Parameter Tuning in Personal Search Systems

Suming J. Chen, Xuanhui Wang, Zhen Qin, Donald Metzler
Google
Mountain View, California, USA
{suming,xuanhui,zhenqin,metzler}@google.com

ABSTRACT

The effectiveness of information retrieval systems is heavily dependent on how various parameters are tuned. One option to find these parameters is to run multiple online experiments using a parameter sweep approach in order to optimize the search system. There are multiple downsides of this approach, including the fact that it may lead to a poor experience for users. Another option is to do offline evaluation, which can act as a safeguard against potential quality issues. Offline evaluation requires a validation set of data that can be benchmarked against different parameter settings. However, for search over personal corpora, e.g. email and file search, it is impractical and often impossible to get a complete representative validation set due to the inability to save raw queries and document information. In this work, we show how to do offline parameter tuning with only a partial validation set. In addition, we demonstrate how to do parameter tuning in situations when we have complete knowledge of the internal implementation of the search system (white-box tuning), as well as situations where we have only partial knowledge (grey-box tuning). The resulting method provides a way of performing offline parameter tuning in a privacy-preserving manner. We demonstrate the effectiveness of the proposed approach by reporting the results from search ranking experiments performed on two large-scale personal search systems.

CCS CONCEPTS

• **Information systems → Evaluation of retrieval results**.

KEYWORDS

parameter tuning, evaluation, black-box tuning, personal search

ACM Reference Format:
Suming J. Chen, Xuanhui Wang, Zhen Qin, Donald Metzler. 2020. Parameter Tuning in Personal Search Systems. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371820

1 INTRODUCTION

Modern search engines usually have multiple processing stages in their systems. A typical flow has a retrieval stage followed with a ranking stage, where the retrieval stage selects N documents and passes them to the ranking stage. The ranking stage often employs a machine learning model based on query, document, and user features. In this work, we present a parameter tuning approach which can be used for either stage. This approach is especially impactful for the retrieval stage – while a large amount of work focuses on optimizing the parameters of the ranking stage, relatively little work [3, 8] covers parameter tuning at the retrieval stage.

Whereas retrieval work in the literature primarily studies approaches such as BM25 [26], search engines in industry have an entirely different set of challenges. Commercial web search engines have been developed for decades by many engineers. The various search system components quickly become complicated and typically have numerous parameters to be tuned. As [7, 13] note that even small variations in parameters of information retrieval systems lead to large variations in retrieval effectiveness, it motivates thoroughly tuning parameters. Another motivation for this work comes from infrastructure reuse in commercial settings, which leads to the same retrieval system being used for multiple applications and not tuned optimally for each one.

Given that an industrial search system can be very complex, a natural and common approach for parameter tuning is to treat the entire system as a black box and run A/B experiments with either some sort of parameter sweep (e.g. grid search, coordinate ascent) [22] or black-box optimization [17] over various parameters in a guess-and-check manner. The drawback to this is that 1) the user experience can be degraded due to exposure to poor quality results, 2) separate online experiments are required for each test set of parameters, and 3) the tuning process can take months.

Offline tuning using a validation set is another option [21]. For instance, as the retrieval component of a search system usually works with query strings and document contents directly, offline tuning in this setting involves collecting raw queries and historical click data and simulating how retrieval effectiveness changes when utilizing different parameters. However, in the *personal search* setting [2, 5, 12, 28], where both queries and documents are private, building a representative validation set is a challenge, as it involves either relying upon donated data (which can be limited and biased) or utilizing a validation set that only contains *partial* information (where any sensitive raw data is not logged). In this paper, we introduce a methodology for utilizing a partial validation set to do parameter tuning for the personal search setting.

Compounding the problem of performing parameter tuning with only a partial validation set is the complexity and uncertainty of the underlying retrieval system. As mentioned above, retrieval and ranking components in commercial systems can become quite complicated, especially over the years as new features continue to be added. Because of this, the inner workings of certain components of the search system can be treated as unknown or hidden, and can effectively be thought of as a *grey-box* system [24] (as opposed to a *white-box* system where the internals are completely known and a *black-box* system where nothing is known). With a partial validation set and uncertainty about the system, it becomes difficult to simulate offline how the results would have been ordered if different parameters had been chosen. The more uncertainty there is about the system, the fewer approaches there are for performing parameter tuning.

In this paper, we demonstrate how a search system can be formulated as either a white-box, grey-box, or a black-box system. We then propose methods for doing offline white-box and grey-box parameter tuning for personal search systems with a privacy-preserving validation set. For grey-box parameter tuning, we show how to exploit the structure of grey-box systems to allow for 1) inferring hidden components of the system, 2) finding suitable parameters to tune, and 3) providing approximate offline estimates of good parameter values that translate into online results. We demonstrate empirically that our method leads to significant increases in search quality metrics for two production systems serving millions of queries a day, requiring far fewer live experiments to be run than alternative methods.

The structure of this paper is as follows. We first discuss related work in parameter tuning and offline evaluation in Section 2. We then show how a search system can be formulated as either a white-box, grey-box, or black-box system in Section 3. In Section 4 we cover methods to do offline grey-box and white-box parameter tuning. We then discuss our empirical evaluation and results in Section 5, where we show that our efforts to tune parameters led to significant increases of search quality. In addition, we demonstrate empirically that our proposed method indeed shrinks the space of possible parameters that need to be tuned. We then conclude the paper with a recap and discussion of future work.

2 RELATED WORK

[7, 13] discuss the importance of parameter tuning, and [27] gives more insight into the difficulty of parameter management. Since new features are often added to search systems to improve the performance of retrieval functions, these features often increase the number of parameters which makes it more challenging to find optimal parameter values.

