

On the Optimality of Top-k Algorithms for Interactive Web Applications*

Yael Amerdamer

Tel Aviv University
and INRIA
yaelamst@post.tau.ac.il

Daniel Deutch

Ben Gurion University
and INRIA
deutchd@cs.bgu.ac.il

Tova Milo

Tel Aviv University
milo@post.tau.ac.il

ABSTRACT

In an interactive Web application, the application state changes according to user choices/actions. To assist users in their interaction with such applications, there is a need to provide them with recommendations for the top-k (according to some ranking metric) interactions. These recommendations must be continually updated, as the user interacts with the application, to be consistent with the actual choices she makes. Efficiency of computation is critical here to provide fast response time and a pleasant user experience. This paper establishes formal foundations for measuring the optimality of top-k algorithms of the aforementioned type, i.e. how well they perform relative to other algorithms, with respect to all possible input instances. We define several intuitive notions of optimality in this setting, analyze the fundamental difficulties in obtaining optimal algorithms, and identify conditions under which such algorithms exist.

1. INTRODUCTION

This paper focuses on interactive Web applications [8, 10, 5], for example applications for online shopping, where the state (shopping cart, database) changes as the user navigates in the website and chooses products. To assist users in their interaction with such applications, there is a need to provide them with recommendations of the *top-k* (according to some ranking metric) interactions. These recommendations have the form of suggested sequence of navigation steps in the application [8, 7]. The user may follow one of the suggestions, but may also choose differently. In the latter case, new suggestions, consistent with the user choices, are proposed, and so on. We refer to such sequence of interleaving choices and recommendations as an *interactive top-k* computation.

The goal of this paper is to study when, and to what extent, optimal top-k algorithms are possible in an interactive

setting. To that end, we (1) provide a simple and generic model for *interactive Web application*, (2) define (interactive) *top-k problems* that should be addressed by recommendation systems in this context, (3) define *cost models* and *measures of optimality* for algorithms that solve these problems, and (4) analyze the fundamental difficulties encountered when attempting to achieve such optimal algorithms and identify practical conditions under which optimal algorithms exist.

Note that several different models [7, 9, 5, 1] can be used for interactive Web applications and some dedicated top-k algorithms were proposed for these models. We stress that our goal here is *not* to devise yet another specific, efficient top-k algorithm for any of these models. Instead, the main objective of this paper is to study, using a generic model that abstracts such specific models, the boundaries of optimality that may be achieved by top-k algorithms on interactive Web applications.

We next provide a brief overview of our contributions.

Modeling Interactive Web Applications. We model an interactive Web application as a directed graph. The nodes represent the application states and the edges model state transitions, triggered by user choices/input. A user *interaction* with the application is modeled as a path in this graph. To capture a generic analysis of applications we assume that for each transition we are only given some quantitative information (such as the price that the transition incurs to the user, the likelihood of the transition based on choices of previous users, etc.). This is modeled by edge weights, which may be aggregated, to form a path weight.

Note. In real-life, the specifications of interactive applications may include loops and recursions, and the underlying state graph may have infinitely many nodes (states). To simplify the presentation we consider here only finite Directed Acyclic Graphs (DAGs), but we stress that all of our results may be extended to general (possibly infinite) graphs with loops. This is described in the full version of the paper [14], where we define a “small world” property that allows the algorithms to cope with possibly infinite paths.

Cost Model and Optimality. We start by defining a simple cost model for the operation of a given algorithm on a given application graph, that is based on the number of graph nodes examined by the algorithm. Then, we define notions of algorithm optimality, inspired by the notion of *instance-optimality*, presented in [12] for relational settings,

*This work has been partially funded by the Israel Science Foundation, by the US-Israel Binational Science Foundation, by the EU grant MANCOOSI and by the ERC grant Webdam under agreement 226513.

in the sense that optimal performance is required on any possible input. Due to the criticality of efficiency in the context of interactive Web applications, this strong notion is much more suitable than worst case optimality. But we go even deeper, and instead of studying performance in terms of orders of magnitudes, we define metrics that are affected by the *exact* cost of algorithms.

(Non-)Existence of Optimal Algorithms. We next study *interactive top-k* computation, namely the on-going process of computing top-k recommendations following each user choice. Each recommendation is a possible interaction starting from the current state of the application. We compare the optimality results for such algorithms to those of “single” top-k computations, given different “domains” of interactive Web applications as input. Interestingly, we show that, in general, the existence of optimal algorithm for a *single* top-k computation for a given input domain *does not imply* the existence of an optimal algorithm for interactive top-k computation. However, we give sufficient conditions for such implication, and further show realistic domains of input where these conditions hold.

