{z_1, z_2\})$ based on Assumption 1 (i.e., monotonicity), and $\Delta_U(q, \{z_1, z_2\}) = \Delta(q, \{z_1\}) + \Delta(q, \{z_2\})$ is an upper bound of $\Delta(q, \{z_1, z_2\})$ based on Assumption 2 (i.e., submodularity).

We now extend the above definition to define the interaction between an index z and an index configuration C w.r.t. a query q:

$$\mathcal{I}(z, C|q) = \frac{\Delta_U(q, C \cup \{z\}) - \Delta(q, C \cup \{z\})}{\Delta_U(q, C \cup \{z\}) - \Delta_L(q, C \cup \{z\})}.$$

Similarly, $\Delta_L(q, C \cup \{z\}) = \max\{\Delta(q, C), \Delta(q, \{z\})\}$ is a lower bound of $\Delta(q, C \cup \{z\})$ by Assumption 1, and $\Delta_U(q, C \cup \{z\}) = \Delta(q, C) + \Delta(q, \{z\})$ is an upper bound of $\Delta(q, C \cup \{z\})$ by Assumption 2.

5.2 A Similarity-based Approach

Note that the interaction $\mathcal{I}(z, C|q)$ defined above cannot be directly computed if we do not have knowledge about $\Delta(q, C)$ and $\Delta(q, C \cup \{z\})$. Therefore, we propose an *implicit* approach to measure index interaction based on the *similarity* between indexes. Intuitively, if two indexes are similar, e.g., they share similar key columns where one is a prefix of the other, then it is likely that one of them cannot improve the workload cost given the presence of the other. As a result, there is strong interaction between the two indexes.

Specifically, given a query q and two indexes z_1, z_2, we compute the similarity $\mathcal{S}(z_1, z_2|q)$ between z_1 and z_2 w.r.t. q as follows:

(1) Convert the query and indexes into feature vectors $\vec{\mathbf{q}}$, $\vec{\mathbf{z}}_1$, and $\vec{\mathbf{z}}_2$. We reuse the feature representation in previous work [37, 43] for this purpose. In more detail, we collect all indexable columns from the workload. Let D be the number of indexable columns collected. We then represent $\vec{\mathbf{q}}$, $\vec{\mathbf{z}}_1$, and $\vec{\mathbf{z}}_2$ as D-dimensional vectors. We assign weights to each indexable column in the query representation $\vec{\mathbf{q}}$ by using the approach proposed in ISUM [37]. Specifically, the weight of a column is computed based on its corresponding table size and the number of candidate indexes that contain it. We further assign weights to each indexable column in the index representation $\vec{\mathbf{z}}$ by using the approach proposed in Wii [43]. Specifically, the weight of a column is determined by its position in the index z, e.g., whether it is a *key column* or an *included column* of z.

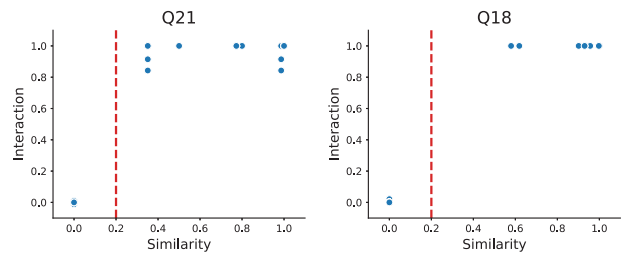


Figure 6: Relationship between pairwise index interaction and pairwise index similarity (TPC-H).

(2) Project the index vectors onto the query vector using dot product, i.e., $\vec{\mathbf{z}}_i^{\mathbf{q}} = \vec{\mathbf{z}}_i \cdot \vec{\mathbf{q}}$ for $i \in \{1, 2\}$. Note that the resulting vectors $\vec{\mathbf{z}}_i^{\mathbf{q}}$ for $i \in \{1, 2\}$ remain D-dimensional vectors. This projection filters out columns in $\vec{\mathbf{z}}_i$ that do not appear in $\vec{\mathbf{q}}$ and therefore do not have impact on the query performance of q.

(3) Calculate the cosine similarity $\mathcal{S}(z_1, z_2|q) = \frac{\vec{\mathbf{z}}_1^{\mathbf{q}} \cdot \vec{\mathbf{z}}_2^{\mathbf{q}}}{\|\vec{\mathbf{z}}_1^{\mathbf{q}}\| \cdot \|\vec{\mathbf{z}}_2^{\mathbf{q}}\|}$.

We can further extend $\mathcal{S}(z_1, z_2|q)$ to represent the similarity between an index z and an index configuration C w.r.t. a query q: $\mathcal{S}(z, C|q) = \frac{\vec{\mathbf{z}}^{\mathbf{q}} \cdot \vec{\mathbf{C}}^{\mathbf{q}}}{\|\vec{\mathbf{z}}^{\mathbf{q}}\| \cdot \|\vec{\mathbf{C}}^{\mathbf{q}}\|}$. All we need is a feature representation $\vec{\mathbf{C}}$ of the configuration C. For this purpose, we use the same approach as in Wii [43], where we featurize an index configuration as a D-dimensional vector as follows. For each dimension d ($1 \le d \le D$), we take the maximum of the feature values from the corresponding dimensions d of the feature representations of the indexes contained by the configuration. The intuition is that, if an indexable column appears in multiple indexes of the configuration, we take the largest weight that represents its most significant role (e.g., a leading key column in some index).

Ideally, we would wish the $\mathcal{S}(z, C|q)$ to be equal to $\mathcal{I}(z, C|q)$. Unfortunately, this is not the case. To shed some light on this, we conduct an empirical study to measure the *correlation* between pairwise index interaction $\mathcal{I}(z_1, z_2|q)$ and pairwise index similarity $\mathcal{S}(z_1, z_2|q)$, using the workloads summarized in Table 2. Specifically, we pick the most costly queries for each workload and evaluate the what-if costs of all single indexes (i.e., singleton configurations) for each query. We then select the top 50 indexes w.r.t. their cost improvement (CI) in decreasing order and evaluate the what-if costs of all $50 \times 49 = 2,450$ configurations that contain a pair of the top-50 indexes. Finally, we compute the pairwise index interaction and the pairwise index similarity of these index pairs. Figure 5 presents their correlation for the two most costly queries of **TPC-H**, and similar results over the other queries and workloads are included in the full version [42]. We observe that there is no strong correlation between the two. Instead, for most of the queries, there is a sudden jump on the pairwise index interaction when the pairwise index similarity increases. That is, when the pairwise index similarity exceeds a certain threshold (e.g., 0.2), the pairwise index interaction will increase to a high value (e.g., close to 1). This motivates us to propose a threshold-based mechanism to utilize the index similarity to characterize the impact of index interaction.

