On the Optimality of Top-k Algorithms for Interactive Web Applications (Full Version)

Yael Amsterdamer
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 We application as a possibly infinite graph. The nodes represent the application states and the edges model state transitions, triggered by user choices / input. Some of the graph nodes are marked as *accepting*, representing the completion of some task. A user *interaction* with the application is modeled as a (possibly infinite) path in this graph. While there are cases where the semantics of state transitions is exposed to the analyst and a tailor-made analysis may be employed, 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.

Top-k Problems. We consider here three problems, as follows.

- We start by considering a (single) top-k computation at a given state n. The top-k recommendations are the k highest weighted interactions, starting at n and leading to an accepting state.
- Second, we study interactive-top-k computation, namely the on-going process of computing top-k recommendations following each user choice.
- 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*, and study lookahead algorithms.

Cost Model and Optimality. As described above, our goal here is not to devise specific algorithms but rather to test the boundaries of optimality, of top-k algorithms of the above flavor. To that end, we define a simple cost model for the operation of a given algorithm on a given Web application specification, that is based on the number of graph nodes examined by the algorithm. We then define notions of algorithm optimality, inspired by the seminal work of [13] that studies optimality of top-k algorithms in relational settings. In [13] the authors introduced the notion of *instance-optimality*: given a cost model, they define an algorithm to be instance-optimal if *for any possible input instance*, its cost of operation is at most the same order of magnitude as that of any other correct algorithm. 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 the algorithms performance in terms of orders of magnitudes, we define metrics that are affected by the *exact* cost of algorithms. Namely we define the notion of *c-optimality*, and say that an algorithm is c-optimal if for any possible input instance, its cost of operation is at most greater by a factor of c than the cost of any other correct algorithm. At the extreme, a *1-optimal* algorithm performs at least as well as any other algorithm, for any given input instance.

(Non-)Existence of Optimal Algorithms. We next outline our results on the (non-)existence of optimal algorithms for the different algorithms classes, under different restrictions.

We start by considering a (single) top-k computation at a given state n. We show that the properties of path weights affect the existence of c-optimal algorithms (for different values of c). Following common practice [13], we focus on weight functions that are monotonic w.r.t. the progress of the interaction (path). This captures most practical scenarios (e.g. the total price of a shopping cart subset does not exceed the price of the full cart, even in presence of discount deals). In the case of general monotone path weights, we show that for any natural number c, no c-optimal algorithm exists. We then consider a slightly restricted class of monotone path weights that arise frequently in practice. We show that a c-optimal algorithm exists here, for c that depends only on the definition of the class, and we present a simple generic such algorithm.

Second, we study interactive top-k computation, namely the on-going process of computing top-k recommendations following each user choice. Interestingly, we show that, in general, the existence of a c-optimal algorithm for a *single* top-k computation for a given input domain *does not imply* the existence of a c-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.

Finally, we consider algorithms for preemptive 'lookahead' computation, and adapt our optimality measures for this setting. We show that a c-optimal lookahead algorithm does not exist in general, even when a 1-optimal interactive-top-k algorithm exists, but identify realistic conditions that allow for such c-optimal algorithms.

Paper Organization. Section 2 describes our generic data model. Section 3 then considers (single) top-k computations, section 4 studies interactive top-k computations and section 5 considers lookahead. We overview related work in Section 6, and conclude in section 7.

2. PRELIMINARIES

We start by presenting the formal definitions for our model, along with intuitive examples.

Interactive Web Application. We abstractly model an interactive Web application as a possibly infinite directed graph whose nodes correspond to application states (configurations) and its directed edges stand for possible transitions between different configurations. Transitions are 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. That is, given a node n, $\psi(n)$ is a finite set of all possible successors of n. Some graph nodes are marked as *accepting*, representing the completion of some task.

We assume in the sequel 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 of the application, $\psi : \mathcal{V} \mapsto 2^{\mathcal{V}}$ is the transition function, s.t. $\psi(v)$ is finite for each $v \in \mathcal{V}$, and the function $acc : \mathcal{V} \mapsto \{0, 1\}$ determines the* accepting *nodes (those for which $acc(v) = 1$). The* underlying graph *of s, $graph(s) = (V, E)$, is then a (possibly infinite) graph, where V, $E \subseteq V \times V$ are the smallest (in terms of set inclusion) sets such that $r \in V$, and if $v \in V$ then $(v, v') \in E$ for each $v' \in \psi(v)$.*

Interaction. A user of an interactive Web application makes a sequence of choices that dictates the transitions in-between ASpec states. To capture that, an *interaction* p with a given ASpec s is defined as a (possibly infinite) path in $graph(s)$. The first node of p is denoted $start(p)$, its last node (in case p is finite) is denoted $end(p)$, and its length is denoted $length(p)$ (for an infinite p, $length(p) = \infty$). We say that an interaction p is *accepting* if p is finite and $end(p)$ 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 online store is expressed using our simple model.

EXAMPLE 2.2. *Fig. 1 depicts the underlying graph of an ASpec for an on-line store (ignore for now the prices annotating the edges). The root is n_0, and each node of the graph stands for a configuration of the application, including for instance the shopping cart of purchased items, user choices history etc. A user of the application first chooses from a variety of products, and may then either pay, or make another choice of product, an unbounded number of times.*

The doubly bounded nodes are accepting*, and correspond to the completion of a purchase. 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 order). The interaction* $[n_0, n_{10}, n_{20}]$ *is an example for an accepting interaction.*

Weighted ASpec. As mentioned in the Introduction, to capture generic analysis of applications, we assume that for each possible transition of the application we are only given some quantitative information (e.g. the transition likelihood, the monetary cost it incurs, etc.). To model this, we define a domain $\mathcal{W}$ of weights with some total order over its elements. We further define three weight functions over ASpecs, as follows: W_e is a function over the ASpec edges, standing for the weight of each possible choice along an interaction; the weights of edges along an interaction path are aggregated, using a function *aggr*, to form the weight of the entire path, named W_p. We make this formal below:

Edge Weight. Given an ASpec s with $graph(s) = (V, E)$, we define a weight function over its edges, namely $W_e : E \to \mathcal{W}$. 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.

The Aggregation Function. The weights of edges along an interaction are aggregated using an aggregation function. The function $aggr : \mathcal{W} \times \mathcal{W} \to \mathcal{W}$ 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 $\mathcal{W} = [0, 1]$.

We consider here aggregation functions that satisfy the following intuitive constraints:

1. *aggr* is associative and commutative, namely for each $x, y, z \in \mathcal{W}$, $aggr(aggr(x, y), z) = aggr(x, aggr(y, z))$, and $aggr(x, y) = aggr(y, x)$.
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 $\mathcal{W}$. Namely, either for each $s, x, y \in \mathcal{W}$ $x \geq y \implies aggr(s, x) \geq aggr(s, y)$ and $aggr(s, x) \geq s$, or the same for $\leq$.