Lookahead. Finally, we observe that ‘idle’ time intervals, where the user views a given set of top-k recommendations, and before she makes her next choice, can be exploited to preempt some of the future top-k computations (thereby shortening future response time). We refer to such preemptive computation as *lookahead*, study lookahead algorithms, and adapt our optimality measures for this setting. We show that an optimal lookahead algorithm does not exist in general, even when an optimal interactive-top-k algorithm does exist, but identify realistic conditions that allow for such optimal lookahead algorithms.

Paper Organization. Section 2 describes our generic data model. Section 3 then considers interactive top-k computations, and Section 4 considers lookahead. We overview related work in Section 5, and conclude in Section 6.

For space constraints, the full proofs are omitted and can be found in the full version of this paper [14].

2. PRELIMINARIES

We start by presenting the formal definitions for our model.

Interactive Web Applications. We abstractly model an interactive Web application as a directed graph whose nodes correspond to application states (configurations). The directed edges stand for possible transitions between those states, triggered by user choices/input. The graph is given by a node r standing for the application initial state, and a transition function ψ capturing the edge relation. I.e. given a node n , $\psi(n)$ is a set of all possible successors of n . Some nodes are marked as *accepting*, representing some task completion. We use a domain \mathcal{V} of nodes with distinct identifiers.

DEFINITION 2.1 (APPLICATION SPECIFICATION). *An application specification (ASpec) is a tuple $s = (r, \psi, acc)$ where $r \in \mathcal{V}$ is the initial node, $\psi : \mathcal{V} \mapsto 2^{\mathcal{V}}$ is the transition function, and the function $acc : \mathcal{V} \mapsto \{0, 1\}$ determines the accepting nodes (for which $acc(v) = 1$). The underlying graph of s , $graph(s) = (V, E)$, is then a graph, where V , E are the smallest sets (in terms of set inclusion) such that*

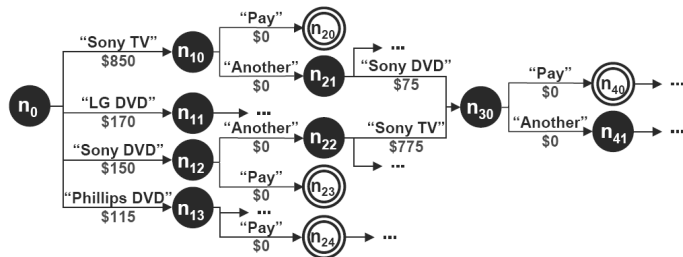


Figure 1: Interactive Web Application Specification

$r \in V$, and if $v \in V$ then $(v, v') \in E$ for each $v' \in \psi(v)$. We assume that $graph(s)$ is a finite Directed Acyclic Graph¹.

Interaction. A user of an interactive Web application makes a sequence of choices that dictates the transitions in-between application states. To capture that, an *interaction* p with a given ASpec s is defined as a path in $graph(s)$. The first node of p is denoted $start(p)$, and its last node is denoted $end(p)$. We say that an interaction p is *accepting* if $end(p)$ is accepting. The set of all possible interactions with s is denoted $inter(s)$. For two interactions p, p' , we denote $p \rightarrow p'$ if p' is a one-step continuation of p , i.e. it consists of the same nodes sequence as p , followed by an additional node $n \in \psi(end(p))$.

We next exemplify how an interactive Web application, an online store, is expressed using our simple model.

EXAMPLE 2.2. *Fig. 1 depicts the underlying graph of an ASpec for a simplified on-line store (ignore for now the prices on the edges). The root is n_0 , and each node of the graph stands for a state of the application, including for instance the shopping cart of purchased items, user choices history etc. An application user first chooses from a variety of products, may then either pay, or make another choice of product, and so on. Accepting nodes are marked with a double outline (thus e.g. $[n_0, n_{10}, n_{20}]$ is an accepting interaction). Several distinct interactions may end at the same node, e.g. $[n_0, n_{10}, n_{21}, n_{30}]$ and $[n_0, n_{12}, n_{22}, n_{30}]$ both correspond to the purchase of Sony TV and DVD (in different orders).*

Weighted ASpec. As mentioned in the Introduction, to capture generic analysis of applications, we assume that for each possible application transition we are only given some quantitative information (e.g. the transition likelihood, the monetary cost it incurs). To model this, we define a domain \mathcal{W} of weights with a total order over its elements. We further define three weight functions over ASpec, as follows: (1) a function over edges, standing for the weight of each possible choice along an interaction; (2) an aggregation function over individual edge weights and (3) a weight function over paths, obtained by using this aggregation.