5.3 Refined Workload-Level Lower Bound

Our basic idea is the following. During each step of the simulated greedy search (SGS) when selecting the next index to be included,

we consider not only the benefit of the index, but also its interaction with the indexes *that have been selected in previous steps* of SGS. Specifically, we quantify the *conditional benefit* $\mu^{(j)}(q, z'_j)$ of the candidate index z'_j based on its interaction with the SGS-selected configuration $C_{j-1} = \{z'_1, ..., z'_{j-1}\}$ and use it to replace the MCI upper bound $u^{(j)}(q, z'_j)$ in Procedure 2 as follows:

$$\mu^{(j)}(q, z'_j) = \begin{cases} 0, & \text{if } \mathcal{S}(z'_j, C_{j-1}|q) > \tau; \\ u^{(j)}(q, z'_j), & \text{otherwise.} \end{cases} \tag{11}$$

Here, $0 \leq \tau \leq 1$ is a threshold. In our experimental evaluation (see Section 7), we found that this threshold-based mechanism can significantly improve the lower bound for *two-phase greedy search* but remains ineffective for *MCTS*, due to the presence of many query-index pairs with unknown what-if costs. We therefore further propose an optimization for *MCTS*. Specifically, for a query-index pair (q, z) with unknown what-if cost, we initialize its MCI upper bound by averaging the MCI upper bounds of indexes with known what-if costs that are similar to z w.r.t. q (see [42] for details).

6 EARLY-STOPPING VERIFICATION

Based on the workload-level lower/upper bounds in Sections 4 and 5, we develop *Esc*, an early-stopping checker for budget-aware index tuning. One main technical challenge faced by *Esc* is to understand *when* to invoke early-stopping verification. While one can employ simple strategies such as a fixed-step verification scheme where a verification is invoked every s what-if calls, as we will see in our experimental evaluation (Section 7) such strategies may incur high computation overhead since obtaining the lower and upper bounds (e.g., by using the simulated greedy search procedure in Section 4.2.1) comes with a cost. In this section, we present our solutions to this problem. We start by giving a heuristic solution to *two-phase greedy search* that exploits special structural properties of this algorithm (Section 6.1). We then propose a generic solution (Section 6.3) by only leveraging improvement rates and convexity properties of the index tuning curve (Section 6.2) without requiring any algorithm-specific knowledge.

6.1 Heuristic Verification Scheme

There is some trade-off in terms of *when* to invoke early-stopping verification (ESV): if we invoke ESV too frequently, then the computation overhead may become considerable; on the other hand, if we invoke ESV insufficiently, then we may miss opportunities for stopping index tuning earlier and allocate more what-if calls than necessary. Clearly, in the early stages of index tuning, there is no need to check for early-stopping, as the index tuning algorithm is still making rapid progress. Ideally, one needs to *detect* when the progress of the index tuning algorithm starts to slow down.

For *two-phase greedy search*, this *inflection point* is not difficult to tell. As an example, consider Figure 2(a) where we run *two-phase greedy search* to tune the **TPC-H** workload. In Figure 2(a) we have marked each greedy step within both Phase 1 and Phase 2. We observe that the progress starts to slow down significantly after the search enters Phase 2, especially during or after the first greedy step of Phase 2. As a result, we can simply skip Phase 1 and start checking early-stopping at the beginning of each greedy step of Phase 2. Our experiments in Section 7 confirm that this simple scheme can result in effective early-stopping while keeping the computation overhead negligible.

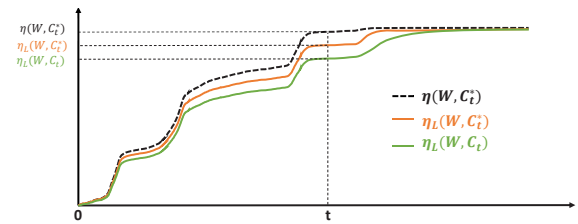


Figure 7: Characterization of the relationship between different definitions of index tuning curve.

This heuristic early-stopping verification scheme clearly cannot work for other algorithms such as *MCTS*. However, the above discussion hinted us to focus on looking for similar *inflection points* of index tuning curves. It leads to a generic early-stopping verification scheme that only relies on improvement rates and convexity properties of index tuning curves, as we will present next.

6.2 Index Tuning Curve Properties

We define the *index tuning curve* (ITC) as a function that maps from the number of what-if calls allocated at time t to the percentage improvement $\eta(W, C_t^*)$ of the corresponding best index configuration found. By definition, the ITC is *monotonically non-decreasing*. The dash line in Figure 4 presents an example of ITC.

Unfortunately, as we have discussed in Section 3.1, the ITC defined above cannot be directly observed without making extra what-if calls. One option is to replace $\eta(W, C_t^*)$ with its lower bound $\eta_L(W, C_t^*)$. However, the computation of $\eta_L(W, C_t^*) = 1 - \frac{d(W, C_t^*)}{c(W, \emptyset)}$ is not free (e.g., requiring running the simulated greedy search) and we therefore choose to use $\eta_L(W, C_t) = 1 - \frac{d(W, C_t)}{c(W, \emptyset)}$, where C_t is the *observed* best configuration at time t without continuing tuning, in lieu of $\eta_L(W, C_t^*)$. $\eta_L(W, C_t)$ is directly available at time t without extra computation. Assuming monotonicity of what-if cost (i.e., Assumption 1), we have $\eta(W, C_t) \leq \eta_L(W, C_t^*)$, because $d(W, C_t) \geq d(W, C_t^*)$ given that C_t is a subset of C_t^*. Figure 7 characterizes the relationship between different definitions of ITC.

6.2.1 Improvement Rate. Suppose that we check *early stopping* at n time points with B_j what-if calls allocated at time point j, where $1 \leq j \leq n$. We call this sequence $\{B_j\}_{j=1}^n$ an *early-stopping verification scheme* (ESVS). Let the observed percentage improvement at time point j be I_j, i.e., $I_j = \eta_L(W, C_{B_j})$. We further define a starting point (B_0, I_0) where we have known both B_0 and I_0. By default, we choose $B_0 = 0$ and $I_0 = 0$.

Definition 1 (Improvement Rate). *We define the* improvement rate r_j *at time point* j *as* $r_j = \frac{I_j - I_0}{B_j - B_0}$.

The *projected improvement* at time point j for budget b of what-if calls (i.e., by making $b - B_j$ more what-if calls) is then defined as

$$p_j(b) = I_j + r_j \cdot (b - B_j). \tag{12}$$

For the default case where $B_0 = 0$ and $I_0 = 0$, we have $p_j(b) = I_j \cdot \frac{b}{B_j}$. For ease of exposition, we will use this default setup in the rest of our discussion throughout this section.

Definition 2 (Latest Improvement Rate). *We define the* latest improvement rate l_j *at time point* j *as* $l_j = \frac{I_j - I_{j-1}}{B_j - B_{j-1}}$.