Observe that the aggregation functions $+$ and $\times$, used above for cost and likelihood, satisfy the constraints.

Path Weight. Last, we define the weight of an interaction path in an ASpec s, namely $W_p : inter(s) \to \mathcal{W}$. W_p is obtained by aggregating the W_e values of edges along the interaction path p. For a finite interaction path $p = [n_0, n_1, ..., n_t]$, W_p is defined recursively as $W_p([n_0]) = 1_{aggr}$, and $W_p([n_0, n_1, ..., n_t]) = aggr(W_p([n_0, ..., n_{t-1}]), W_e(n_{t-1}, n_t))$. For infinitely long interactions $p = [n_0, n_1, ...]$, $W_p(p) = lim_{t \to \infty} W_p([n_0, ..., n_t])$. As we require above that *aggr* is monotone, the limit exists (but may be infinite).

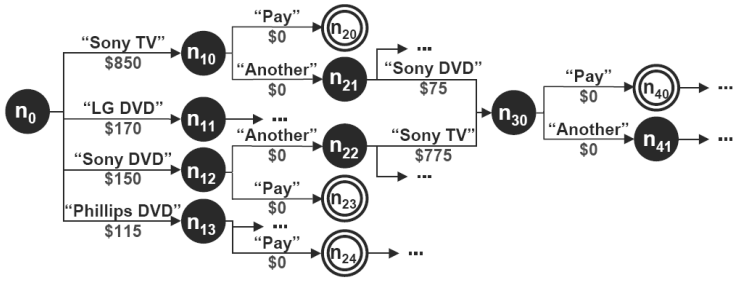

Figure 1: Interactive Web Application Specification

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). As all definitions and algorithms presented below apply symmetrically to both cases, we consider from now on only monotonically decreasing functions and top-k interactions.

Given a node $n \in graph(s)$, we denote by $top\text{-}k(s, n)$ the k-highest weighted, accepting, finite interactions p in $inter(s)$ such that $start(p) = n$ [1]. $top\text{-}k(s, n)$ is well-defined when there exist at least k distinct such interactions; in the sequel whenever we consider $top\text{-}k(s, n)$ for some s,n,k, we assume that this is the case.

We focus on finite interactions as they are more interesting for analysis purpose, in the sense that they may serve as reasonable recommendations for users on how to interact with the application. We shall see, however, that the existence of infinite interactions with high weight may still render the identification of the top-k finite ones harder.

In addition, we note that, in general, 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 essentially "intersect" the ASpec s and the query q, yielding a refined ASpec s', consisting only of interactions of s that

[1] As multiple interactions may share the same weight, this set is not necessarily unique, and we pick one such set arbitrarily.

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) weight functions, namely W_e over the edges of $graph(s)$ and an aggregation function $aggr$, (3) a node $n \in graph(s)$ and (4) a number k of requested results, the set $top\text{-}k(s, n)$.

We note that algorithms with the above input may discover the shape of s only 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. To assist the user, typical recommendation systems provides her with suggestions on how to interact with the application [8]. In our model, this corresponds to $top\text{-}k(s, r)$. She may then either follow one of the given recommendations, or choose to make choices different than those proposed. In the latter case, new top-k recommendations, consistent with the actual user choices, are proposed. This is repeated as the interaction continues.

Recall that a TOP-K algorithm takes as input an ASpec s and a single node n. An *interactive* top-k algorithm receives, instead of a single node, an interaction $p = [n_0, n_1, ...]$, given node by node. For every node n_j of p, the algorithm should compute $top\text{-}k(s, n_j)$.

More concretely, we define the problem of ITOP-K as follows: we are first given an ASpec $s = (r, \psi, acc)$ as input, and then given an interaction in s, node by node, starting from the ASpec root $r = n_0$. We should first compute $top\text{-}k(s, n_0)$. Then, at the i'th step, $i > 0$, we are given a node $n_i \in \psi(n_{i-1})$, and should output $top\text{-}k(s, n_i)$. For that, the algorithm may use computations done in prior steps, and may also invoke ψ to further explore $graph(s)$ (and obtain weights of explored edges / paths).

EXAMPLE 2.4. *Re-consider Fig.!1, and assume $k = 1$. The top-1 interaction starting from the ASpec root, denoted $top\text{-}1(s, n_0)$, 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 make a choice of a Philips DVD, in which case the top-1 recommendation does not need to be recomputed ($top\text{-}1(s, n_{13}) = [n_{13}, n_{24}]$), or she may wish to make other purchases, and may e.g. choose to purchase a Sony TV, leading to n_{10}. In this case, the algorithm should compute $top\text{-}1(s, n_{10})$. If now the user chooses to continue purchasing products (i.e. get to n_{21}), then the cheapest product for her may now be a Sony DVD, thus $top\text{-}1(s, n_{21}) = [n_{21}, n_{30}, n_{40}]$, etc.*

We next consider TOP-K algorithms, then study ITOP-K algorithms, and finally accompany such algorithms with a preemptive computation. For each problem we define appropriate notions of optimality, and explore under which conditions they can be achieved.

3. TOP-K

Observe that, as our model captures arbitrary infinite state machines (and specifically Turing Machines), solving TOP-K is generally impossible (as a solution to TOP-K would imply a solution to the halting problem). Consequently, we define $\mathcal{A}_{\text{TOP-K}}$ as the class of all *sound* (and not necessarily halting) deterministic algorithms for TOP-K that have no additional input other than that depicted above.

However, we may impose a reasonable restriction over the input to guarantee termination of analysis algorithms. Intuitively, we study input instances that satisfy a "small world" property allowing top-k algorithms to look only at a finite subset of interactions. To formalize this, let $kWeight(n)$ denote the lowest weight of an interaction in $top\text{-}k(s, n)$.

DEFINITION 3.1. *Given an ASpec s (with some weight functions) and a number k, we say that s is* k-finitely enumerable *if for each node n, there exists a finite l such that for every path p s.t. $start(p) = n$ and $length(p) \geq l$, $W_p(p) < kWeight(n)$.*

We note that typically, the "interesting" fragments of real-life applications (and weight functions over which) are k-finitely enumerable, for every value of k (where $top\text{-}k(s, n)$ is well-defined). Consider e.g. a weight function reflecting the popularity of each choice among users: it's unreasonable in real-life cases that the cumulative popularity of some infinitely long interaction is higher than that of the k-most popular finite interactions. For weight functions depicting product prices, an example for an application that is not k-finitely enumerable is one with an infinitely long path where all choices induce 0 price; such path is typically "not interesting" for analysis purposes, and may be pruned in pre-processing.