Edge Weight. Given an ASpec s with $graph(s) = (V, E)$, we define a weight function over its edges, namely $W_e : E \rightarrow \mathcal{W}$. W_e may e.g. stand for the price incurred by the choice that the edge represents, the incurred delivery time of the purchased product, or its relative popularity among users.

¹As mentioned in the Introduction, we show in the full paper how the results extend to infinite graphs.

The Aggregation Function. The weights of edges along an interaction are aggregated using an aggregation function. The function $aggr : \mathcal{W} \times \mathcal{W} \rightarrow \mathcal{W}$ receives two weights as inputs; the first intuitively corresponds to the aggregated weight computed so far, and the second is the new W_e to be aggregated with the previous value. For instance, when computing purchase cost $aggr = +$ and $\mathcal{W} = [0, \infty)$; when computing path likelihood, $aggr = \times$ and $\mathcal{W} = [0, 1]$.

Following [7], we consider here aggregation functions that satisfy the following intuitive constraints:

1. $aggr$ is associative and commutative.
2. $aggr$ is *continuous*, that is for each $x, y, z \in \mathcal{W}$, if $aggr(x, y) < aggr(x, z)$ then there exists $w \in \mathcal{W}$ such that $aggr(x, y) < aggr(x, w) < aggr(x, z)$.
3. $aggr$ has a neutral value, denoted 1_{aggr} . Namely for each $x \in \mathcal{W}$, $aggr(x, 1_{aggr}) = aggr(1_{aggr}, x) = x$.
4. $aggr$ is monotonically increasing or decreasing over \mathcal{W} . I.e., either $\forall s, x, y \in \mathcal{W} \ x \geq y \Rightarrow aggr(s, x) \geq aggr(s, y)$ and $aggr(s, x) \geq s$, or the same for \leq .

Path Weight. Last, we define the weight of an interaction path in an ASpec s , namely $W_p : inter(s) \rightarrow \mathcal{W}$. W_p is obtained by aggregating the W_e values of edges along the interaction path p . For an interaction path $p = [n_0, n_1, \dots, n_t]$, W_p is defined recursively as $W_p([n_0]) = 1_{aggr}$, and $W_p([n_0, n_1, \dots, n_t]) = aggr(W_p([n_0, \dots, n_{t-1}]), W_e(n_{t-1}, n_t))$.

EXAMPLE 2.3. *Re-consider Fig. 1, and note now the prices annotating the different choices (edges). For example $W_e(n_0, n_{10}) = 850\$$, accounting for the price of a Sony TV. Also note that the weight function allows expressing dependencies of weights on prior choices, such as combined deals: the price of a Sony DVD is 150\$ when purchased as a single product ($W_e(n_0, n_{12}) = 150\$$), but is 75\$ for customers that also bought a Sony TV ($W_e(n_{21}, n_{30}) = 75\$$). $aggr = +$ here, and thus W_p reflects the total price of the purchases along a given interaction. For instance, $W_p([n_0, n_{12}, n_{22}, n_{30}]) = 150\$ + 0\$ + 775\$ = 925\$$, accounting for the total price of purchasing Sony DVD and Sony TV. In this example, the two interactions leading to the same node (n_{30}) bear the same weight, but in general different interactions ending at the same node may bear different weights.*

Top-k. Observe that when $aggr$ is monotonically increasing (resp. decreasing), so is W_p , in the sense that the weight of an interaction increases (decreases) as it advances. Generally, when $aggr$ is monotonically increasing (as, e.g., for the overall price of purchases), we are interested in the bottom-k interactions (e.g. the cheapest overall price). When $aggr$ is monotonically decreasing (as, e.g., path popularity), we are interested in the top-k (e.g. the most likely). All our results apply symmetrically to both cases, thus *we focus from now on only monotonically decreasing functions.*

Given a node $n \in graph(s)$, we denote by $top-k(s, n)$ the k -highest weighted, accepting interactions p in $inter(s)$ such that $start(p) = n^2$. $top-k(s, n)$ is well-defined when there exist at least k distinct such interactions; in the sequel we always assume that this is the case.

²As multiple interactions may share the same weight, this set may not be unique, and we pick one arbitrarily.