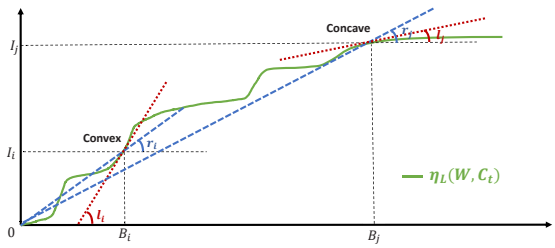


Figure 8: Relationship between improvement rates and convexity/concavity of index tuning curve. The latest improvement rate l_j approximates the *tangent* of the index tuning curve at the point (B_j, I_j).

6.2.2 Convexity and Concavity. Let $I = f(b)$ be the function that represents the index tuning curve. That is, $f(b) = \eta_L(W, C_b)$ where C_b is the observed best configuration with b what-if calls allocated.

Lemma 1. *If f is strictly concave and twice-differentiable, then $f'(b) < \frac{f(b)}{b}$ for any $0 < b \leq B$.*

We have the following immediate result based on Lemma 1:

Theorem 3. *If f is strictly concave and twice-differentiable, then $l_j < r_j$ for a given* early-stopping verification scheme $\{B_j\}_{j=1}^{n}$.

We have a similar result for a *convex* index tuning curve:

Theorem 4. *If f is strictly convex and twice-differentiable, then $l_j > r_j$ for a given* early-stopping verification scheme $\{B_j\}_{j=1}^{n}$.

6.2.3 Summary and Discussion. The previous analysis implies some potential relationship between the improvement rates that we defined and the *convexity/concavity* properties of an index tuning curve: (1) if the index tuning curve in (B_{j-1}, B_j) is convex, i.e., it is making accelerating progress, then we will observe $l_j > r_j$; (2) on the other hand, if the index tuning curve in (B_{j-1}, B_j) is concave, then we will observe $l_j < r_j$. Figure 8 illustrates this relationship.

In practice, an index tuning curve can be partitioned into ranges where in each range the curve can fall into one of the three categories: (1) *convex*, (2) *concave*, and (3) *flat* (i.e., $l_j = 0$). In general, we would expect that the curve is more likely to be convex in early stages of index tuning and is more likely to be concave or flat towards the end of tuning. This observation leads us to develop a generic ESVS that will be detailed next, where we leverage the convexity of the ITC to skip unnecessary invocations of early-stopping verification and put the overall verification overhead under control.

6.3 Generic Verification Scheme

We start from the aforementioned simple ESVS with fixed step size s, i.e., $B_j = B_{j-1} + s$, where s can be a small number of what-if calls. We then compute l_j and r_j at each B_j accordingly.

Now consider a specific time point j. If we observe that $l_j > r_j$, then it is likely that the index tuning curve in (B_{j-1}, B_j) is convex. Note that the condition in Theorem 4 is not necessary, so the convexity is not guaranteed when observing $l_j > r_j$. In this case we can *skip* the early-stopping verification, because the index tuner is still making *accelerating* progress. On the other hand, if we observe that $l_j < r_j$, then it is likely that the index tuning curve in (B_{j-1}, B_j) is concave, i.e., the progress is *decelerating*, which implies that we perhaps can perform a verification.

There are some subtleties in the above proposal. First, although it is reasonable to assume that the index tuning curve will *eventually* become concave/flat, it is not guaranteed that the index tuner has entered this final stage of tuning when $l_j < r_j$ is observed. Second, even if the index tuner has entered the final stage, the deceleration process may be slow before we can conclude that the improvement loss will be lower than the user-given threshold ϵ, which voids the necessity of the (expensive) early-stopping verification.

6.3.1 Significance of Concavity. To address these challenges, we measure the *significance* of the potential concavity of the index tuning curve. For this purpose, we *project* the percentage improvement at B_{j+1} using the improvement rates l_j and r_j and compare it with I_{j+1} to decide whether we want to invoke early-stopping verification (ESV) at the time point $j+1$. Specifically, we define the projected *improvement gap* between the projected improvements p^r_{j+1} and p^l_{j+1} (using Equation 12) as $\Delta_{j+1} = p^r_{j+1} - p^l_{j+1}$. Clearly, $\Delta_{j+1} > 0$ since $l_j < r_j$. Moreover, the larger Δ_{j+1} is, the more significant the corresponding *concavity* is. Therefore, intuitively, we should have a higher probability of invoking ESV.

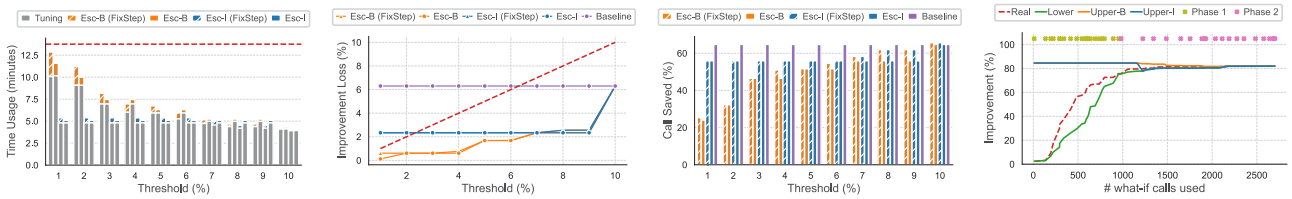
Now consider the relationship between I_{j+1} and $p^{l,r}_{j+1}$. We have the following three possible cases:

- $p^l_{j+1} < p^r_{j+1} < I_{j+1}$: This suggests that f grows even faster than r_j when moving from B_j to B_{j+1}, which implies that a verification at $j+1$ is unnecessary.
- $p^l_{j+1} < I_{j+1} < p^r_{j+1}$: This suggests that f grows more slowly than r_j but faster than l_j. We further define $\delta_{j+1} = p^r_{j+1} - I_{j+1}$ and define the *significance of concavity* σ_{j+1} as $\sigma_{j+1} = \frac{\delta_{j+1}}{\Delta_{j+1}}$. Clearly, $0 < \delta_{j+1} < \Delta_{j+1}$. We then set a threshold $0 < \sigma < 1$ and perform an early-stopping verification if $\sigma_{j+1} \geq \sigma$.
- $I_{j+1} < p^l_{j+1}$: This suggests that f grows even more slowly than l_j, which implies that a verification at $j+1$ is perhaps helpful.

6.3.2 A Probabilistic Mechanism for Invoking ESV. One problem is that, if the observed improvement is flat (i.e., $l_i = 0$) but the lower and upper bounds are not converging yet, then it may result in unnecessary ESV invocations. We therefore need to further consider the convergence of the bounds. Specifically, we use the following probabilistic mechanism for invoking ESV. We define $\rho_j = \frac{U_j(W, C^*_B) - L_j(W, C^*_t)}{\epsilon}$ as the *relative gap* w.r.t. the threshold ϵ of improvement loss. Instead of always invoking ESV as was outlined in Section 6.3.1, we invoke it with probability $\lambda_j = \frac{1}{\rho_j}$.