Denote $\mathcal{I}$ as the set of all input instances for TOP-K, and further denote $\mathcal{I}_{fin} \subset \mathcal{I}$ as the subset of input instances where the weighted ASpec is k-finitely enumerable (w.r.t. k given as part of the input). We show below (proof of Theorem 3.6) the existence of a sound TOP-K algorithm for $\mathcal{I}$, which is further guaranteed to terminate when given as input an instance from $\mathcal{I}_{fin}$.

3.1 Cost Model and Optimality Notions

We next define a cost model for the operation of top-k algorithms in this context. This cost model will be used in the sequel for defining the optimality of algorithms.

Given an algorithm A and an ASpec $s = (r, \psi, acc)$ fed as input to A, we consider the 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. Given an algorithm $A \in \mathcal{A}_{\text{TOP-K}}$ and an input instance $I \in \mathcal{I}$, we denote $nodes(A, I)$ as the set of all nodes n for which A invokes $\psi(n)$, when executed over I. We note that in principle, 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 and reuse it later on when needed. Consequently, each data access is performed at most once. Such algorithms have naturally lower cost (in terms of data access) than algorithms with repeated accesses, and thus are better candidates for optimality analysis. We mention the case of limited amount of memory in Section 7.

Optimality Notions. We next define several optimality notions for top-k algorithms on interactive Web applications. As outlined in the Introduction, we will consider very strong notions of optimality, namely optimality *for every possible input instance*, in a given input domain. We further care about constants in the algorithms cost, and define notions of optimality with respect to such constants, as follows.

Given a natural number c, we say that an algorithm $A \in \mathcal{A}_{\text{TOP-K}}$ is *c-optimal* with respect to an input domain $\mathcal{D} \subseteq \mathcal{I}$ if for every $A' \in \mathcal{A}_{\text{TOP-K}}$ and an input instance $I \in \mathcal{D}$, $|\, nodes(A, I)| \leq c \cdot |\, nodes(A', I)|$. For $c = 1$, a 1-optimal algorithm A satisfies $|\, nodes(A, I)| \leq |\, nodes(A', I)|$ for each such A' and I, i.e. a 1-optimal algorithm is the "best possible", in a very strong sense.

For an algorithm A that does not terminate when executed over I, we assume that there exists some natural number n such that A performs at least one data access following every n computation steps; in this case $|\, nodes(A, I)| = \infty$, and consequently any other algorithm has at most the cost of A for the input instance I.

We also introduce here an additional, stricter, notion of optimality. Recall that c-optimality concerns the *number* of data accesses, and requires that every sound algorithm must make at least the same number of accesses as the c-optimal algorithm (up to a fraction of c), and possibly more. We define the notion of *strict c-optimality* that concerns the data accesses themselves, and requires that every terminating sound algorithm makes the *exact same* accesses as the strictly c-optimal algorithm (except for $c-1$ such accesses[2]) and possibly more.

Formally, let $c \geq 1$ be a natural number. $A \in \mathcal{A}_{\text{TOP-K}}$ is strictly c-optimal w.r.t. $\mathcal{D}$ if for every $A' \in \mathcal{A}_{\text{TOP-K}}$, and for every $I \in \mathcal{D}$ s.t. A' halts when given I as input, $|\, nodes(A, I) - nodes(A', I)| < c$.

Clearly, if A is strictly c-optimal then it is c-optimal, and specifically if it is strictly 1-optimal then it is 1-optimal. The need for this stronger term of strict c-optimality will become evident later on, when we discuss optimality of interactive top-k algorithms.

3.2 Optimal Top-k algorithms

As we next show, the existence of a c-optimal top-k algorithm depends on properties of the path weight function W_p. First, we may show that in general, for any given natural number c, no algorithm is c-optimal, even for the restricted input domain $\mathcal{I}_{fin}$.

THEOREM 3.2. *For every natural number c, there exists no c-optimal algorithm within $\mathcal{A}_{TOP\text{-}K}$ w.r.t. $\mathcal{I}_{fin}$.*

PROOF. Let c be a natural number. Assume, in contradiction, that some algorithm $A \in \mathcal{A}_{\text{TOP-K}}$ is c-optimal. Consider $k = 1$, namely we look for the top-1 interaction. Let $n = c^2$. We gradually construct an ASpec s, consisting (at the end of the construction) of $2n$ nodes. At each step we obtain an intermediate ASpec, execute A on it, and change parts of its structure (only those parts that were not observed by A) according to the execution of A. The construction is repeated until we obtain an ASpec for which A performs $\Theta(n)$ $(= \Theta(c^2))$ data accesses, but some other algorithm $A' \in \mathcal{A}_{\text{TOP-K}}$ performs only $O(log(n)) = O(2{\cdot}log(c))$ - a contradiction to the c-optimality of A.

At the beginning $s = (r, \psi, acc)$, where $\psi(r) = \{N_1, N_2\}$, and we set at first $\psi(N_1) = \psi(N_2) = \{a\}$, and only a is accepting. $W_e(e) = 1$ for every edge e, and we use multiplication for *aggr*. We execute A over s, and observe on which node A invoked ψ first: N_1 or N_2. Since $A \in \mathcal{A}_{\text{TOP-K}}$, this choice *does not depend on* $\psi(N_1)$ *and* $\psi(N_2)$. Thus, we change s as follows: if A chose to invoke $\psi(N_1)$ first, we set $\psi(N_1) = \{N_3, N_4\}$, for two new nodes N_3, N_4, while keeping $\psi(N_2)$ intact (and symmetrically if A chose to invoke $\psi(N_2)$). We also set $\psi(N_3) = \psi(N_4) = \{a\}$. Next, A has three choices for the next invocation of ψ: N_2 (or N_1), N_3, and N_4. For the chosen node N', we again set $\psi(N') = \{N_5, N_6\}$, for two new nodes N_5, N_6, and $\psi(N_5) = \psi(N_6) = \{a\}$. We repeat this process for the node N' chosen next, until obtaining an underlying graph with $2n$ nodes.

Observe that when A is executed over the constructed ASpec s, it invokes ψ on n distinct nodes. We argue that there exists an algorithm $A' \in \mathcal{A}_{\text{TOP-K}}$ that, when executed on s, performs $O(log(n))$ invocations of ψ. To show this, we first observe that there exists an accepting interaction p_{short} in s of length $O(log(n))$. This follows from the fact that (1) without the accepting state a, $graph(s)$ has the shape of a binary tree, (2) the length of the shortest root-to-leaf path in a binary tree is at most logarithmic in the tree size, and (3) a is pointed exactly by all the leaves of the tree.