We note that there are cases where only interactions that satisfy some query criteria are of interest to the user. For many Web applications models (e.g. the one based on Business Processes [2, 7], or Active XML [1]) there are known query evaluation algorithms that “intersect” the ASpec s and the query q , yielding a refined ASpec s' , consisting only of interactions of s that satisfy q . Top-k analysis may then be employed over s' .

We define TOP-K as the problem of computing, given the following input: (1) an ASpec $s = (r, \psi, acc)$, (2) the weight functions, W_e and $aggr$, (3) a node n of s , and (4) a number k of requested results, the set $top-k(s, n)$.

We denote the class of algorithms that correctly solve TOP-K by \mathcal{A}_{TOP-K} , and require that every algorithm in \mathcal{A}_{TOP-K} may only discover the shape of s via applications of ψ on the root or previously discovered nodes, and apply the weight functions only on such discovered edges/paths.

Interactive Top-k. The user starts her interaction with an ASpec s at the root state $r = n_0$. To assist the user, there are recommendation systems that provide her with suggestions on how to interact with the application [8]. In our model, this corresponds to $top-k(s, n_0)$. She may then either follow one of the given recommendations, or choose differently. In the latter case, new top-k recommendations, consistent with the actual user choices, are proposed. This is repeated as the interaction continues.

An *interactive* top-k algorithm receives, instead of a single node, an interaction $p = [n_0, n_1, \dots]$, given node by node. For every node n_j of p , the algorithm should compute $top-k(s, n_j)$.

We formally define the problem of ITOP-K as follows: we are given an ASpec $s = (n_0, \psi, acc)$ as input, and then an interaction in s , $p = [n_0, n_1, \dots]$, given node by node. We should first compute $top-k(s, n_0)$. Then, at the i -th step, $i > 0$, we are given a node $n_i \in \psi(n_{i-1})$, and should output $top-k(s, n_i)$. For that, the algorithm may use computations done in prior steps, or invoke ψ to further explore $graph(s)$ (and obtain weights of explored edges). We denote the class of algorithms that correctly solve ITOP-K by \mathcal{A}_{ITOP-K} .

EXAMPLE 2.4. *Re-consider Fig. 1. The top-1 interaction starting from the application root, denoted $top-1(s, n_0)$, consists of the purchase of Philips DVD, followed by a payment, i.e. is the interaction $[n_0, n_{13}, n_{24}]$. Indeed, this is the cheapest product available. The user may then either choose a Philips DVD, in which case the recommendation is not recomputed ($top-1(s, n_{13}) = [n_{13}, n_{24}]$), or she may wish to make other purchases, and e.g. choose to purchase a Sony TV, leading to n_{10} . In this case, the algorithm should compute $top-1(s, n_{10})$. If the user chooses to continue purchasing (i.e. get to n_{21}), then the cheapest product is now a Sony DVD, thus $top-1(s, n_{21}) = [n_{21}, n_{30}, n_{40}]$, etc.*

3. OPTIMAL TOP-K ALGORITHMS

We define in this section cost model and optimality notions for top-k algorithms in our context, then study the existence of such optimal algorithms.

3.1 Cost Model and Optimality Notions

Denote \mathcal{I} (\mathcal{I}^{inter}) as the set of all input instances for TOP-K (ITOP-K). Given a (interactive) top-k algorithm A and an ASpec $s = (r, \psi, acc)$ as a part of the input to A , we consider the total number of calls A makes to ψ , referred to as *data*

accesses, as the dominant computational cost factor, since it indicates the number of nodes of $graph(s)$ that A visits. We denote by $nodes(A, I)$ the set of all nodes n for which A invokes $\psi(n)$, when executed over I .

Note. In general an algorithm may invoke ψ multiple times over the same node. However, we focus here on algorithms that, whenever they invoke $\psi(n)$ for some node n , can record the results set $\psi(n)$ in memory for later reuse. Consequently, each data access is performed at most once. We mention the case of limited amount of memory in Section 6.

c-optimality. We next define several optimality notions for top-k algorithms on interactive applications. Since we observe below that the existence of optimal algorithms depend on the input applications properties, we will define optimality with respect to a given (restricted) domain of inputs.