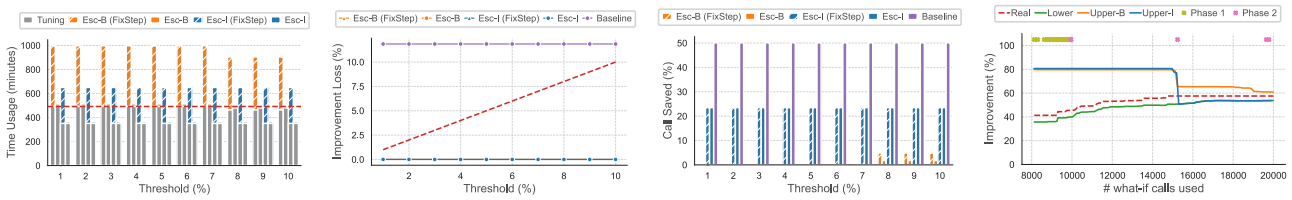
6.3.3 Refinement of Improvement Rates. If early-stopping verification is invoked at B_{j+1}, there will be two possible outcomes:

- The early-stopping verification returns true, then we terminate index tuning accordingly.
- The early-stopping verification returns false. In this case, we let $L_{j+1}(W, C^*_t)$ and $U_{j+1}(W, C^*_B)$ be the lower and upper bounds returned. We can use L_{j+1} and U_{j+1} to further refine the improvement rates l_{j+1} and r_{j+1}. Specifically, we have $p^r_{j+2} = I_{j+1} + r_{j+1} \cdot s < U_{j+1}$ and $p^l_{j+2} = I_{j+1} + l_{j+1} \cdot s < U_{j+1}$, which gives $r_{j+1} < \frac{U_{j+1} - I_{j+1}}{s}$ and $l_{j+1} < \frac{U_{j+1} - I_{j+1}}{s}$. Therefore, $r_{j+1} = \min\{\frac{I_{j+1}}{B_{j+1}}, \frac{U_{j+1} - I_{j+1}}{s}\}$, and $l_{j+1} = \min\{\frac{I_{j+1} - I_j}{s}, \frac{U_{j+1} - I_{j+1}}{s}\}$. This refinement can be applied to all later steps $j+3, j+4, \cdots$ as well.



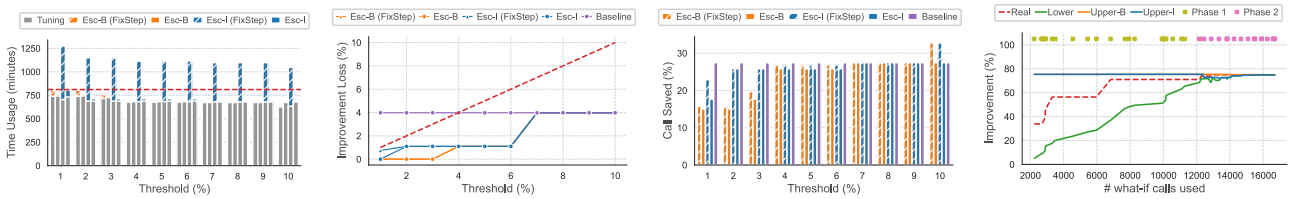
(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 9: Two-phase greedy search, TPC-H, $K = 20$, $B = 20k$.



(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 10: Two-phase greedy search, TPC-DS, $K = 20$, $B = 20k$.



(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 11: Two-phase greedy search, Real-D, $K = 20$, $B = 20k$.

Name	DB Size	#Queries	#Tables	#Joins	#Scans	#Indexes
TPC-H	sf=10	22	8	2.8	3.7	168
TPC-DS	sf=10	99	24	7.7	8.8	848
JOB	9.2GB	33	21	7.9	2.5	66
Real-D	587GB	32	7,912	15.6	17	417
Real-M	26GB	31	474	13.3	14.3	642

Table 2: Summary of database and workload statistics.

7 EVALUATION

We conduct extensive experimental evaluation of *Esc* and report the evaluation results in this section.

7.1 Experiment Settings

7.1.1 Databases and Workloads. We use standard benchmarks as well as real customer workloads in our experiments. For benchmark workloads, we use (1) **TPC-H**, (2) **TPC-DS**, and (3) the "Join Order Benchmark" (**JOB**) [22]. We also use two real workloads, denoted by **Real-D** and **Real-M**. Table 2 summarizes some basic properties of the workloads, in terms of schema complexity (e.g., the number of tables), query complexity (e.g., the average number of joins and table scans contained by a query), database/workload size, and the number of candidate indexes found for index tuning.

7.1.2 Budget-aware Index Tuning Algorithms. We focus on evaluating two state-of-the-art budget-aware index tuning algorithms, (1) *two-phase greedy search* and (2) *MCTS*, as well as their enhanced versions with Wii, i.e., *what-if call interception* [43].

7.1.3 Variants of Early-Stopping Verification Schemes. We use the heuristic ESVS in Section 6.1 for *two-phase greedy search* and use the generic ESVS in Section 6.3 for *MCTS*. We compare four variants: (1) ***Esc*-B**, where we use the corresponding ESVS with lower/upper bounds that do not consider index interaction; (2) ***Esc*-I**, which further uses index interaction to refine the lower bound, as discussed in Section 5.3; (3) ***Esc*-B (FixStep)**, which is a baseline of ***Esc*-B** that instead adopts the fixed-step ESVS; and similarly, (4) ***Esc*-I (FixStep)**, a baseline of ***Esc*-I** with the fixed-step ESVS.

7.1.4 Evaluation Metrics. We vary the improvement-loss threshold ϵ from 1% to 10% in our evaluation. For each ϵ, let b_ϵ be the number of what-if calls allocated when early-stopping is triggered, and let $\tilde{B}$ be the number of what-if calls allocated without early-stopping. Note that $\tilde{B}$ can be smaller than the budget B on the number of what-if calls, because algorithms such as greedy search can terminate if no better configuration can be found (regardless of whether there is remaining budget on the number of what-if calls). We then measure the following performance metrics of early-stopping: (a) *extra time overhead of early-stopping verification*, which is measured as the total time spent on invoking early-stopping verification; (b) *improvement loss*, defined as $\Delta(b_\epsilon) = \eta(W, C_B^*) - \eta(W, C_{b_\epsilon}^*)$; and (c) *savings on the number of what-if calls*, defined as $(1 - \frac{b_\epsilon}{\tilde{B}}) \times 100\%$.

7.1.5 Other Experimental Settings. We vary the number of indexes allowed $K \in \{10, 20\}$. We set the budget on what-if calls $B = 20,000$ to make sure that index tuning can finish without early stopping; otherwise, early stopping would have never been triggered, which is correct but a tedious situation. Moreover, we set the threshold of index interaction for refinement of the lower-bound in Section 5.3 to be $\tau = 0.2$, based on our empirical study in [42]. For the generic ESVS in Section 6.3 and the baseline fixed-step ESVS, we set the step size $s = 100$ (see [42] for results with $s = 500$); furthermore, we set the threshold $\sigma = 0.5$ for the significance of concavity.