The majority of work regarding parameter tuning is for *black-box* systems, where no knowledge of the internal implementation is known. For these approaches, given an unknown function f, we are allowed to evaluate $f(x)$ for any value $x \in X$ in an effort to maximize (or minimize) $f(x)$. Typical approaches include Bayesian optimization [11, 17] as well as hybrid methods using random search alongside multi-armed bandit techniques [19]. In particular, [17] discusses how Bayesian optimization can be used for optimizing retrieval systems. These approaches generally assume some prior belief over f and draw samples to get a posterior that better approximates f. Other Bayesian optimization work consists of that of [15], which remarks on the importance of determining the impactfulness of different hyperparameters in order to reduce the exploration needed and speed up the optimization process. Bayesian optimization techniques differ from our work as our work assumes some (though incomplete) knowledge about the function f is known.

There has been significant recent work in how parameter tuning can be done better by leveraging knowledge of the internal implementation [4, 16, 20]. If we have *full* knowledge of the internals of a search system, we would have a *white-box* system. If we have *partial* knowledge, then we would consider it a *grey-box* system [24].

For white-box parameter tuning, [16] investigate tuning the parameters of existing programs. They assume full knowledge of program stages is available and provide a library where the user can have flexible access to internal program states in order to specify how tuning can be done (e.g. what range of values should be explored and how many samples to generate). Their approach leverages independence between computation stages to reduce the search space of parameters. They demonstrate that their approach beats general black box tuning. In the field of automated algorithm design, [1] show that using a white box approach and exploiting internal knowledge of algorithm evaluation functions allows for faster optimization.

For grey-box systems, [20] uses system-level monitoring information in conjunction with standard hill climbing algorithms to significantly improve MapReduce application performance. [4] demonstrate benefits of using automated parameter tuning in optimizing big-data streaming applications. In their work, they transform their multi-objective optimization function into a single-objective optimization problem and show that their rule-based approach of incorporating prior knowledge for parameter selection allows them to converge significantly faster than standard hill-climbing algorithms used for typical black box problems. In our work, we demonstrate how to leverage partial knowledge of the system to improve parameter tuning efforts.

Offline evaluation is critical for production search systems to sanity check any new experiments and prevent system behavior that may be detrimental to the users' experience. [14] use previously collected click data in order to perform interleaved comparison methods between various rankers and find that it can be less expensive than running live interleaving experiments. [18] provides a method of doing offline evaluation of different contextual bandit based article recommendations, where a primary motivation is to not hurt user experience by exposing potentially poor-quality algorithms.

3 PROBLEM FORMULATION

As stated in the introduction, one option of doing offline evaluation of search systems is to use a validation set. Here, different queries and documents (including which documents were clicked) are collected. The validation set can be passed into the search system with different parameters in offline evaluations to learn which parameters work best. However, for *personal* search systems, it is typically not allowable to log the raw queries and personal document contents due to privacy considerations. To the best of our knowledge, offline parameter tuning in the personal search setting has not been explored in detail previously.

Even though we may not be able to log the raw query and document contents, we may still be able to log some privacy-aware non-sensitive signals. For instance, in search systems, there is typically some scoring system component that takes a query and document and then outputs a score after undergoing numerous stages of query and document parsing, spell checking, synonym expansion, and more [7]. There are intermediate computed scores (which we refer to as *subscores*) that are non-sensitive and can be logged (e.g. term frequency (tf), inverse document frequency (idf), document length normalization).[1] These subscores are the output of some operation within the scoring function and get piped further downstream to other functions that further transform and combine the scores. In the following section, we show how to construct a validation set, hereafter referred to as Q, that is not comprised of the traditional raw queries and raw documents, but instead consists of subscores. Subsequently, we demonstrate that Q can be used to tune both white-box search systems and grey-box search systems.

3.1 White-box Formulation

Given that scoring happens in different stages and numerous functions are called, we can formulate the general computation as a directed acyclic graph (DAG). To better define our formulation, we use the same terminology as is typically used in dealing with graphical models [9, 23]:

- $pa(x) \triangleq$ parents of x
- $ch(x) \triangleq$ children of x
- $fam(x) \triangleq x \cup pa(x)$

In our white-box formulation, the query and document are the roots of the DAG and the leaf is the outputted score. The various intermediate derived subscores $s_1, \dots, s_m$, scoring functions $f_1, \dots, f_n$, and parameters $p_1, \dots, p_o$ (controlling function computation) can be thought of as the internal nodes of the graph. Each edge of this DAG represents the flow of computation. Our formulation has the following properties:

(1) $pa(f_i) \in \{s_1, \dots, s_m, p_1, \dots, p_o\} \forall i$
(2) $pa(s_i) \in \{f_1, \dots, f_n\} \forall i$
(3) $pa(p_i) \in \varnothing$

Basically these properties enforce that the scoring function nodes are always separated from each other by exactly one layer of subscore nodes.

An example formulation can be seen in Figure 1. Here, we can see that even though the raw query s_q and document s_d are privacy-preserving and not available in our offline validation set (and as such, colored black), that subscores s_1, s_2, s_3, s_4 (which may refer to signals related to tf/idf score) and final score s_f can be logged in the validation set Q as they are not privacy-preserving fields (colored white). We can thus say that $type(s_q, s_d) = hidden$ to contrast with other visible nodes such as $\{s_1, s_2, s_3, s_4\}$, where $type(s_1, s_2, s_3, s_4) = visible$. Also, note that, as the internal system implementation is known, $\forall f_i \in \{f_1, \dots, f_n\}$, $type(f_i) = visible$.

A query q in the validation set Q then consists of all the *visible* subscores for query-document pairs. In this case, for the grey-box system in Figure 1, the per-query information includes 1) per-doc information of what the subscores $\{s_1, \dots, s_m, s_f\}$ were for each result document d and 2) which document was clicked. Table 1 shows an example of the validation information for an arbitrary query $q \in Q$. Notice that any information regarding s_q and s_d is entirely absent from Q, since they are privacy-preserving.