DEFINITION 3.1. *Given a natural number c , we say that an algorithm $A \in \mathcal{A}_{TOP-K}$ (resp. \mathcal{A}_{ITOP-K}) is c -optimal with respect to an input domain $\mathcal{D} \subseteq \mathcal{I}$ (resp. $\mathcal{D}^{inter} \subseteq \mathcal{I}^{inter}$) if for every $A' \in \mathcal{A}_{TOP-K}$ (resp. \mathcal{A}_{ITOP-K}) and an input instance $I \in \mathcal{D}$ (resp. \mathcal{D}^{inter}), $|nodes(A, I)| \leq c \cdot |nodes(A', I)|$.*

Note that a 1-optimal algorithm A satisfies $|nodes(A, I)| \leq |nodes(A', I)|$ for each such A' and I .

In the sequel, given an input domain $\mathcal{D} \subseteq \mathcal{I}$, we use D^{inter} to denote the corresponding input domain to an interactive top-k computation (i.e. the application specifications appearing in both are the same, but in D the input includes a start node, while in D^{inter} it includes an interaction).

3.2 Optimality Results

The existence of optimal algorithms for *single* top-k computation was studied in [7], in the context of Business Process applications, and we show in the full version of this paper [14] that the results extend to our generalized setting. We start by a brief overview of these results, which will be of importance when we consider *interactive* top-k below.

(Single) Top-k Computation and Restricted Input Domains. Recall that we assumed (weak) monotonicity of the weight function W_p , i.e. that the weight of a path cannot increase as more choices are made. We show in [14] that the existence of an optimal top-k algorithm depends heavily on “*how monotone*” is W_p . In particular, we can show (see [14]) weak monotonicity is not enough to allow for c -optimal algorithms. Consequently, we define the notion of *i-strong monotonicity*, as follows.

DEFINITION 3.2. *Given a natural number i , W_p is i -strongly monotone if the number of different paths that start from any node n and bear identical weight is bounded by i .*

i -strongly monotone functions occur naturally in practice, with a relatively small bound i . For instance, if the weight function stands for popularity of user choices then W_p captures the aggregated popularity of choices sequence, and it is uncommon for many such sequences to have the exact same popularity. Similarly for weight function capturing monetary cost, the number of different purchases incurring the exact same price is typically bounded; an exception to this are paths having a weight of 0 (where no products are purchased), but such paths are typically “not interesting” for analysis purposes, and may be pruned in pre-processing.

We use $\mathcal{I}_{mono(i)}$ to denote the class of all input instances where W_p is i -strongly monotone. The following proposition is proved in [14].

PROPOSITION 3.3. *Let c be a natural number. There exists a TOP-K algorithm A s.t. A is c -optimal w.r.t. the input domain $\mathcal{I}_{mono(c)}$.*

Note that in particular this implies the existence of a 1-optimal algorithm for $\mathcal{I}_{mono(1)}$, the class of top-k input instances with a *strongly monotone* weight function.

Interactive Top-k Computation. We now turn to analyze the possible c -optimality of interactive top-k algorithms, for different input domains partial to \mathcal{I}^{inter} .

First, we are interested in whether or not conditions on the input class that were enough to guarantee the existence of optimal top-k algorithms, also allow for interactive top-k algorithms. The answer is no: the existence of a 1-optimal TOP-K algorithm in a given domain does not imply the existence of even a c -optimal ITOP-K algorithm.

THEOREM 3.4. *There exists an input domain $D \subseteq \mathcal{I}$ of infinite cardinality s.t. there exists an algorithm in \mathcal{A}_{TOP-K} that is 1-optimal w.r.t. D , but, for every natural c , no algorithm in \mathcal{A}_{ITOP-K} is c -optimal w.r.t. D^{inter} .*

PROOF SKETCH. We first define an input domain D where there exist many paths with equal W_p . We give a simple algorithm in \mathcal{A}_{TOP-K} which is 1-optimal for this domain, and then show that there exists no c -optimal algorithm in \mathcal{A}_{ITOP-K} for D^{inter} . Intuitively, this is because the algorithm does not always pick the same paths as the user. \square

However, a stricter requirement on the optimality of top-k algorithms can guarantee optimal interactive top-k algorithms, as follows.

DEFINITION 3.5. *$A \in \mathcal{A}_{TOP-K}$ is strictly c -optimal w.r.t. \mathcal{D} if there exists some natural c such that for every $A' \in \mathcal{A}_{TOP-K}$, and for every $I \in \mathcal{D}$ s.t. A' halts when given I as input, $|nodes(A, I) - nodes(A', I)| \leq c - 1$.*

Note that *strict c -optimality* that concerns the data accesses themselves, and requires that every correct algorithm makes the *exact same* accesses as the strictly c -optimal algorithm (except for $c - 1$ ³ such accesses) and possibly more. Clearly, if A is strictly c -optimal then it is c -optimal, thus it is a “stronger” optimality measure. We can strengthen proposition 3.6 to show that in $\mathcal{I}_{mono(c)}$ there exists even a *strictly c -optimal* TOP-K algorithm.