7.1.6 Baselines. We also compare *Esc* with baseline approaches that are based on simple heuristics. Specifically, for *two-phase greedy search*, we compare *Esc* with a baseline that simply stops tuning after the first phase of greedy search; for *MCTS*, we compare *Esc* with a baseline that simply stops tuning if the observed percentage improvement I_j over the existing configuration is greater than some fixed threshold (we set the threshold to be 30% in our evaluation).

7.2 Two-phase Greedy Search

Figures 9 to 11 present the results when running *two-phase greedy search* on top of **TPC-H**, **TPC-DS**, and **Real-D**. The results on **JOB** and **Real-M** are included in [42]. In each figure, we present (a) the extra time overhead (in minutes) of early-stopping verification, (b) the improvement loss when early-stopping is triggered, (c) the savings on the number of what-if calls, and (d) the index tuning curve as well as the corresponding lower and upper bounds.

7.2.1 Extra Time Overhead of Early-Stopping Verification. As a reference point, in each plot (a) the red dashed line represents the corresponding index tuning time *without* early-stopping verification, whereas the gray bars represent the net index tuning time *with* early-stopping verification. We observe that the extra time overhead of both ***Esc*-B** and ***Esc*-I** is negligible compared to the index tuning time, across all workloads tested. On the other hand, ***Esc*-B (FixStep)** and ***Esc*-I (FixStep)** sometimes result in considerable extra time overhead. For example, as shown in Figure 10(a), on **TPC-DS** the extra time overhead of ***Esc*-B (FixStep)** is comparable to the index tuning time when varying the threshold ϵ from 1% to 7%. Overall, the savings in terms of end-to-end index tuning time by applying *Esc* resonate with the corresponding savings on what-if calls shown in each plot (c).

7.2.2 Improvement Loss. The red dashed line in each plot (b) delineates the acceptable improvement loss. That is, any improvement loss above that line violates the threshold ϵ set by the user. We observe that violation occurs rarely, e.g., when setting $\epsilon = 1\%$ on **TPC-H** and using ***Esc*-I** for early stopping. Moreover, the actual improvement loss is often much smaller than the threshold ϵ when early-stopping is triggered. One reason for this is that our lower bound $\eta_L(W, C_t^*)$ and upper bound $\eta_U(W, C_B^*)$ are more conservative than the actual improvements $\eta(W, C_t^*)$ and $\eta(W, C_B^*)$ needed for triggering early-stopping (ref. Section 3.2).

7.2.3 Savings on What-If Calls. The plot (c) in each figure represents the (percentage) savings on the number of what-if calls. We have the following observations. First, the savings typically increase as the threshold ϵ increases. Intuitively, a less stringent ϵ can trigger early-stopping sooner. Second, the savings vary on different workloads. For example, with $\epsilon = 5\%$, the savings are around 60% on **TPC-H**; however, the savings drop to 25% on **TPC-DS** and **Real-D**. We can understand this better by looking at the corresponding index tuning curve in the plot (d). Third, considering index interaction typically leads to an improved upper bound, which results in more savings on what-if calls.

7.2.4 Comparison with Baseline. We now compare *Esc* with the baseline approach that simply stops tuning after the first phase of greedy search, in terms of the improvement loss and the savings on what-if calls. As shown by the plots (b) and (c) of each figure, the baseline can achieve higher savings on what-if calls but can suffer from significantly higher improvement loss. For example, as Figure 10(b) shows, on **TPC-DS** the improvement loss of the baseline is around 12% while *Esc* has zero improvement loss.

7.3 Monte Carlo Tree Search

Figures 12 and 13 present the results for *MCTS* on **TPC-H** and **Real-D**. The results on the other workloads can be found in [42].

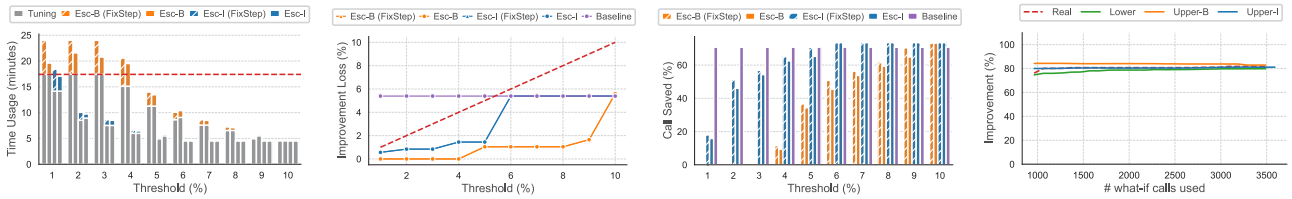
7.3.1 Extra Time Overhead of Early-Stopping Verification. Again, we observe that the extra time overhead of early-stopping verification is negligible compared to the index tuning time in most of the cases tested. However, we also notice a few cases where the extra time overhead of early-stopping verification is considerable. This typically happens when it is difficult to trigger early-stopping using the lower and upper bounds. As a result, all the ESV invocations are unnecessary, which indicates opportunities for further improvement of the generic ESVS proposed in Section 6.3.

Meanwhile, the generic ESVS again significantly reduces the extra time overhead compared to the fixed-step ESVS, by comparing ***Esc*-B** and ***Esc*-I** with ***Esc*-B (FixStep)** and ***Esc*-I (FixStep)**, respectively. Moreover, like in *two-phase greedy search*, the relationship between the extra time overhead of ***Esc*-B** and ***Esc*-I** is inconclusive. In general, each invocation of early-stopping verification using ***Esc*-B** is less expensive than using ***Esc*-I**, because considering index interactions requires more computation. However, since ***Esc*-I** improves the upper bound $\eta_U(W, C_B^*)$, it can trigger early-stopping sooner, which leads to fewer invocations of early-stopping verification. Therefore, the overall extra time overhead of ***Esc*-I** can be smaller than that of ***Esc*-B**, as showcased in Figure 12(a) for **TPC-H**. On the other hand, the overall extra time overhead of ***Esc*-I** is considerably larger than that of ***Esc*-B** for the workload **Real-D**, as evidenced by Figure 13(a). Regarding the savings on end-to-end tuning time, for **TPC-H** the savings are similar to the corresponding savings on what-if calls, as Figure 12(c) shows; for **Real-D** the savings are similar when ***Esc*-B** is used but are vanished when ***Esc*-I** is used due to its much higher computation overhead.

7.3.2 Improvement Loss. Like in *two-phase greedy search*, we see almost no violation of the improvement-loss threshold ϵ when early-stopping is triggered for *MCTS*. Moreover, the actual improvement loss is typically much lower than the threshold ϵ.