We show two lemmas, as follows:

LEMMA 3.3. *There exists an accepting interaction p_{short} in $inter(s)$, with $W_p = 1$, such that the length of p_{short} is at most $log(n)$.*

PROOF. Assume by contradiction that *every* accepting interaction in $inter(s)$ consists of more than $log(n)$ nodes. Since every node in $graph(s)$ has two descendants, each containing a distinct nodes, we obtain that there are over $2^{log(n)} = n$ distinct nodes in s, in contradiction to the way s was constructed. □

LEMMA 3.4. *There exists an algorithm $A' \in \mathcal{A}_{TOP\text{-}K}$ that, when executed on s to find its top-1 interaction, invokes ψ only on nodes appearing in p_{short}.*

We defer the proof of this Lemma to the end of this section, as it will make use of tools introduced below. This concludes the proof of Theorem 3.2. □

Nevertheless, we show that in some important sub-domains, c-optimal algorithms do exist. For that, we define the notion of i-strongly monotone path weight functions:

DEFINITION 3.5. *Let i be a natural number. W_p is i-strongly monotone w.r.t. an ASpec s, if for every node n in $graph(s)$, there are at most i interactions starting at n that have equal W_p values. I.e., for every node n and every $w \in \mathcal{W}$, $|\, \{p | p \in inter(s)\,, start(p) = n, W_p(p) = w\}\, | \leq i$.*

Such functions occur naturally in practice, with a relatively low value of the bound i. For instance, if the weight function stands for popularity of choices (e.g. for Data Cleaning), 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 in an online shopping setting, 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, as we mentioned above, such paths are typically "not interesting" for analysis purposes, and may be pruned in pre-processing.

Denoting by $\mathcal{I}_{mono(i)}$ the class of all input instances in $\mathcal{I}_{fin}$ where W_p is i-strongly monotone, we obtain:

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

THEOREM 3.6. *Let c be a natural number. There exists an algorithm A s.t. A is* strictly c-optimal*, w.r.t. the input domain $\mathcal{I}_{mono(i)}$, when $i = c + 1$.*

PROOF. Consider the following simple TOP-K-ALGO algorithm that follows the lines of the well-known A^* algorithm [19] (that, in turn, follows Dijkstra's algorithm [11]).

TOP-K-ALGO. The algorithm maintains two global priority queues, *Frontier* and *Out*, of interactions along with their W_p values. The interactions in each queue are ordered by W_p (best to worst), with accepting interactions having priority over non-accepting ones in case of equal weights. Initially, *Frontier* contains only a single interaction, consisting of the input node r, and *Out* is empty. The algorithm operates iteratively, where in each iteration it pops the top interaction p from *Frontier*. If p is accepting, then the algorithm inserts it to *Out*. If *Out* contains k interactions, the algorithm halts and outputs them. Otherwise it adds to *Frontier* all interactions p' s.t. $p \to p'$ (if there are such). These interactions are obtained from p by considering the last node n of p, and concatenating each $n' \in \psi(n)$ to p.

Whenever the algorithm invokes $\psi(n)$ for a node n, the results set $\psi(n)$ is recorded in memory. Consequently, if at some later point of the computation, the algorithm again requires $\psi(n)$ for the same node n, it can use the recorded results rather than performing another invocation of ψ.

To conclude the proof, we will show that TOP-K-ALGO is strictly c-optimal with respect to $\mathcal{I}_{mono(i)}$. This will be proved by assuming by contradiction that there is a correct algorithm A that performs c less data accesses than TOP-K-ALGO for some input instance I, and showing that it leads us to a contradiction.

Formally,

LEMMA 3.7. *For every natural number c,* TOP-K-ALGO *is* strictly c-optimal *w.r.t. $\mathcal{I}_{mono(i)}$, when $i = c + 1$.*

PROOF. Assume by contradiction that TOP-K-ALGO is not strictly c-optimal, then there exists some input instance $I = (s, r) \in \mathcal{I}_{mono(i)}$, where s is an ASpec and r its root node, and an algorithm A' such that $|\, nodes(\texttt{TOP-K-ALGO}, I) - nodes(A', I)| \geq c$. Let p_{term} be the worst solution (with lowest weight) that appears in *Out* upon termination, and let $w_{term} = w_p(p_{term})$. We note that whenever TOP-K-ALGO pops an interaction p from *Frontier*, then $w_p(p) \geq w_{term}$. As $I \in I_{mono(i)}$, the number of such p for which $w_p(p) = w_{term}$ is at most i (recall that $i = c + 1$).

First, assume there exists at least one interaction p s.t. $w_p(p) > w_{term}$, but A' does not access $n = end(p)$ (while TOP-K-ALGO, as we noted, does call $\psi(n)$). Let w^* be a weight such that $w_{term} < aggr(w_p(p), w^*) < w_p(p)$ (As *aggr* is *continuous*, there exists such w^*). We construct a new ASpec s' which is exactly the same as s, except for setting $\psi(n) = a$, for a new accepting node a, and setting the edge weight $W_e(n, a) = w^*$. Now the corresponding interaction has a weight higher than w_{term} and should be outputted. But it will be missed by Algorithm A' in contradiction to A' being correct.

Consequently, we can assume that the c nodes accessed by TOP-K-ALGO but not by A' are ends of (non-accepting) paths with weight w_{term}. This means that p_{term} is the only accepting path with weight w_{term}, and must be also outputted by A'. By the order in the priority queue *Frontier*, we know that p_{term} can only be inserted to frontier *after* the last non-accepting path with weight w_{term}, p' was popped out of the queue; otherwise, TOP-K-ALGO would pop and output p_{term} before accessing all the ends of all non-accepting paths of weight w_{term}. Thus, it must be the case that $p' \to p_{term}$. But then, since A' outputs p_{term}, it must have called $\psi(end(p'))$, an thus there can be at most $c - 1$ nodes accessed by TOP-K-ALGO and not by A'. □

This concludes the proof of theorem 3.6. □

We note that it is easy to show that the algorithm TOP-K-ALGO used in our positive results above is sound for $\mathcal{I}$, and complete for $\mathcal{I}_{fin}$. We omit the proof for space constraints.

We use $\mathcal{I}_{mono}$ as another denotation for $\mathcal{I}_{mono(1)}$, where *every* two distinct interactions $p \neq p' \in inter(s)$ that start at any given node n do not bear the same weight, which gives a *strongly monotone* weight fuction. Since TOP-K-ALGO is strictly 1-optimal w.r.t. $\mathcal{I}_{mono(2)} \supset \mathcal{I}_{mono}$, we obtain:

COROLLARY 3.8. *There exists an algorithm $A \in \mathcal{A}_{TOP\text{-}K}$ that is* strictly 1-optimal *with respect to $\mathcal{I}_{mono}$.*

The algorithm TOP-K-ALGO is *not* strictly $(c-1)$-optimal w.r.t. $\mathcal{I}_{mono(i)}$, $i = c + 1 > 1$. We next show that this is inevitable, as there exists no such strictly $(c-1)$-optimal algorithm.