PROPOSITION 3.6. *There exists an algorithm TOP-K-ALGO $\in \mathcal{A}_{TOP-K}$, such that for every natural number $c \geq 1$, TOP-K-ALGO is strictly c -optimal in $\mathcal{I}_{mono(c)}$.*

Now, in contrast to Theorem 3.4, we show that the existence of a *strictly* 1-optimal TOP-K algorithm does guarantee the existence of a 1-optimal ITOP-K algorithm. In fact, it assures the existence of a *strictly* 1-optimal such algorithm.

THEOREM 3.7. *For any input domain $D \subseteq \mathcal{I}$, if there exists an algorithm $A \in \mathcal{A}_{TOP-K}$ that is strictly 1-optimal w.r.t. D , then there exists an algorithm $B \in \mathcal{A}_{ITOP-K}$ that is strictly 1-optimal w.r.t. D^{inter} .*

³Using $c - 1$ here instead of c is only for considerations of symmetry w.r.t. the definition of c -optimality

PROOF SKETCH. Such ITOP-K algorithm can be obtained simply by applying the strictly 1-optimal TOP-K algorithm at each step of the interaction. Intuitively, the proof shows that since no data access is performed more than once (as noted above), and all the performed accesses are necessary for some top-k computation along the interaction, the ITOP-K algorithm is indeed (strictly) 1-optimal. \square

Combined with Proposition 3.6, we obtain the following.

COROLLARY 3.8. *There exists a strictly 1-optimal ITOP-K algorithm over $\mathcal{I}_{mono(1)}^{inter}$.*

Similarly, we may show that for every natural number c the existence of a strictly c -optimal top-k algorithm suffices for the existence of a c -optimal (but not necessarily strictly c -optimal) interactive top-k algorithm.

THEOREM 3.9. *For any input domain $D \subseteq \mathcal{I}$, if there exists an algorithm $A \in \mathcal{A}_{TOP-K}$ that is strictly c -optimal over D for some natural c , then there exists an algorithm $B \in \mathcal{A}_{ITOP-K}$ that is c -optimal over D^{inter} .*

Combined with Proposition 3.6 we obtain:

COROLLARY 3.10. *For every natural i , there exists a c -optimal ITOP-K algorithm over $\mathcal{I}_{mono(i)}^{inter}$, with $c = i$.*

4. LOOKAHEAD

So far, we have measured the performance of algorithms for ITOP-K as a function of the overall computation (data accesses) it performs. In practice, the *timing* of computations is also crucial: there are intervals of time during the interaction when the system is idle - when the user views the top-k results, and before she makes her next choice. An ideal algorithm would preempt some computations and execute them in these intervals of time (thereby shortening future response time). We refer to such preemptive computation as *lookahead*, and study optimal lookahead algorithms.

Lookahead Instructions. A lookahead algorithm LA aims to assist an algorithm $A \in \mathcal{A}_{ITOP-K}$ by instructing it which top-k computations to perform in advance, when, and how much resources (data accesses) to allocate for each such computation. A lookahead *instruction* is thus a pair (n, st) such that n is a node and st is a natural number. Its semantics is: “(continue to) compute $top-k(s, n)$ for st steps (or until interrupted)”. If $st = \infty$, the semantics of the instruction is to run $top-k(s, n)$ until completing the computation (unless interrupted earlier). We assume that the top-k computation can be stopped at any point and resumed later on from the same point, thus previous (partial) computations are useful.

Input to Lookahead Algorithms. To decide which instructions to issue, the lookahead algorithm has access to the ASpec $s = (r, \psi, acc)$. Similarly to algorithms in \mathcal{A}_{ITOP-K} , the lookahead algorithm is further given an interaction in s , node by node, starting from the root $r = n_0$. At the i -th step, $i > 0$, it is given a node $n_i \in \psi(n_{i-1})$; the goal is to preempt computations performed for $top-k(s, n_j)$, for $j > i$. (The $top-k(s, n_j)$ results for $j \leq i$ were already computed.)

4.1 Performance Measures

We define the class \mathcal{A}_{LA} of all deterministic lookahead algorithms that are given the same input as depicted above.