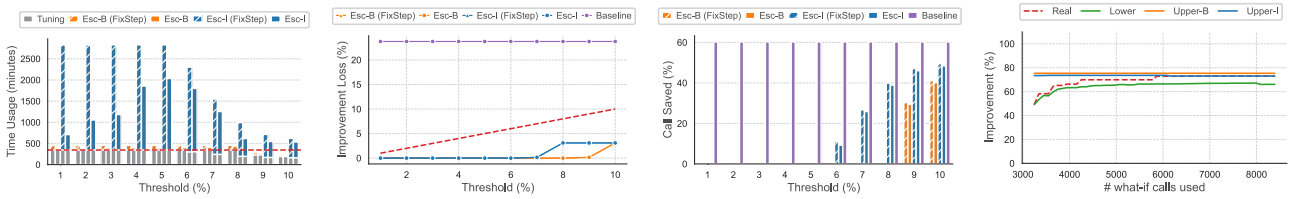
7.3.3 Savings on What-If Calls. The (percentage) savings on the number of what-if calls again vary across the workloads tested. For example, on **TPC-H** we can save 60% what-if calls by using ***Esc*-I** when the improvement-loss threshold ϵ is set to 5%, as shown in Figure 12(c). The actual improvement loss when early-stopping is triggered, however, is less than 2% instead of the 5% threshold, based on Figure 12(b). For **Real-D** we can only start saving on what-if calls with $\epsilon > 5\%$, though we can save up to 40% what-if calls when setting $\epsilon = 10\%$ and using ***Esc*-B**, as Figure 13(c) indicates. Note that, although we can save up to 50% what-if calls by using ***Esc*-I**, its extra time overhead is prohibitively high based on Figure 13(a), while the extra time overhead of using ***Esc*-B** is significantly lower than the overall index tuning time. Moreover, a larger threshold ϵ typically leads to larger savings on the what-if calls, as it is easier for the gap between the lower and upper bounds to meet the threshold.

7.3.4 Comparison with Baseline. Compared to *Esc*, the baseline approach that simply stops tuning after observing 30% improvement again can suffer from significant improvement loss. For example, as Figure 13(b) shows, the improvement loss of the baseline on **Real-D**



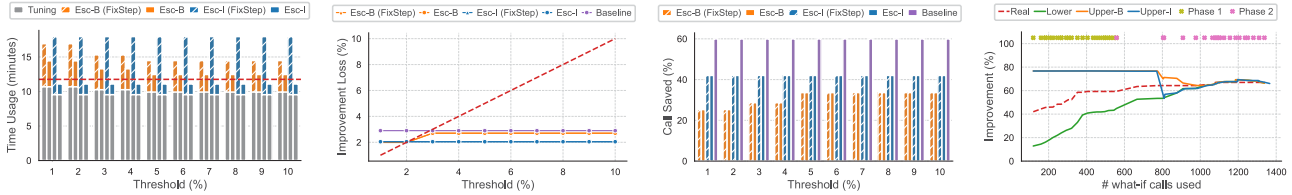
(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 12: MCTS, TPC-H, $K = 20$, $B = 20k$.



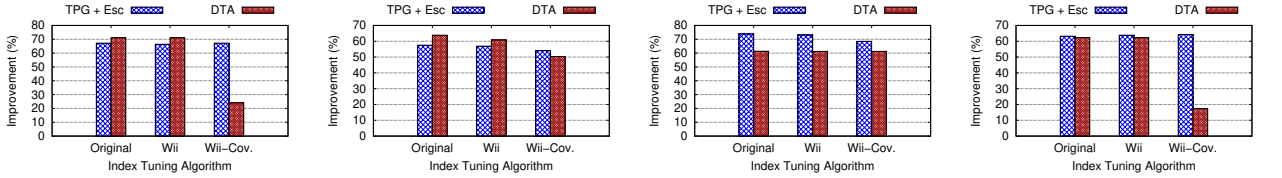
(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 13: MCTS, Real-D, $K = 20$, $B = 20k$.



(a) Time Overhead (b) Improvement Loss (c) What-If Call Savings (d) Learning Curve

Figure 14: Two-phase greedy search (with Wii-Coverage), Real-M, $K = 20$, $B = 20k$.



(a) **JOB** (b) **TPC-DS** (c) **Real-D** (d) **Real-M**

Figure 15: Comparison of two-phase greedy (TPG) search with *Esc* (without or with what-if call interception) against DTA.

is around 25%, whereas *Esc* has almost no loss. One could argue that having a threshold different than the 30% used may make a difference; however, choosing an appropriate threshold *upfront* for the baseline approach is itself a challenging problem.

7.4 What-If Call Interception

We have observed several cases where early-stopping offers little or no benefit, e.g., when running *two-phase greedy search* on top of **Real-M**, or when running *MCTS* on top of **TPC-DS** and **Real-M**, as shown in the full version [42]. The main reason for this inefficacy is the slow convergence of the gap between the lower and upper bounds used for triggering early-stopping. This phenomenon can be alleviated by using Wii, the what-if call interception mechanism developed in [43], which skips *inessential* what-if calls whose what-if costs are close to their derived costs.

For example, the heuristic ESVS in Section 6.1 only invokes early-stopping verification when *two-phase greedy search* enters Phase 2, when the upper bound is expected to drop sharply. With Wii integrated into *two-phase greedy search*, it can enter Phase 2 faster by skipping inessential what-if calls in Phase 1. As a result, we can expect *Esc* to be more effective for Wii-enhanced *two-phase greedy search*. To demonstrate this, we present the corresponding results for **Real-M** in Figure 14 using the Wii-enhanced *two-phase greedy search* with the coverage-based refinement. We observe that the savings on the number of what-if calls can further increase to 30% (using ***Esc*-B**) and 40% (using ***Esc*-I**), as Figure 14(c) presents.

Remarks. While Wii can often significantly bring down the number of what-if calls, this is a side effect that is not by design. Indeed, the goal of Wii is only to skip inessential what-if calls. Nevertheless, it does reduce the number of what-if calls that need to be made—if this number is smaller than the given budget we will see a (sometimes significant) drop on the total number of what-if calls made. Therefore, the contributions of early stopping and Wii in terms of reducing what-if calls are orthogonal and should not be directly compared. That is, there are cases where Wii can and cannot reduce the number of what-if calls while early stopping can make similar (e.g., 20% to 40%) reductions.

7.5 Comparison with DTA

To understand the overall benefit of budget-aware index tuning with *Esc* enabled, when compared to *other* index tuning algorithms, we further compare *two-phase greedy search* with *Esc* (TPG-*Esc*) against DTA, which employs *anytime* index tuning techniques [6] that can achieve state-of-the-art tuning performance [20]. In our

evaluation, we set the threshold of improvement loss $\epsilon = 5\%$. We measure the corresponding time spent by TPG-*Esc* and use that as the tuning time allowed for DTA [1], for a fair comparison.