[1]In this paper, for simplicity we treat subscores as single float numbers, but in practice subscores can represent a wider range of types, e.g. string representation for an "expanded query" subscore.

The *key insight* here is that even if we only have a partial validation set Q that is missing s_q and s_d, we can still tune certain parameters given full knowledge of how the computation process is affected. For a white-box system, we can always tune any function f_i where $pa(f_i) \notin \{s_q, s_d\}$—in the depicted example, f_3, f_4, f_5 can be tuned offline since the exact inputs ($pa(f)$) and outputs are available in our validation set. We can then adjust p_4, p_5, p_6 to see if we can get a better ordering of the documents of each query session (i.e. if the average click position over the validation set can be decreased). With this white-box formulation, offline parameter tuning for personal search systems is straightforward if we "prune" off the parameters related to functions taking $\{s_q, s_d\}$ as input and focus on tuning the remaining parameters.

Example 1. We use a toy example to demonstrate how this tuning can work. Suppose we have the following definitions for components in Figure 1:

- $f_3(s_1, s_2|p_4) = s_1 + p_4 \cdot s_2$
- $f_4(s_2|p_5) = s_2^{p_5}$
- $f_5(s_3, s_4|p_6) = \frac{s_3}{s_4 \cdot p_6}$

For an arbitrary example query $q \in Q$, suppose that we have the following offline scores captured in Table 1. The documents are ordered by their final score s_f, which is computed according to the above function definitions.[2] As a common goal in information retrieval systems is to place relevant results at a higher rank [7], in this example having d_3 ranked at position 3 is not ideal. Even without knowing what s_q and s_d are offline, we can run a parameter sweep over p_4, p_5, p_6 in order to find a better ordering. One such ordering is achieved by setting $p_4 = 0.2, p_5 = 0, p_6 = 1$, resulting in final scores s'_f, leading to the ranking seen in Table 2.[3]

In a more realistic setting, we would want to fit our parameters according to the entire validation set Q, and given tunable parameters $\mathbf{p}$ would be looking to find the parameters to maximize the aggregate metrics:

$$\arg\max_{p \in \mathbf{p}} \frac{1}{|Q|} \sum_{q \in Q} M(q, s'_f(p)) \tag{1}$$

In the above equation, $s'_f(p)$ denotes what the per-document scores would have been had parameters p been used, whereas M represents any generic search metric such as click-through rate or mean-reciprocal rank. Note that instead of using the entire validation set Q to find parameters, we can use standard cross-validation parameter tuning techniques as used in [6, 16].

3.2 Black-box and Grey-box Formulation

The white-box example in Figure 1 contained an entirely transparent system implementation where all nodes in the graph are *visible* and only s_q and s_d are *hidden*. We can have hidden functions ($type(f_i) = hidden$) as well. If all functions are hidden we have a black-box system, as shown in Figure 2. This may be a system that is just a compiled binary (with source code missing), or a system

[2]Note that per the privacy restrictions we have discussed, s_q and s_d are not logged in the table.
[3]Note that these toy example only includes differentiable functions, so gradient-based optimization approaches can be used. However, in general the functions may not be differentiable.

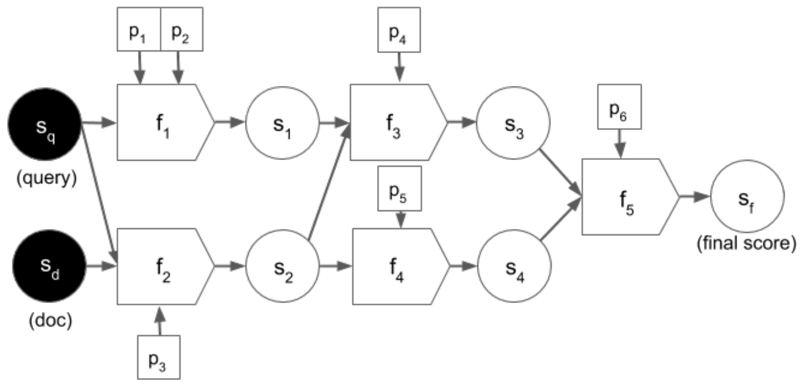

Figure 1: A DAG formulation of a scoring system where the internal implementation is entirely known (white-box). We see here how the parent nodes of a function $pa(f_i)$ are the inputs and the child node $ch(f_i)$ is the output of the function.

Table 1: Example document scores and subscores for an arbitrary query q that is passed through the scoring system defined in Figure 1 with $p_4 = p_5 = p_6 = 1$. The clicked document is highlighted in grey. A validation set Q would contain numerous instances of different query-document pairs.

doc	s_1	s_2	s_3	s_4	s_f
d_1	5	10	15	10	1.5
d_2	5	20	25	20	1.25
d_3	6	30	36	30	1.2
d_4	1	20	21	20	1.05

Table 2: Example document scores and subscores for the example given in Table 1 with $p_4 = 0.2, p_5 = 0, p_6 = 1$. The clicked document is highlighted in grey.

doc	s_1	s_2	s_3	s_4	s'_f
d_3	6	30	12	1	12
d_2	5	20	9	1	9
d_1	5	10	7	1	7
d_4	1	20	5	1	5

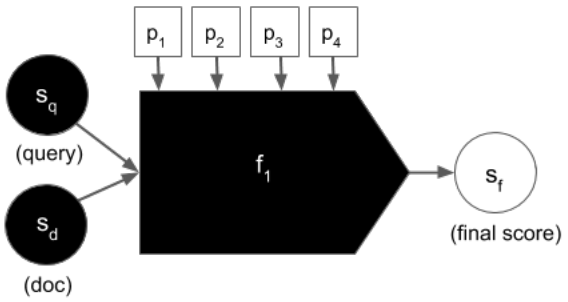

Figure 2: Black-box example.