THEOREM 3.9. *For a natural $c > 1$ and $i = c+1$, no algorithm within $\mathcal{A}_{TOP\text{-}K}$ is strictly $(c-1)$-optimal w.r.t. $\mathcal{I}_{mono(i)}$.*

PROOF. Let $k = 1$. We gradually construct an input I, as follows. Let the root of the ASpec s be r, and let $(r, n_1), ...(r, n_c)$ be edges with some weight w, going out of r, where $n_1, ..., n_m$ are non-accepting nodes. Assume by contradiction that there exists an algorithm A which is strictly $(c-1)$-optimal w.r.t. $\mathcal{I}_{mono(i)}$. Assume that after calling $\psi(r)$, A calls $\psi(n_j)$. Then we set $\psi(n_j) = \{n'_j\}$, and set some weight for (n_j, n'_j) such that $w_p([r, n_j, n'_j]) < w$. We repeat this construction for every n_j A accesses, except for the last one. Assume w.l.o.g. that A accesses n_c last. Then we set $\psi(n_c)$ to consist of a new accepting node n_{c+1}, such that $w_p([r, n_c, n_{c+1}]) = w$. $[r, n_c, n_{c+1}]$ is then the top-1 interaction that A should output.

Note that $(s, r) \in \mathcal{I}_{mono(i)}$, since there are $i = c + 1$ paths of weight w (and we can set every other path going from r to have a different weight lower than w). A accesses $c+1$ nodes totally, whereas an algorithm A' optimal for (s, r) would call ψ only on r and n_c. Thus $|\, nodes(A, I) - nodes(A', I)| = c - 1$, and A is not strictly $(c-1)$-optimal w.r.t. $\mathcal{I}_{mono(i)}$. □

Note that in particular, in input domains for which exists a strictly 1-optimal algorithm, every 1-optimal algorithm is also strictly 1-optimal. In combination with Theorem 3.9, we get the next corollary.

COROLLARY 3.10. *No algorithm within $\mathcal{A}_{TOP\text{-}K}$ is 1-optimal w.r.t. $\mathcal{I}_{mono(i)}$, for $i > 2$.*

Proof of Lemma 3.4. Now that we have seen Algorithm TOP-K-ALGO, we are ready to present the proof of Lemma 3.4 and conclude this section.

PROOF. Recall that the algorithm TOP-K-ALGO described above maintains a priority queue *Frontier* where interactions

are ordered by their w_p values; the algorithm does not specify an order in-between two equal weight interactions that are both not accepting; any variant that sets such order is also correct. In particular, as w_p values of all interactions with s are equal, the variant that, when invoked on s, invokes ψ exactly on the nodes of p_{short}, is a correct one, thus $\in \mathcal{A}_{\text{TOP-K}}$. □

4. INTERACTIVE TOP-K

In the previous section we have considered TOP-K, namely a single computation of top-k interactions starting from a given node. We now turn to study ITOP-K, i.e. the on-going process of computing top-k recommendations following each user choice.

To that end, we use $\mathcal{A}_{\text{ITOP-K}}$ to denote the class of all deterministic algorithms that receive as input an ASpec s along with a weight function, a number k and an interaction $p \in inter(s)$, node by node, and no additional input, and solve ITOP-K. Furthermore, we have defined above $\mathcal{I}$ as the domain of all possible inputs for TOP-K. We use $\mathcal{I}^{inter}$ to denote the domain of all possible inputs for ITOP-K. $\mathcal{I}^{inter}$ consists of pairs of (instance from $\mathcal{I}$, interaction).

Similarly, for every input domain $D \subseteq \mathcal{I}$ we use D^{inter} to denote the input domain obtained by accompanying each input of D with some interaction. Note that $D^{inter} \subseteq \mathcal{I}^{inter}$.

The cost of an interactive top-k algorithm is measured, similarly to the cost of top-k algorithms, by the data accesses it performs. We thus adopt the cost model and optimality notions defined in Section 3, and analyze the existence of a c-optimal algorithm in $\mathcal{A}_{\text{ITOP-K}}$, with respect to these notions. Clearly, for an input domain D where no c-optimal solution exists for TOP-K, no c-optimal solution exists for ITOP-K over D^{inter}. It remains to study, thus, the existence of a c-optimal ITOP-K algorithm in domains D^{inter} s.t. a c-optimal TOP-K algorithm exists for D.

4.1 Optimality

We first show that the existence of even a 1-optimal algorithm for TOP-K does not imply the existence of a c-optimal algorithm for ITOP-K. This holds even for sub-domains of $\mathcal{I}_{fin}$.

THEOREM 4.1. *There exists an input domain $D \subseteq \mathcal{I}_{fin}$ of infinite cardinality s.t. there exists an algorithm in $\mathcal{A}_{TOP\text{-}K}$ that is* 1-optimal *w.r.t. D, but, for every natural number c,* no algorithm *in $\mathcal{A}_{ITOP\text{-}K}$ is* c-optimal *w.r.t. D^{inter}.*

PROOF. We use in our construction weight functions such that $aggr = \times$, and weights are in $[0, 1]$. Given an ASpec s and a node n in s, we denote by $ZeroPaths(s, n)$ the set of all maximal (as defined below) accepting paths $p \in inter(s)$ reachable from n s.t. for the first edge e of p, $W_e(e) = 0$ (note that, in particular, $W_p(p) = 0$). "Maximal" means here that p is not a sub-path of any other path satisfying the above criteria. D is then the domain of all ASpecs s, such that (1) for every node n, all paths in $ZeroPaths(s, n)$ are of equal lengths, and (2) there are no two paths $p \neq p'$ outgoing n such that $W_p(p) = W_p(p') \neq 0$. For this domain, we will now first show the existence of a 1-optimal top-k algorithm, then show that for every natural number c there exists no c-optimal interactive top-k algorithm. This is proved in the following two lemmas.

LEMMA 4.2. *There exists a 1-optimal algorithm within $\mathcal{A}_{TOP\text{-}K}$ over D.*

PROOF. We construct a slight adaptation of TOP-K-ALGO given in the previous section: in case of "ties", i.e. two interactions with equal weights, their order in *Frontier* was arbitrary. In contrast, in case of weight ties we now give priority, in *Frontier*, to interactions containing more nodes (and in case of tie also of length, choose their order arbitrarily). The obtained algorithm is 1-optimal within $\mathcal{A}_{\text{TOP-K}}$ with respect to D. Consider two cases. First, assume that the top-k interactions do not contain an interaction of weight 0. In this case, our algorithm never pops an interaction of weight 0 from *Frontier*. From the properties of D, this means that in all parts of the ASpec that the algorithm observed, W_p is in fact strongly monotone. Thus in this case, following Lemma 4.2 our algorithm is 1-optimal (with respect to all algorithms that did not traverse a 0-weighted edge; an adversary algorithm that did traverse a 0-weighted edge, of course performed only worse).