Figure 15 presents the results. We omit the results on **TPC-H** as TPG-*Esc* and DTA achieve the same 79% improvement. We have the following observations on the other workloads. On **JOB**, TPG-*Esc* significantly outperforms DTA when Wii-coverage is enabled (67% by TPG-*Esc* vs. 24% by DTA). On **TPC-DS**, TPG-*Esc* and DTA perform similarly. On **Real-D**, TPG-*Esc* outperforms DTA by around 10%. On **Real-M**, TPG-*Esc* significantly outperforms DTA, again when Wii-coverage is enabled (64% by TPG-*Esc* vs. 17% by DTA). Overall, we observe that TPG-*Esc* either performs similarly to DTA or outperforms DTA by a noticeable margin in terms of percentage improvement, within the same amount of tuning time. Note that DTA leverages additional optimizations (e.g., "table subset" selection [2, 6], index merging [9], prioritized index selection [6], etc.) that we did not implement for TPG-*Esc*. On the other hand, it remains interesting to see the further improvement on DTA by integrating *Esc*, which is beyond the scope of this paper.

7.6 Discussion and Future Work

Violation of Improvement Loss. Violation is very rare based on our evaluation results, but it can happen if the assumptions about the what-if cost function, i.e., monotonicitiy and submodularity, are invalid. In such situations, the lower and upper bounds derived for the workload-level what-if cost are also invalid and therefore can mislead the early-stopping checker. One possible solution is then to validate the assumptions of monotonicity and submodularity while checking for early stopping. If validation fails frequently, then we will have lower confidence on the validity of the bounds and thus we can stop running the early-stopping checker to avoid potential violation on the promised improvement loss.

Hard Cases. As an example, the **TPC-DS** results in Figure 10 represent a difficult case for *Esc* when applied to *two-phase greedy search*. From Table 2, we observe a large search space for *two-phase greedy search* over **TPC-DS** with 848 candidate indexes. Moreover, the workload size of **TPC-DS** with 99 queries is also considerably larger than the other workloads in Table 2. As a result, the heuristic early-stopping verification scheme designed for two-phase greedy search (Section 6.1) works less effectively, because verification will not be invoked until entering the second phase of greedy search. Lots of what-if calls have been made in the first phase as well as the first step of the second phase, before the bounds start converging sharply. To improve on this case, we have to make the bounds converge earlier, which is challenging given the conservative nature of the bounds. We therefore leave this for future work.

8 RELATED WORK

Cost-based Index Tuning. Offline index tuning has been extensively studied in the literature (e.g., [5–7, 11, 18, 20, 32, 41, 44, 51]). Early work focused on index configuration enumeration algorithms, including, e.g., *Drop* [44], *AutoAdmin* [7], *DTA* [6], *DB2Advisor* [41], *Relaxation* [5], *CoPhy* [11], *Dexter* [18], and *Extend* [32]. We refer the readers to the recent benchmark studies [20, 56] for more details and performance comparisons of these solutions. More recent work has been focusing on addressing scalability issues of index tuning when dealing with large and complex workloads (e.g., [4, 37, 39, 43, 51, 54]) and query performance regressions when the recommended indexes are actually deployed (e.g., [12, 13, 35, 46, 55]). The latter essentially addresses the problem of modeling query execution cost in the context of index tuning, and there has been lots of work devoted to this problem (e.g., [3, 16, 17, 23–25, 27, 36, 40, 47–50, 52]). There has also been recent work on *online* index tuning with a focus of applying deep learning and reinforcement learning technologies (e.g. [21, 28, 29, 34]). Online index tuning assumes a continuous workload model where queries are observed in a streaming manner, which is different from offline index tuning that assumes all queries have been observed before index tuning starts.

Learning Curve and Early Stopping. Our notion of index tuning curve is akin to the term "learning curve" in the machine learning (ML) literature, which is used to characterize the performance of an iterative ML algorithm as a function of its training time or number of iterations [14, 19]. It is a popular tool for visualizing the concept of *overfitting*: although the performance of the ML model on the training dataset improves over time, its performance on the test dataset often degrades eventually. The study of learning curve has led to *early stopping* as a form of regularization used to avoid overfitting when training an ML model with an iterative method such as *gradient descent* [30, 31, 53]. Early-stopping in budget-aware index tuning, however, is different, with the goal of saving what-if calls instead of improving index quality, though the generic early-stopping verification scheme developed in Section 6.3 relies on the convexity/concavity properties of the index tuning curve.

Index Interaction. Some early work (e.g. [10, 15, 45]) has noted down the importance of modeling index interactions. A more systematic study of index interaction was performed by Schnaitter et al. [33], and our definition of index interaction presented in Section 5.1 can be viewed as a simplified case of the definition proposed in that work. Here, we are only concerned with the interaction between the next index to be selected and the indexes that have been selected in the simulated greedy search outlined by Procedure 2. In contrast, the previous work [33] aims to quantify any pairwise index interaction within a given configuration, with respect to the presence of all other indexes within the same configuration. To compute the index interaction so defined, one then needs to enumerate *all possible subsets* of the configuration, which is computationally much more expensive. Since we need a rough but efficient way of quantifying index interaction, we do not pursue the definition proposed by [33] due to its computational complexity.

9 CONCLUSION

We have presented *Esc*, an early-stopping checker for budget-aware index tuning. It extends call-level lower and upper bounds of what-if cost to develop workload-level improvement bounds that converge efficiently as index tuning proceeds. It further adopts a generic early-stopping verification scheme that exploits the convexity/concavity properties of the index tuning curve to skip unnecessary invocations of early-stopping verification. Evaluation on top of both industrial benchmarks and real customer workloads demonstrates that *Esc* can effectively terminate index tuning with improvement loss below the user-specified threshold while at the same time significantly reduce the amount of what-if calls made for index tuning. Moreover, the extra computation time introduced by early-stopping verification is negligible compared to the overall index tuning time.

REFERENCES

[1] 2023. DTA utility. https://docs.microsoft.com/en-us/sql/tools/dta/dta-utility?view=sql-server-ver15.

[2] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. 2000. Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB*. 496–505.

[3] Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. 2012. Learning-based Query Performance Modeling and Prediction. In *ICDE*. 390–401.

[4] Matteo Brucato, Tarique Siddiqui, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. 2024. Wred: Workload Reduction for Scalable Index Tuning. *Proc. ACM Manag. Data* 2, 1, Article 50 (2024), 26 pages.

[5] Nicolas Bruno and Surajit Chaudhuri. 2005. Automatic Physical Database Tuning: A Relaxation-based Approach. In *SIGMOD*. 227–238.

[6] Surajit Chaudhuri and Vivek Narasayya. 2020. Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server.

[7] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *VLDB*. 146–155.

[8] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *SIGMOD*. 367–378.

[9] Surajit Chaudhuri and Vivek R. Narasayya. 1999. Index Merging. In *ICDE*.

[10] Sunil Choenni, Henk M. Blanken, and Thiel Chang. 1993. On the Selection of Secondary Indices in Relational Databases. *Data Knowl. Eng.* 11, 3 (1993).

[11] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *Proc. VLDB Endow.* 4, 6 (2011), 362–372.

[12] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *SIGMOD*. 1241–1258.