We consider two properties of lookahead algorithms, namely its *resource* consumption and its *utilization*, defined next.

Resources. At any point i of the interaction, a lookahead algorithm has a limited time interval in which it may perform computations. This interval is the time after top-k computation for the user choice n_i has been completed, and before the following user choice n_{i+1} is submitted. We denote the maximal number of data accesses that fits in this time interval by Δ_i .

For a lookahead algorithm, the number of data accesses it induces at the i -th interval of the interaction is the total number of data accesses it performs itself, plus the total number of steps listed in instructions (to the top-k algorithm) it outputted. This quantity must not exceed Δ_i . Δ_i is unknown to the lookahead algorithm: the instructions (and further operations, for this step) of LA are “cut” once the total number of data accesses reaches Δ_i .

Utilization. The *utilization* of an algorithm LA captures the “quality” of its instructions, i.e. how many useful computations (data accesses) it preempted. To define this formally, we use the following notations. Consider an algorithm $A \in \mathcal{A}_{ITOP-K}$ when operating on an ASpec s and an interaction $p = [n_0, \dots, n_t]$ (no lookahead used). We use $nodes^{>i}(A, s, p)$, for $i = 0, \dots, t-1$, to denote the set of nodes for which A invokes ψ , after receiving n_{i+1} and until the end of the interaction. Now, given also a lookahead algorithm LA , each instruction of LA causes A to perform some sequence of preempted data accesses; we denote the set of such accesses performed at the i -th step (i.e. after computing $top-k(s, n_i)$ but before receiving n_{i+1}), as $pre^i(LA, A, s, p)$. The utilization of LA w.r.t. A is then defined as follows.

DEFINITION 4.1. *Let $LA \in \mathcal{A}_{LA}$ and $A \in \mathcal{A}_{ITOP-K}$. Given an ASpec s and some interaction $p = [n_0, \dots, n_t]$ with s , the utilization of LA w.r.t. A, s and p , is defined as $util(LA, A, s, p) = \sum_{i=0, \dots, t-1} |pre^i(LA, A, s, p) \cap nodes^{>i}(A, s, p)|$.*

4.2 Optimality

We next study a number of possible optimality measures for lookahead algorithms, based on their utilization. We denote below by $accInter(s)$ the set of accepting interactions with s , starting at its initial state.

Maximal Utilization. One possible goal of a lookahead algorithm LA is to maximize $util(LA, A, s, p)$ for every algorithm $A \in \mathcal{A}_{ITOP-K}$, every ASpec s and interaction $p \in accInter(s)$. This is appealing, as such algorithm will minimize the overall response time for all accepting interactions. We thus say that, given a natural c , an algorithm LA is c -optimal if $c \cdot util(LA, A, s, p) \geq util(LA', A, s, p)$ for every A, s and p and every $LA' \in \mathcal{A}_{LA}$.

However, no c -optimal algorithm exists, even for very simple input classes that do allow for 1-optimal (interactive) top-k algorithms. More concretely, let $\mathcal{I}_{MonoTree}^{inter}$ be the class of all input ASpecs whose weight function is strictly monotone, and whose graph is tree-shaped.

THEOREM 4.2. *For every natural c , there exists no c -optimal algorithm in \mathcal{A}_{LA} , w.r.t. $\mathcal{I}_{MonoTree}^{inter}$.*

Maximal Expected Utilization. Since a c -optimal algorithm is impossible even for simple classes, we also consider a more relaxed notion of optimality, based on the observation that not all interactions are equally common. To that end, we define the notion of interaction *likelihood* as a specific weight function. Namely, we use W_e that is a distribution (i.e. for every n that has outgoing edges, $\sum_{n' \in \psi(n)} W_e(n, n') = 1$), and use $aggr = \times$. We denote the obtained W_p by $pLikelihood$. Based on this function, we then define the *expected* utilization of a lookahead algorithm.

DEFINITION 4.3. *The expected utilization of lookahead algorithm LA , w.r.t. an ASpec s and $A \in \mathcal{A}_{ITOP-K}$ is defined as $E[util](LA, A, s) = \sum_{p \in accInter(s)} pLikelihood(p) \cdot util(LA, A, s, p)$.*

We say that an algorithm LA is *exp- c -optimal* for a given natural c if $cE[util](LA, A, s) \geq E[util](LA', A, s)$ for every A, s and every $LA' \in \mathcal{A}_{LA}$. We next identify a case where an exp- c -optimal algorithm exists.