Moreover, as TOP-K-ALGO is strictly 1-optimal over strongly monotone weight functions, it follows that every algorithm must have made at least the same data accesses made by our algorithm up until reaching a 0-weight path. From the properties of TOP-K-ALGO it further follows that if the algorithm does pop an interaction of weight 0, this means that there exists such 0-weight interaction within the top-k results. Thus that every algorithm would have to traverse such 0-weighted interaction. As our algorithm orders interactions, in *Frontier*, by their length, we are guaranteed to consider first the longest interactions of weight 0; as all such accepting 0-weighted interactions are of equal length, traversing longest interactions first guarantees minimal data accesses until reaching an accepting node. □

Finally, the following lemma holds.

LEMMA 4.3. *For every natural number c, there is no c-optimal algorithm within $\mathcal{A}_{ITOP\text{-}K}$ over D.*

PROOF. For each natural number n, consider the following ASpec $s = (r, \psi, acc)$. $graph(s)$ is a balanced tree, where r is the tree root, each node has n^2 children, and the tree height is n. the leaves of $graph(s)$ are the accepting nodes. The weight W_e of every edge is 0, $aggr = \times$, and we seek for top-1 interaction. Given an interaction $p \in inter(s)$, the "best" algorithm for s, p would perform just n data accesses (this is an algorithm that happens to computes p as the top-1 interaction, and thus does not require additional data accesses as the interaction continues). Assume, by contradiction, that some algorithm $A \in \mathcal{A}_{\text{ITOP-K}}$ is c-optimal; in particular it is c-optimal for s and for every interaction $p \in inter(s)$. We construct an interaction p gradually, according to the algorithm behavior. At the first step (r is the first node of p), if A invokes ψ for all children of r, then it performs at least n^2 data accesses and is not c-optimal. Otherwise, let $m_1 \in \psi(r)$ be a node s.t. A does not invoke $\psi(m_1)$. We set m_1 as the next node of p; again, A either accessed all children of m_1 thus making at least n^2 steps, or there exists a node m_2 not accessed, etc. Consider $I = (r, m_1, ..., m_n)$. When executed on (s, p), in each step of the interaction A is given a node that it has not accessed before and thus must compute a full top-1 path starting from this node. In the i'th step, the length of such a path is $n - i$

and thus A must perform at least $n-i$ data accesses, leading to a total of $\Omega(n^2)$. Recall that there is a correct algorithm making just n accesses for the same input instance, thus A is not c-optimal. □

This concludes the Theorem proof. □

In contrast, we may show that the existence of a *strictly* 1-optimal TOP-K algorithm guarantees the existence of a 1-optimal ITOP-K algorithm. In fact, it also assures the existence of a *strictly 1-optimal* such algorithm.

THEOREM 4.4. *For any input domain $D \subseteq I$, if there exists a* strictly 1-optimal TOP-K *algorithm w.r.t. D, then there exists a strictly 1-optimal algorithm* ITOP-K *algorithm w.r.t. D^{inter}.*

PROOF. The proof builds on the fact that any ITOP-K algorithm must perform (at least) all data accesses the strictly 1-optimal TOP-K algorithm would perform for the individual nodes on the interaction path. Thus it may not benefit from replacing data accesses done to compute the top-k interactions at a given node by others performed in previous steps.

Consider a simple interactive top-k algorithm B that, for an ASpec s and an interaction $p = (n_1, ..., n_m) \in inter(s)$, repeatedly runs the strictly 1-optimal top-k algorithm A and keeps in memory all results of data accesses performed by A. We claim that B is strictly 1-optimal. Thus, we show that for each ASpec s and an interaction $p \in inter(s)$, for each node n such that B invokes $\psi(n)$, so does every correct algorithm B' applied over (s, p) as input. Assume the existence of a contradicting node n, and assume that $\psi(n)$ was invoked at the i'th step. Now consider a *top-k* algorithm A' that makes the exact invocations of ψ as B' does, and in particular, invokes ψ on all nodes of p. A' computes in particular *top-k*(s, n_i) without invoking $\psi(n)$, while A invoked $\psi(n)$ in this computation, in contradiction to the strict 1-optimality of A. □

Combined with Theorem 3.6 and Corollary 3.8, we obtain the following.

COROLLARY 4.5. *There exists a strictly 1-optimal* ITOP-K *algorithm for $\mathcal{I}^{inter}_{mono}$ and $\mathcal{I}^{inter}_{mono(2)}$.*

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 interactive top-k algorithm. Observe that this is slightly weaker than what we have previously seen for the case of 1-optimality (there strict 1-optimality for top-k implied *strict* 1-optimality for interactive top-k). Finding sufficient conditions for a *strictly* c-optimal ITOP-K algorithm is an open problem.

THEOREM 4.6. *For any input domain $D \subseteq I$, if there exists an algorithm $A \in \mathcal{A}_{TOP\text{-}K}$ that is* strictly c-optimal *over D for some natural number c, then there exists an algorithm $B \in \mathcal{A}_{ITOP\text{-}K}$ that is c-optimal over D^{inter}.*

PROOF. We use the same algorithm B from the proof of Theorem 4.4, this time invoking a strictly c-optimal algorithm A. Given an interaction $[n_0, ...n_m]$, consider the set N_i of nodes for which A invokes ψ, when invoked to find *top-k*(s, n_i). As A is strictly c-optimal, every algorithm that computes *top-k*(s, n_i) invokes ψ on some N'_i s.t. $|N_i - N'_i| < c$. The overall number of nodes for which B invokes ψ but not every correct algorithm does is bounded by $m \cdot c$ (for the rest of accessed nodes, the proof proceeds as in Theorem 4.4). Now, any algorithm must perform at least m steps (as m is the length of the interaction), thus B is c-optimal. □

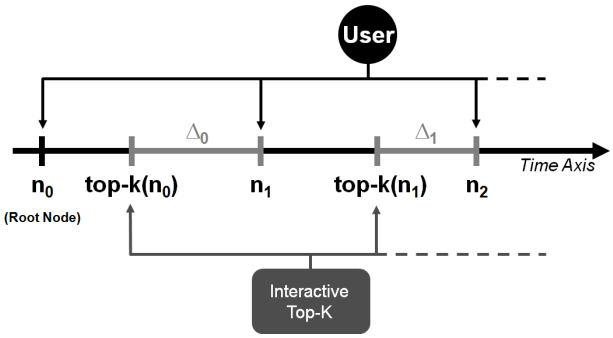

Figure 2: Interactive Session

Combined with Theorem 3.6, we obtain:

COROLLARY 4.7. *For every natural $c \geq 1$, there exists a c-optimal* ITOP-K *algorithm over $\mathcal{I}^{inter}_{mono(i)}$, with $i = c + 1$.*

5. 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 real-life interactive Web applications, the *timing* of computations is also crucial: there are intervals of time during the interaction where the system is idle, namely in the time interval where 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*.