that has gotten so complex and has so much technical debt that it is easier just to treat as a black-box. In this case, even though the parameter values and output scores for the system are known, the validation set Q is near-trivial as it only contains s_f as no internal subscores s_i are logged, and as such no offline evaluation at all can be done.[4]

However, often times we may have *some* knowledge of the internals of a system [4, 20]. An example of this is a search system where the general graph of computation is known, as in [16], but different scoring functions f_i and subscores s_i are unknown ($type(f_i) = type(s_i) = hidden$). For instance, we may have no idea how the query parsing library works. Or, we may have exact knowledge of how a BM25 score is produced but do not actually have the logging capability to retrieve that subscore. In this case, we can still formulate a grey-box model as a DAG, as seen in Figure 3, which serves as a running example. Some more practical examples:

- $\{s_q, s_d\} \rightarrow f_2 \rightarrow s_2$ represents sensitive inputs (e.g. raw query and document contents) passed to a function that computes a BM25 score that is logged and available in a validation set.
- $s_d \rightarrow f_3 \rightarrow s_3$ can represent a sensitive input (e.g. the document contents) passed to a document quality function that produces a quality score that is logged and available in a validation set.
- $s_q \rightarrow f_1 \rightarrow s_1$ can represent a sensitive input (e.g. raw query) passed to a synonym expansion library and outputting the expanded raw query (with added synonyms). In this case, the internal logic of library is unknown and the outputted expanded raw query is also hidden.
- $\{s_2, s_3\} \rightarrow f_6 \rightarrow s_6$ represents two visible inputs (e.g. query score and a document score) into a well known function. The output s_6 can represent some query/doc match numerical score that is not visible, but can be *inferred* since the inputs and function are visible. This is the key insight that allows Algorithm 2 in the next section to work.

[4]In this case, a common solution to doing performance tuning is to either do black-box tuning or build a reranking system on top and leave the complex system untouched.

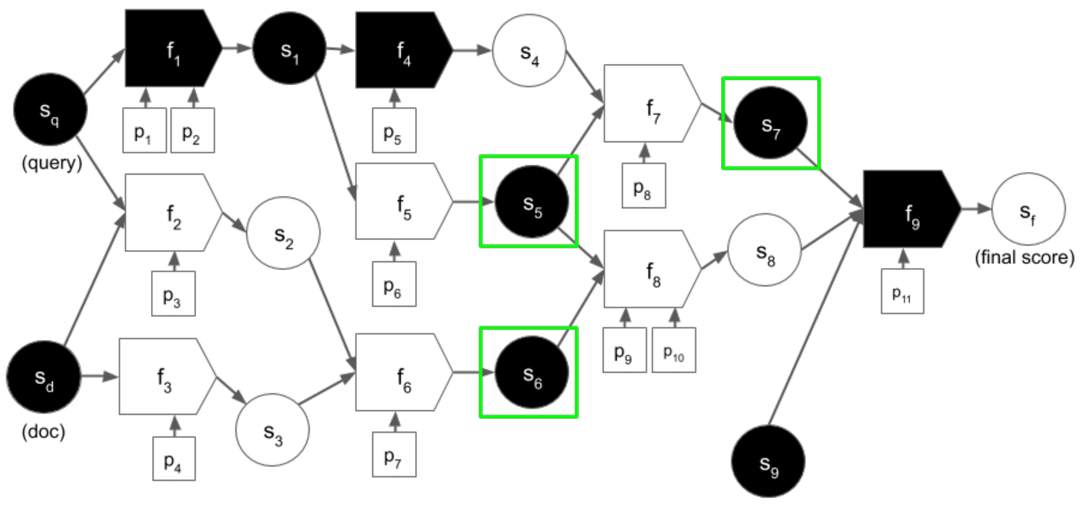

Figure 3: Grey-box example. Subscores bounded by green boxes, e.g. $\boxed{s_5}$, $\boxed{s_6}$, $\boxed{s_7}$, can be inferred after running Algorithm 2.

For the grey-box system shown in Figure 3, the partial validation set Q contains the query-document pairs of the visible scores, $\{s_2, s_3, s_4, s_8, s_f\}$. In the next section, we show that by formulating the computational process as a DAG, we can represent a rich set of data dependencies that enables us to make inferences about the scoring process [25] that allow us to 1) infer the values of some hidden subscores, 2) find promising parameters to tune, and 3) find optimal parameter values given the partial validation set Q.

4 METHODOLOGY

In this section, we start off by describing how the technique introduced in the last section for white-box tuning can be applied more generally in a grey-box setting. We then describe a technique which allows for inferring the values of the hidden nodes of the graph, and close the section with a general methodology that demonstrates how both techniques can work in conjunction.

4.1 Generalizing White-box Tuning

In the last section, we showed (via Example 1) how a white-box search system can be partially optimized by only tuning f_i where $pa(f_i) \notin \{s_q, s_d\}$. We see in Figure 3 that this approach cannot work due to the hidden nodes in a grey-box system that prevent us from computing how different parameter values affect s_f.