[13] Bailu Ding, Sudipto Das, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2018. Plan Stitch: Harnessing the Best of Many Plans. *Proc. VLDB Endow.* 11, 10 (2018), 1123–1136.

[14] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In *IJCAI*. 3460–3468.

[15] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. 1988. Physical Database Design for Relational Databases. *ACM Trans. Database Syst.* 13, 1 (1988).

[16] Archana Ganapathi, Harumi A. Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael I. Jordan, and David A. Patterson. 2009. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *ICDE*.

[17] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *Proc. VLDB Endow.* 15, 11 (2022), 2361–2374.

[18] Andrew Kane. 2017. Introducing Dexter, the Automatic Indexer for Postgres. https://medium.com/@ankane/introducing-dexter-the-automatic-indexer-for-postgres-5f8fa8b28f27.

[19] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2017. Learning Curve Prediction with Bayesian Neural Networks. In *ICLR*.

[20] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, and Rainer Schlosser. 2020. Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms. *Proc. VLDB Endow.* 13, 11 (2020), 2382–2395.

[21] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *CIKM*. 2105–2108.

[22] Viktor Leis. 2015. Join Order Benchmark. https://github.com/gregrahn/join-order-benchmark.

[23] Jiexing Li, Arnd Christian König, Vivek R. Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. *Proc. VLDB Endow.* 5, 11 (2012), 1555–1566.

[24] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718.

[25] Ryan C. Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (2019), 1733–1746.

[26] Stratos Papadomanolakis, Debabrata Dash, and Anastassia Ailamaki. 2007. Efficient Use of the Query Optimizer for Automated Database Design. ACM.

[27] Debjyoti Paul, Jie Cao, Feifei Li, and Vivek Srikumar. 2021. Database Workload Characterization with Query Plan Encoders. *Proc. VLDB Endow.* 15, 4 (2021), 923–935.

[28] R. Malinga Perera, Bastian Oetomo, Benjamin I. P. Rubinstein, and Renata Borovica-Gajic. 2021. DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees. In *ICDE*. IEEE, 600–611.

[29] R. Malinga Perera, Bastian Oetomo, Benjamin I. P. Rubinstein, and Renata Borovica-Gajic. 2022. HMAB: Self-Driving Hierarchy of Bandits for Integrated Physical Database Design Tuning. *Proc. VLDB Endow.* 16, 2 (2022), 216–229.

[30] Lutz Prechelt. 2012. Early Stopping — But When? *Neural Networks: Tricks of the Trade: Second Edition* (2012), 53–67.

[31] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. 2014. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *J. Mach. Learn. Res.* 15, 1 (2014), 335–366.

[32] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *ICDE*. 1238–1249.

[33] Karl Schnaitter, Neoklis Polyzotis, and Lise Getoor. 2009. Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. *Proc. VLDB Endow.* 2, 1 (2009), 1234–1245.

[34] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration using Deep Reinforcement Learning. *CoRR* abs/1801.05643 (2018).

[35] Jiachen Shi, Gao Cong, and Xiaoli Li. 2022. Learned Index Benefits: Machine Learning Based Index Performance Estimation. *Proc. VLDB Endow.* 15, 13 (2022), 3950–3962.

[36] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings. In *SIGMOD*. ACM, 99–113.

[37] Tarique Siddiqui, Saehan Jo, Wentao Wu, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2022. ISUM: Efficiently Compressing Large and Complex Workloads for Scalable Index Tuning. In *SIGMOD*. ACM, 660–673.

[38] Tarique Siddiqui and Wentao Wu. 2023. ML-Powered Index Tuning: An Overview of Recent Progress and Open Challenges. *SIGMOD Rec.* 52, 4 (2023), 19–30.

[39] Tarique Siddiqui, Wentao Wu, Vivek R. Narasayya, and Surajit Chaudhuri. 2022. DISTILL: Low-Overhead Data-Driven Techniques for Filtering and Costing Indexes for Scalable Index Tuning. *Proc. VLDB Endow.* 15, 10 (2022), 2019–2031.

[40] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proc. VLDB Endow.* 13, 3 (2019), 307–319.

[41] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *ICDE*. 101–110.

[42] Xiaoying Wang, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. 2024. *Esc: An Early-Stopping Checker for Budget-aware Index Tuning (Extended Version)*. Technical Report. Microsoft Research. https://www.microsoft.com/en-us/research/people/wentwu/publications/

[43] Xiaoying Wang, Wentao Wu, Chi Wang, Vivek Narasayya, and Surajit Chaudhuri. 2024. Wii: Dynamic Budget Reallocation In Index Tuning. *Proc. ACM Manag. Data* 2, 3, Article 182 (2024), 26 pages.

[44] Kyu-Young Whang. 1985. Index Selection in Relational Databases. In *Foundations of Data Organization*. 487–500.

[45] Kyu-Young Whang, Gio Wiederhold, and Daniel Sagalowicz. 1981. Separability - An Approach to Physical Data Base Design. In *VLDB*. 320–332.

[46] Wentao Wu. 2025. Hybrid Cost Modeling for Reducing Query Performance Regression in Index Tuning. *IEEE Trans. Knowl. Data Eng.* 37, 1 (2025), 379–391.

[47] Wentao Wu, Yun Chi, Hakan Hacigümüs, and Jeffrey F. Naughton. 2013. Towards Predicting Query Execution Time for Concurrent and Dynamic Database Workloads. *Proc. VLDB Endow.* 6, 10 (2013), 925–936.

[48] Wentao Wu, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton. 2013. Predicting query execution time: Are optimizer cost models really unusable?. In *ICDE*. 1081–1092.

[49] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *SIGMOD*. ACM, 1721–1736.

[50] Wentao Wu and Chi Wang. 2024. Budget-aware Query Tuning: An AutoML Perspective. *SIGMOD Rec.* 53, 3 (2024), 20–26.

[51] Wentao Wu, Chi Wang, Tarique Siddiqui, Junxiong Wang, Vivek R. Narasayya, Surajit Chaudhuri, and Philip A. Bernstein. 2022. Budget-aware Index Tuning with Reinforcement Learning. In *SIGMOD*. ACM, 1528–1541.

[52] Wentao Wu, Xi Wu, Hakan Hacigümüs, and Jeffrey F. Naughton. 2014. Uncertainty Aware Query Execution Time Prediction. *Proc. VLDB Endow.* 7, 14 (2014), 1857–1868.

[53] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation* 26, 2 (2007), 289–315.

[54] Tao Yu, Zhaonian Zou, Weihua Sun, and Yu Yan. 2024. Refactoring Index Tuning Process with Benefit Estimation. *Proc. VLDB Endow.* 17, 7 (2024), 1528–1541.

[55] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670.

[56] Wei Zhou, Chen Lin, Xuanhe Zhou, and Guoliang Li. 2024. Breaking It Down: An In-depth Study of Index Advisors. *Proc. VLDB Endow.* 17, 10 (2024), 2405–2418.