Figure 2 depicts an interaction on a time axis, highlighting the time frame allocated for lookahead computations; Intuitively, this is the time in-between the retrieval of *top-k*(s, n_i) and receiving the next input node n_{i+1}. Δ_i is the number of data accesses that the look-ahead algorithm may perform during that time. It aims that the data accesses that it performs, are indeed used by the interactive-top-k algorithm in later stages of the interaction.

A lookahead algorithm LA aims to assist an algorithm $A \in \mathcal{A}_{\text{ITOP-K}}$ 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.

To decide which instructions to issue, the lookahead algorithm has access to the ASpec $s = (r, \psi, acc)$. Similarly to algorithms in $\mathcal{A}_{\text{ITOP-K}}$, 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, $i > 0$, we are given a node

$n_i \in \psi(n_{i-1})$; our 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.)

EXAMPLE 5.1. *Re-consider Example 2.4; say that we have presented the user with the results of computing top-1*(s, n_0)*, but she has not yet made her next interaction choice. We may use the time before she submits her next choice to perform additional computations, which may be required in later stages of the interaction. For instance, if we know that a Sony TV is popular among users, it may be worthwhile to start performing an analysis of top-k*(s, n_{10})*. We will interrupt this analysis when the actual user choice is submitted; but in case this choice is indeed of Sony TV, the computations already done will be utilized by the algorithm to promptly respond to the user choice. Alternatively, we may divide the available computation steps among several options, to be helpful for a larger number of interactions (at the cost of being less helpful for each individual interaction). For example, we may perform some partial computations for both top-k*(s, n_{10}) *and top-k*(s, n_{12})*.*

5.1 Performance Measures

We define the class $\mathcal{A}_{\text{LA}}$ of all deterministic lookahead algorithms that are given the same input as depicted above. We consider two properties of lookahead algorithms, namely its *resources* 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 time 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. As Δ_i is unknown to the lookahead algorithm, in practice, the instructions (and further operations, for this step) of LA are "cut" once the total number of data accesses incurred reaches Δ_i.

Utilization. The *utilization* of an algorithm LA captures the quality of its instructions, namely how many useful computations (data accesses) it preempted. To define this formally, we use the following notation. Consider an algorithm $A \in \mathcal{A}_{\text{ITOP-K}}$ (with no lookahead used) when operating on an ASpec s and an interaction $p = [n_0, ..., n_t]$ with s. We use $nodes^{>i}(A, s, p)$, for $i = 0, ..., t-1$, to denote the set of all nodes for which A invokes ψ, after receiving n_{i+1} and up until the end of the interaction. Now, given also a lookahead algorithm LA, each instruction of LA causes A to perform some preempted sequence of data accesses; we denote the set of such preempted 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 5.2. *Let $LA \in \mathcal{A}_{LA}$ and $A \in \mathcal{A}_{ITOP\text{-}K}$. Given an ASpec s and some interaction $p = [n_0, ..., n_t]$ with s, the utilization of LA w.r.t. A, s and p, is defined as* $util(LA, A, s, p) = \sum_{i=0,...,t-1} \left| pre^i(LA, A, s, p) \bigcap nodes^{>i}(A, s, p) \right|$.

We note that the utilization of LA also implicitly depends on the time intervals allowed for lookahead. When we shall compare below utilizations of different lookahead algorithms, we thus always assume that they are both allowed the same lookahead time. We next study a number of possible optimality measurements for lookahead algorithms, based on the notion of utilization. We denote bellow 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 *all possible* algorithms $A \in \mathcal{A}_{\text{ITOP-K}}$, and all ASpecs s and interactions $p \in accInter(s)$. This is appealing, as such algorithm will minimize the overall response time for all accepting interactions. We thus say that, given a natural number 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 $LA' \in \mathcal{A}_{\text{LA}}$.

Maximal* Expected *Utilization. As we show below, devising such a c-optimal algorithm is impossible. Thus, 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 function by *pLikelihood*. Based on this function, we then define the *expected* utilization of a lookahead algorithm, where the expectancy is over all accepting interactions with the given application.

DEFINITION 5.3. *The* expected *utilization of lookahead algorithm LA, w.r.t. an ASpec s and $A \in \mathcal{A}_{ITOP\text{-}K}$ 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 number c if $c \cdot E[util](LA, A, s) \geq E[util](LA', A, s)$ for every A, s and every $LA' \in \mathcal{A}_{\text{LA}}$.

Fairness. While an exp-c-optimal algorithm serves well the "average" user, it may be *unfair*, in the sense that some users benefit more than others. To ensure a minimal utilization over all users, we consider *fairness-optimal* algorithms. To that end, we define

$MinUtil(LA, A, s) = min_{p \in accInter(s)} \frac{util(LA, A, s, p)}{|p|}$. This is the utilization of LA w.r.t. the "worst" interaction p with s, averaged by the interaction length to avoid bias towards short interactions. An algorithm LA is *fairness-c-optimal* for a given natural number c if $c \cdot MinUtil(LA, A, s) \geq MinUtil(LA', A, s)$ for every A, s and $LA' \in \mathcal{A}_{\text{LA}}$.

5.2 Negative Optimality Results

We next discuss the existence of optimal lookahead algorithms in $\mathcal{A}_{\text{LA}}$, with respect to the optimality notions defined above. We will show that none of the above optimality notions may be achieved in the general case. This holds even under restrictive assumptions on the input, that do allow for 1-optimal top-k (and interactive top-k) algorithms. More concretely, let $\mathcal{I}^{inter}_{MonoTree}$ be the class of all input ASpec

whose weight function is strictly monotone, and furthermore whose graph is tree-shaped. We show that even in $\mathcal{I}^{inter}_{MonoTree}$ no c-optimal and no fairness-c-optimal lookahead algorithm exists, and no exp-1-optimal algorithm exists. The existence of an exp-c-optimal algorithm for the general case remains an open problem.