However, our previous insight of pruning off certain sections of the graph and only tuning a subset of the parameters can still be applied here. Though we cannot tune s_f, there may be other subscores that can be tuned. If there is some visible subscore s_i that has high correlation with the final score s_f then we can use s_i as a proxy for the final score. When trying out a new set of parameters, instead of ordering documents in Table 2 by s'_f (which is not computable given the hidden nodes), we can instead order documents by:

$$\frac{s_f \cdot s'_o(p)}{s_o} \tag{2}$$

Algorithm 1 `FindParameters`(G, x) — uses a BFS to find parameters to tune the subcomponent x. `visible` is a function that takes in a set of nodes and returns True only if the set of nodes is not empty and all nodes are *visible*.

```
1:  $p \leftarrow \{\}$                                ▷ The return parameters
2:  $Q \leftarrow \{\}$                                ▷ Initialize a queue
3:  $Q$.push($x$)
4:  while $Q$ is not empty do
5:      $v \leftarrow Q$.pop()
6:      if visible($fam(pa(v))$) then
7:          for $u$ in $pa(pa(v))$ do
8:              if $u \in \{s_1, \cdots, s_m\}$ then
9:                  $Q$.push($u$)
10:             else if $u \in \{p_1, \cdots, p_o\}$ then
11:                 $p \leftarrow p \cup u$
12: return $p$
```

where $s_o = \arg\max_{s_i} |corr(s_i, s_f)|$, the subscore with highest correlation to s_f (over the validation set Q), and $s'_o(p)$ denotes what the score of s_o would have been had parameters p been used. Note that the correlation we use is Pearson correlation coefficient:

$$corr(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

where $cov(X, Y)$ is the covariance between X and Y and σ_X and σ_Y are the standard deviations of X and Y, respectively.

The assumption we make here is that since s_o has high correlation with s_f, we can approximate what the final score would be given p by looking at the value of $s'_o(p)$. Note that there are other approaches we considered to approximate the final score (e.g. we could also consider the two subscores with highest correlation and tune all associated parameters based on them) but we went with the simplest approach as it was highly effective in practice.

We now introduce an approach for finding parameters for s_o. Algorithm 1 uses a breadth first search (BFS) approach, exploiting the graph structure and leveraging independence between computational stages to find any tunable parameters that can affect the computation for s_o. For our running example in Figure 3, we can run Algorithm 1 on one of s_2, s_3, s_4, s_8 (depending on correlation). Unfortunately, we find that since this algorithm depends on visible subscores and functions, it does not return any tunable parameters. This motivates our next contribution, which is a method for inferring hidden values to expand the set of parameters to tune.

4.2 Inferring Hidden Subscores

While it is true that hidden values are typically privacy-preserving or typically not logged due to complexity reasons, we can still do an on-the-fly computation to infer what a value would have been given some parameter changes. For example, in Figure 3 we have $s_6 = f_6(s_2, s_3, p_7)$. Even though s_6 is hidden, since both its parent function f_6 and the parent function's inputs are visible, we can infer the value of s_6. For our purposes, $type(s_6) = visible$, as we can substitute $f_6(s_2, s_3, p_7)$ in place of s_6 into the input of f_8. Note that this can work from the other end as well: given that $s_8 = f_8(s_5, s_6, p_9, p_{10})$, given the substitution of s_6 and that $type(s_8) = visible$, we can then construct an inverse f_8^{-1} to recover what the value of s_5 is: $s_5 = s_8 \cdot f_8^{-1}(s_6, p_9, p_{10})$.[5]

Algorithm 2 `InferSubscores`(G) — repeatedly traversing through the graph G of subscores to leverage visible subscores and functions in order to infer hidden subscore values. `visible` is defined the same as Algorithm 1.

```
1:  $G'$ = topological ordering of vertices in $G$
2:  $repeat \leftarrow true$                ▷ Whether or not to repeat procedure
3:  while $repeat == true$ do
4:      $repeat \leftarrow false$           ▷ Repeat only if variable inferred
5:      for $x$ in $G'$ do
6:          if $x \notin \{s_1, \cdots, s_m\}$ then
7:              continue
8:          if visible($x$) then
9:              continue
10:         if visible($fam(pa(x))$) then
11:             $type(x) \leftarrow visible$
12:             $repeat \leftarrow true$
13:         for $c$ in $ch(x)$ do
14:             if visible($fam(c) \setminus x \cup ch(c)$) then
15:                 $type(x) \leftarrow visible$
16:                 $repeat \leftarrow true$
```

Given the importance of having visible nodes for Algorithm 1, we introduce a simple inference algorithm, Algorithm 2, that can be run to demystify as much of the grey-box as possible. This algorithm essentially makes multiple passes through the DAG, utilizing the Markov blanket of a node x to infer whether or not $type(x)$ can be made $visible$. When the algorithm is able to infer a hidden value, the algorithm will loop again in case the newly inferred visible variable can be used to infer some other previously hidden value.

[5]This assumes the existence of a one-to-one inverse function ($\exists f_8^{-1}$), which we generally found to hold true for the visible functions in our domain.

For our running example in Figure 3, we find that upon running Algorithm 2, the following occurs:

(1) $type(s_6) \leftarrow visible$
(2) $type(s_5) \leftarrow visible$
(3) $type(s_7) \leftarrow visible$

In our example, 3 nodes $\boxed{s_5}$, $\boxed{s_6}$, $\boxed{s_7}$ are able to be transformed to visible and are bounded by green boxes in Figure 3. When subsequently running Algorithm 2, we are now able to find that 4 parameters p_7, p_8, p_9, p_{10} can now be used for tuning the values of s_6, s_7, s_8, as the full computation can be simulated offline.