DEFINITION 4.4. *Given a natural i , denote by $I_{shallow}(i)$ the domain of all input instances such that the total number of interactions p, p' where $length(p) > length(p')$ and $pLikelihood(p) > pLikelihood(p')$, is bounded by i .*

For example, consider an ASpec of an online shop, where paths correspond to choices of products categories, sub-categories, etc. up until the choice of a specific product. The above bound means here e.g. that not many individual products are more likely than an entire category of products, which is typically the case.

The following theorem holds:

THEOREM 4.5. *For every natural c , there exists an exp- c -optimal lookahead algorithm over $I_{shallow}(c)$.*

The existence of an exp- c -optimal algorithm for the general case is an open problem.

5. RELATED WORK

Top-k algorithms were studied in different contexts (e.g. [12, 18, 19, 17, 4, 6]), Specifically, the seminal work of [12] introduced the notions of *instance-optimality*, and had many follow-ups (e.g. [19, 4]). Instance-optimality uses order of magnitude of the computational cost as the measure for optimality. Due to the criticality of efficiency in the context of interactive applications, our work has further considered the *exact* computational cost rather than orders of magnitude. Analogous definitions appear in the context of on-line algorithms, referred to as *competitiveness* notions [16].

Our work has built upon a simple and generic modeling of interactive Web applications as state machines. Various top-k algorithms were developed for analysis of state-machines/graphs (e.g. [3, 6, 11]). However, our work is the first (to our knowledge) to propose generic measures of optimality for *interactive* top-k algorithms, and to characterize under which conditions such optimality may be achieved. Our results may be used as yardsticks on what one may expect, in terms of optimality, from (interactive) top-k algorithms on a given application model.

We note that our notion of optimal *utilization* used to measure the quality of lookahead algorithms, is inspired by notions from the theory of communication networks [15, 13].

As in networks, we aim to optimize the quality of service [13] provided to users; however, research in this area does not consider (interactive) top-k analysis of applications, rendering the problems and algorithms completely different.

6. CONCLUSION

This paper establishes formal foundations for measuring the optimality of top-k algorithms for interactive Web applications. We have defined several intuitive notions of c -optimality, analyzed the difficulties in achieving them, and identified conditions under which c -optimal algorithms exist.

Future Work. We assumed that the algorithms may record in memory, and then reuse, information about previous data accesses. Limited memory renders c -optimality harder. Initial results that we obtained show that, in the general case, if the available memory does not suffice to record all nodes accessed by a strictly c -optimal TOP-K (resp. ITOP-K) algorithm operating without memory constrains, then no TOP-K (resp. ITOP-K) algorithm with such memory bound is c -optimal. In contrast, if the graph of the input ASpec is guaranteed to be tree-shaped, sufficient memory for (a single) TOP-K computation allows all our optimality results, for both TOP-K and ITOP-K, to go through. A full analysis of c -optimality under memory (and other) budgetary constraints is an ongoing research direction. Additional future work includes the analysis of further restricted yet practical cases that allow for c -optimal algorithms.

7. REFERENCES

- [1] S. Abiteboul, P. Bourhis, and B. Mariniou. Satisfiability and relevance for queries over active documents. In *PODS*, 2009.
- [2] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *Proc. of VLDB*, 2006.
- [3] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, 1991.
- [4] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal xml pattern matching. In *SIGMOD*, 2002.
- [5] T. Bultan, J. Su, and X. Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1), 2006.
- [6] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proc. of ICDT*, 2009.
- [7] D. Deutch, T. Milo, N. Polyzotis, and T. Yam. Optimal top-k query evaluation for weighted business processes. *PVLDB*, 3(1), 2010.
- [8] D. Deutch, T. Milo, and T. Yam. Goal Oriented Website Navigation for Online Shoppers. In *Proc. of VLDB*, 2009.
- [9] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *Proc. of SIGMOD*, 2005.
- [10] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *PODS*, 2006.
- [11] D. Eppstein. Finding the k shortest paths. In *FOCS*, 1994.
- [12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.
- [13] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet*. Addison-wesley, 1998.
- [14] Full version. <http://www.cs.tau.ac.il/~yaelamst/InteractiveFull.pdf>.
- [15] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *JORS*, 49, 1998.
- [16] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *STOC*, 1988.
- [17] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in xml. In *ICDE*, 2005.
- [18] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE*, 2007.
- [19] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *ICDE*, 2009.