4.3 Optimizing parameters in a grey-box setting

Given a DAG G and offline evaluation set of queries/documents Q, our approach for grey-box parameter tuning is then:

(1) `InferSubscores`(G)
(2) $o = \arg\max_i |corr(s_i, s_f)|$
(3) $\mathbf{p}$ = `FindParameters`(G, s_o)
(4) Return parameters that optimize the aggregate metrics:

$$\arg\max_{p \in \mathbf{p}} \frac{1}{|Q|} \sum_{q \in Q} M(q, \frac{s_f \cdot s'_o(p)}{s_o}) \tag{3}$$

where $s'_o(p)$ denotes what the score of s_o would have been had parameters p been used. For white-box parameter tuning, we would start at step 3 for $s_o = s_f$, and the above equation would simply reduce down to Equation 1. In the next section, we show how our approach works for real-world personal search systems, and how $\mathbf{p}$ can be robust in the sense that tuning additional parameters $\mathbf{p}' \in \{p_1, \ldots, p_o\} \setminus \mathbf{p}$ can be redundant to improving quality.

5 EVALUATION

We applied our approach to two personal search systems: GMail and Google Drive. Both GMail and Drive are large and complex systems operating over sensitive data. As such, our empirical evaluation only includes logged offline data that contains high-level metadata. For the partial validation set Q, we utilized around 1 million Gmail queries and 250 thousand Drive queries pulled from a 1 week period. Each query in the validation set resulted in a click. In this validation set, each query is associated with up to 6 emails and 5 documents, respectively. Given the proprietary nature of these search systems, we are unable to reveal the actual DAG structure. Instead, we demonstrate empirically that:

(1) Our approach is superior to a standard parameter sweep as well as state-of-the-art black-box optimization techniques, allowing us to find optimal parameters faster and without degrading the user search experience.
(2) Our approach is robust and any gains that we find for parameter tuning offline translate to the online setting in a correlated manner.
(3) Utilizing our approach of finding parameters can exploit the graphical structure of the grey-box in order to reduce the space of parameters that need to be explored.

5.1 Metrics

We use the following search metrics to evaluate the impact of our proposed method:

- ACP: Average click position, where a *decrease* is desirable.
- CTR: Click-through rate, i.e. the fraction of queries where a result was clicked.
- MRR: Mean reciprocal rank.

$$\frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{4}$$

The reciprocal rank is the multiplicative inverse of the rank of the clicked result: 1 for the first item, 0.5 for the second item, etc, and is 0 if there is no click. MRR is the mean of reciprocal ranks of all responses.

5.2 Experiment Results

5.2.1 Improvement over existing parameter tuning techniques. We demonstrate that leveraging internal knowledge of the system can allow for efficiently tuning the parameters. Since we only had partial knowledge of both GMail and Drive search systems, we used the approach described in Section 4.3 to find parameters to tune. We found parameters that improved search metrics offline and then ran live experiments. The search quality metrics for these experiments are shown in Table 3. We see that our approach led to tuning parameters that led to statistically significant metric increases for both GMail and Drive. Note that we saw a bigger increase for Drive than for GMail. This bigger increase corresponds well to offline numbers as well, which showed that Drive in general had more headroom for improvement in parameter tuning than GMail.

Our approach allows us to directly run live experiments with high confidence that any parameters found offline can be used in the online setting as well, thereby reducing the number of live experiments that need to be run and greatly eliminating the possibility of running experiments that hurt user experience. In contrast, any existing methodology that relies on black-box tuning requires multiple online experiments that may degrade quality. For instance, from past live experiments to improve quality, we observed that around 4% of parameter sweep trials resulted in a significant improvement in metrics. A large percentage of these trials resulted in worse quality results, hurting the user experience. With our approach, we were able to discriminate easily between various sets of parameters offline with Algorithm 1 and found that optimal set of parameters we discovered offline led to significant improvements in live experiments for both GMail and Drive.

We also note that the usefulness of our proposed methodology is contingent on having subscores that are of high correlation (> 0.75) to the final score. We found that using lower correlation subscores as a proxy led to finding sub-optimal parameters – these parameters could improve the offline metrics, but did not translate into improving online performance.[6]

In addition, we also demonstrate that our approach of grey-box tuning is superior to a state-of-the-art black-box tuning approach. We used Vizier [11], a black-box optimization toolkit designed specifically for parameter tuning. Vizier works by inputting in a feasible region of parameters. Vizier will then suggest parameter values to run a trial with. After a trial (e.g. live experiment) has been started, the feedback (e.g. live metrics) is used by Vizier with Gaussian Process Bandits [10] to suggest new parameters.

Table 3: Results of using our grey-box parameter tuning approach. Statistical significant numbers are bolded.

Scenario	ACP decrease	CTR gain	MRR gain	% queries affected
Gmail	**0.51**	-0.22	0.0	17.74
Drive	**1.27**	**2.60**	**3.62**	11.24

Table 4: Results of using a black-box optimization toolkit. Statistically significant numbers are bolded.

trial id	ACP decrease	CTR gain	MRR gain	% queries affected
trial1	-1.01	**-4.64**	**-5.06**	10.75
trial2	-0.47	0.25	0.31	11.22
trial3	-0.94	-3.59	-4.33	10.58
trial4	-1.07	-4.24	**-4.82**	12.31

We ran a Vizier study to see if we could improve upon the standard parameter sweep, but found that it still was much more inefficient compared to running simulations offline because 1) we had to wait a long period of time to collect feedback and 2) there were multiple experiments that led to significant negative results. Results can be see in Table 4. Note that we only ran a few trials since we stopped early due to lack of positive results. Though black box optimization can take many trials before converging, these results show the disadvantage of running live experiments and further motivates offline parameter tuning to reduce the number of live experiments that harmed quality.

5.2.2 Offline results are robust to the online setting. We report metrics for using our proposed parameter tuning approach, demonstrating that parameters leading to improvements offline in general lead to online improvements. To give us additional data points, instead of just taking the top correlated subscore (as is the proposed step in Equation 3) and tuning the associated parameters for just that one subscore, we ran Algorithm 1 for various other subscores to get multiple sets of parameters to tune.

With these different sets of parameters, we then used a standard parameter sweep in order to find the best performing offline parameter instantiation. The various instantiations of parameters were then used to start multiple experimental trials. Tables 5 and 6 respectively show the results of various Drive and Gmail experiments, where each trial in the table corresponds to a different set of parameters that were used.[7] We see that there is a clear link between any offline produced score and the online metrics. For both search systems, we also ran ablation experiments to check that parameters that led to poor offline gains also led to poor online gains as well, demonstrating that our approach can be used to safeguard against running poorly performing live experiments.

[6]An example of this is `trial7` in Table 5.

[7]Note that we did not experiment with combining different sets of parameters to see if experimental gains stack, which is out of the scope of this paper.

Table 5: Results of live Drive experiments contrasted with offline metrics. Each trial represents a set of different parameters that were used in a live experiment. Statistically significant numbers are bolded. We found that there were multiple sets of parameters that led to significant increases in metrics for live experimental trials. The table also shows the percentage of query traffic that was affected by the parameter changes to give a better sense of the total impact.

trial id	% offline gain	corr score	% queries covered	ACP decrease	CTR gain	MRR gain	% queries affected
trial1	0.64	0.93	98	**1.27**	**2.60**	**3.62**	11.24
trial2	0.53	0.93	90	-1.06	2.85	2.18	15.98
trial3	0.39	0.88	97	0.91	**1.41**	**2.41**	12.34
trial4	0.37	0.88	90	-0.41	0.85	0.56	25.73
trial5	0.32	0.65	99	-0.81	0.61	0.97	18.75
trial6	0.3	0.77	98	0.73	1.58	1.84	8.43
trial7	0.25	0.65	90	0.56	0.28	0.45	15.48
trial8	0.22	0.77	86	**3.2**	-3.15	-0.42	13.48
trial9	0.18	0.78	98	-0.01	1.09	1.06	25.48
trial10	0.18	0.88	90	0.88	0.41	1.21	16.43
trial11	0.17	0.93	91	-0.92	0.43	0.13	16.57
trial12	0.18	0.78	98	0.89	0.05	-0.14	12.14
trial13	-0.48	0.93	98	**-2.47**	-1.26	-1.26	17.48
trial14	-1.25	0.93	98	1.83	**-4.96**	**-4.00**	20.69

Table 6: Results of live Gmail experimental trials contrasted with offline metrics. Each trial represents a set of different parameters that were used in a live experiment. Statistically significant numbers are bolded. We found that there was less headroom both offline and online in improving GMail.

trial id	% offline gain	corr score	% queries covered	ACP decrease	CTR gain	MRR gain	% queries affected
trial1	0.26	0.92	98	**0.51**	-0.22	0.00	17.74
trial2	0.21	0.88	97	0.42	0.10	-0.53	17.22
trial3	0.17	0.65	99	0.10	-0.58	-0.68	21.86
trial4	0.12	0.77	97	0.32	-0.21	0.15	15.47
trial5	0.10	0.77	86	0.25	-0.38	-0.04	35.27
trial6	-1.12	0.92	98	**-0.66**	**-5.18**	**-4.99**	10.7
trial7	-1.58	0.92	98	**-1.95**	**-4.20**	**-2.69**	22.4

These experimental results suggest the offline gain is important, the degree of correlation of the score we are tuning (compared to the actual score) is quite important as well, as any of the statistically significant results obtained had relatively high correlation scores. Figure 4 visualizes the results of Tables 5 and 6 by plotting a weighted offline metric, the % offline gain scaled by the correlation, against the mean of the three online metrics normalized by the % of queries affected by the experiment. For GMail we found that there was less headroom for parameter tuning, whereas for Drive there was more headroom, as we were able to find multiple instantiations of parameters that led to both significant offline and online gain.

Lastly, for the offline evaluation sets we were using we were not always able to get a full reproduction of every query to document. As such, Tables 5 and 6 also show for each of these offline parameter tuning experiments how much of the offline evaluation set we were actually able to use. This appears to be slightly important, but not as important as the offline gain and correlation score.

5.2.3 Reducing the space of parameters that need to be explored.

[7] notes the increasing difficulty of tuning search systems as the

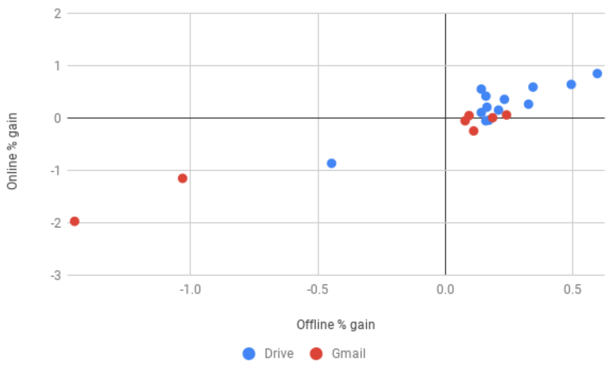

Figure 4: Relationship between weighted offline metric versus weighted online metric.

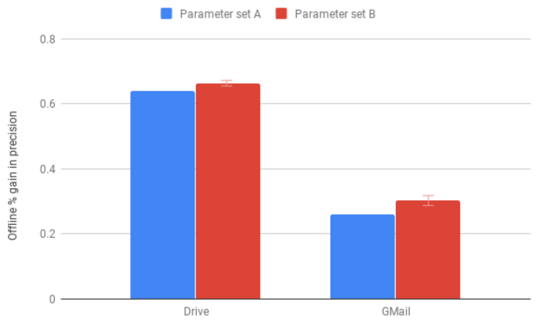

Figure 5: Performance of base set of parameters versus parameter sets double the size. Error bars for the randomly selected larger parameter sets are shown. We see that in this domain tuning the base set is sufficient and there is diminishing return in tuning additional parameters.

number of parameters increases. Even if we are doing offline evaluation, a parameter sweep over all possible configurations does not scale well. This combinatorial difficulty is the reason that [15] notes that it is necessary to determine important parameters and exploit the information to speed up the optimization problem.

We show that our approach to determine which the parameters should be tuned is fairly robust against tuning additional parameters. Our experiment methodology involves using Algorithm 1 to select parameters to tune, which results in k parameters being selected.[8] We treat these as the set of *baseline parameters* (**B**) to tune. We then repeatedly sample from other parameters in order to find k additional parameters to tune (**A**). Results are shown in Figure 5. There we can see that the baseline parameters **B** algorithm performs nearly as well as if a set including additional parameters **A** are tuned. Clearly, the majority of the performance gain in both settings can be tied to the original k parameters of **B**. These results are similar to that of the discovery of [15], who demonstrated that

[8]The value of k is deliberately omitted to protect proprietary information.

even in very high dimensional cases, performance variation is only dependent on a few parameters.

We see that there is more headroom for improvement in Drive versus GMail even if additional parameters are used, which confirms our earlier suspicions that GMail is already much more optimized than Drive is. We do see that tuning additional parameters (beyond the base k) in GMail is more beneficial than tuning more than k additional parameters in Drive, which motivates future work to explore tuning the parameters for more than just one subscore.

6 CONCLUSION AND FUTURE WORK

In this paper we introduce a novel approach for how parameter tuning can be done offline for personal search systems. We demonstrate that it is effective in tuning real-world production systems, requiring significantly fewer live experiments to be run and leading to significant gains in live metrics for Google Drive and Gmail. We expect that our presented formulation and methodology can be extended to not work for just search systems, but any form of complex system where 1) there is significant uncertainty in the inner-workings of the system and 2) user privacy needs to be taken into account, meaning that anonymized (and therefore diminished) validation sets. As future work, we plan to study how much we can improve on parameter tuning if we tune multiple components of the system at once and combine different sets of learned parameters to see how experimental gains would stack.

ACKNOWLEDGMENTS

We thank Michael Bendersky, Roberto Bayardo, and Arthur Choi for comments on improving the clarity and framing of this paper.

REFERENCES

[1] Steven Adriaensen and Ann Nowé. 2016. Towards a White Box Approach to Automated Algorithm Design. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press, 554–560. http://dl.acm.org/citation.cfm?id=3060621.3060699

[2] Qingyao Ai, Susan T Dumais, Nick Craswell, and Dan Liebling. 2017. Characterizing email search using large-scale behavioral logs and surveys. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1511–1520.

[3] Nima Asadi and Jimmy Lin. 2013. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 997–1000.

[4] Muhammad Bilal and Marco Canini. 2017. Towards Automatic Parameter Tuning of Stream Processing Systems. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. ACM, New York, NY, USA, 189–200. https://doi.org/10.1145/3127479.3127492

[5] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by time or by relevance?: Revisiting email search. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 283–292.

[6] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. 2002. Choosing multiple parameters for support vector machines. *Machine learning* 46, 1-3 (2002), 131–159.

[7] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading.

[8] Van Dang, Michael Bendersky, and W Bruce Croft. 2013. Two-stage learning to rank for information retrieval. In *European Conference on Information Retrieval*. Springer, 423–434.

[9] Adnan Darwiche. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

[10] Thomas Desautels, Andreas Krause, and Joel W Burdick. 2014. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *The Journal of Machine Learning Research* 15, 1 (2014), 3873–3923.

[11] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1487–1495. https://doi.org/10.1145/3097983.3098043

[12] Jai Gupta, Zhen Qin, Michael Bendersky, and Donald Metzler. 2019. Personalized Online Spell Correction for Personal Search. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 2785–2791. https://doi.org/10.1145/3308558.3313706

[13] Ben HE and Iadh Ounis. 2003. A Study of Parameter Tuning for Term Frequency Normalization. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM '03)*. ACM, New York, NY, USA, 10–16. https://doi.org/10.1145/956863.956867

[14] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2012. Estimating interleaved comparison outcomes from historical click data. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 1779–1783.

[15] Holger Hoos, UBC Ca, and Kevin Leyton-Brown. 2014. An efficient approach for assessing hyperparameter importance. In *International Conference on Machine Learning*. 754–762.

[16] Wen-Chuan Lee, Yingqi Liu, Peng Liu, Shiqing Ma, Hongjun Choi, Xiangyu Zhang, and Rajiv Gupta. 2019. White-box Program Tuning. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO 2019)*. IEEE Press, Piscataway, NJ, USA, 122–135. http://dl.acm.org/citation.cfm?id=3314872.3314889

[17] Dan Li and Evangelos Kanoulas. 2018. Bayesian Optimization for Optimizing Retrieval Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, New York, NY, USA, 360–368. https://doi.org/10.1145/3159652.3159665

[18] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 297–306.

[19] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560* (2016).

[20] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali R. Butt, and Nicholas Fuller. 2014. MRONLINE: MapReduce Online Performance Tuning. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing (HPDC '14)*. ACM, New York, NY, USA, 165–176. https://doi.org/10.1145/2600212.2600229

[21] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[22] Donald Metzler and W Bruce Croft. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10, 3 (2007), 257–274.

[23] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.

[24] Ron Patton. 2005. *Software Testing (2Nd Edition)*. Sams, Indianapolis, IN, USA.

[25] Judea Pearl and Thomas Verma. 1987. The Logic of Representing Dependencies by Directed Graphs. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1 (AAAI'87)*. AAAI Press, 374–379. http://dl.acm.org/citation.cfm?id=1863696.1863763

[26] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[27] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. 2006. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*. ACM, 585–593.

[28] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational context for ranking in personal search. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1531–1540.