Formally,

THEOREM 5.4. *For every natural c, there is no c-optimal algorithm in $\mathcal{A}_{LA}$, w.r.t. $\mathcal{I}^{inter}_{MonoTree}$.*

PROOF. Assume, by contradiction, the existence of a c-optimal algorithm LA. I.e., there exists a c such that $util(LA, A, s, p) \geq c\cdot util(LA', A, s, p)$ for every lookahead algorithm LA'. We use for A our interactive top-k algorithm from the proof of Thm. 4.4 above, using `TOP-K-ALGO` for A. We construct an ASpec $s = (r, \psi, acc)$ and an interaction $p \in inter(s)$ of length c, gradually, according to the operation of LA. Initially, we set $\psi(r) = \{n, n^0, ..., n^{\Delta_0}\}$ (Δ_i values are unknown to LA, but we may still use it in our construction), $\psi(n) = \{m_1, m_2\}$, and only m_1 is accepting. $W_e(r, n) = 0.75$, $W_e(n, m_1) = 0.99$,$W_e(n, m_2) = 0.01$ and the sum of weights of all $W_e(r, n^i)$ is 0.25 (while they are all distinct). The interaction starts at $n_0 = r$, with a top-1 computation that only accesses r, m_1. As the cost of LA may not exceed Δ_0, there exists some node $n_1 \neq n \in \psi(r)$ such that, at the first step of p, LA neither accesses n_1 nor instructs the top-k algorithm to operate on it. We then set the next node of the interaction to be n_1, and change s such that $\psi(n_1)$ is a copy of $\psi(r)$, with new identifiers assigned to nodes, and corresponding weights. At the next point of the interaction, denote the node that LA did not accessed by n_2 and so on, up to $n_{|p|}$. The utilization of LA w.r.t. p is 0; a lookahead algorithm LA' that instructs A to execute *top-1*(n_i, ∞) at each point bears utilization of $2\cdot|p| = 2\cdot c$, in contradiction to the assumption of c-optimality. □

Similarly,

THEOREM 5.5. *For every natural number c, there is no* fairness-c*-optimal algorithm in $\mathcal{A}_{LA}$, w.r.t. $\mathcal{I}^{inter}_{MonoTree}$.*

PROOF. The construction in this proof is based on the fact that the more outgoing paths some node has, the less top-k computations can be done for each of them. Thus, a lookahead algorithm which allocates more computation steps for the more "branching" nodes is better in terms of fairness. However, the "branching" of uninspected nodes is not known to the algorithm, and thus it cannot be fairness-c-optimal for every ASpec.

Assume by contradiction that a lookahead algorithm LA is fairness-c-optimal for some natural number c, i.e. that $MinUtil(LA, A, S) \geq c\cdot MinUtil(LA', A, S)$ for each lookahead algorithm LA', interactive top-k algorithm A and ASpec s. We consider an ASpec $s = (r, \psi, acc)$ where $\psi(r) = \{n_1, ..., n_{2c}\}$, and assume first that LA does not invoke ψ.

W.l.o.g., we may assume that LA performs a round-robin of instructions over $\{n_1, ..., n_c\}$, i.e. its instruction sequence are of the form $((n_1, 1), ...(n_{2c}, 1), (n_1, 1), ...)$ (it is easy to show that other allocations will not increase the minimal utilization in some cases). We set $|\psi(n_i)| = 1$ for $i = 1, ..., 2c-1$, and $|\psi(n_c)| = 2c$; all nodes in $\psi(n_i)$ are accepting (LA does not invoke ψ, thus is indifferent to its construction). The minimal utilization of LA is $\frac{\Delta}{2c}$. However, an algorithm that "guesses" that $\psi(n_2c)$ is a large set, would allocate more such steps to $\psi(n_c)$ in advance. Namely, the best algorithm for this case would allocate $x\cdot\Delta$ data accesses to n_c and $(1-x)\cdot\Delta$ data accesses divided equally by n_i, $i = 1, ..c-1$. The optimal value for x is $\frac{2c-2}{2c-1}$ (computation omitted for brevity), yielding a minimal utilization of $(\frac{2c-2}{2c-1} + \frac{1}{c})\cdot\Delta$. This is better by a fraction of more than c than the minimal utilization of LA. If LA does invoke ψ, we may construct an ASpec whose graph is an arbitrary long chain, followed by a construction as above, and we may again show that LA is not fairness-c-optimal. □

Last, there is no exp-1-optimal algorithm in $\mathcal{A}_{\text{LA}}$, even under the above restrictions.

THEOREM 5.6. *For every natural number c, there is no* exp-*1-optimal algorithm in $\mathcal{A}_{LA}$, w.r.t. $\mathcal{I}^{inter}_{MonoTree}$.*

PROOF. In this proof we use the fact that when the following time intervals are long, an algorithm that performs computations for the possible immediate next user choices is better; and when the following time intervals are short, an algorithm which performs computations for the most likely user choice (which are not necessarily a possible immediate next choice) is better, because there will not be enough time to make these computations later on.

Assume the existence of an exp-1-optimal lookahead algorithm LA, and consider an ASpec $s = (r, \psi, acc)$ such that $\psi(r) = \{m, n\}$, $\psi(n) = \{n_1, n_2\}$, $\psi(m) = \{m_1, m_2\}$, $\psi(n_1) = \psi(n_2) = \psi(m_1) = \psi(m_2) = \{a\}$, and only a is accepting. The likelihood function assigns a likelihood of 0.9 to (r, n), 0.1 to (r, m), 0.9 to (n, n_1), 0.1 to (n, n_2), 0.9 to (m, m_1), 0.1 to (m, m_2), and 1 to (x, a) for every x.

The weight function used for top-k computation is some arbitrary strongly monotone weight function. It is easy to show that at the first step of the interaction, the instructions of LA must start by (n, ∞), i.e. fully compute *top-k*(s, n) (otherwise we may improve the expectancy). Denote by K the number of data accesses such computations incur. We set the time for the first lookahead $\Delta_0 = K + 2$ (i.e. there is time for two additional steps). Let us review possible action courses the algorithm might take:

1. LA may instruct the top-k algorithm to compute 2 steps of *top-k*(s, m), in which case we consider the case where $\Delta_j = 0$ for all $j \geq 1$, i.e. there is no additional lookahead time. The expected lookahead of LA in this case is $0.9\cdot K + 0.1\cdot 2$. A different lookahead algorithm that would first invoke ψ over n, then instruct the top-k algorithm to compute a single step for n_1, will have an expectancy of $0.9\cdot K + 0.81$ higher than that of LA (note that the expectancy of all accepting interactions starting at n_1 is $0.9\cdot 0.9 = 0.81$).

2. If LA indeed performs $\psi(n)$ followed by instructing the top-k algorithm to compute a single step for n_1, we set $\Delta_j = 100$ for all $j \geq 1$. Now a lookahead algorithm that at the current step execute 2 steps for m, and then in the second step complete the top-k computation for the remaining nodes, has an expectancy that is greater by 0.2.

An algorithm that performs any other 2 steps may easily be shown to be not 1-exp-optimal. □

5.3 Positive Optimality Results

While for c-optimal algorithms it seems difficult to identify natural classes of input, we are able to identify such classes for exp-c-optimal and fairness-c-optimal algorithms.

Exp-c-optimality. Given a natural number 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.

We may show that there exists a natural number i s.t. no lookahead algorithm is c-optimal over $I_{shallow}(i)$ for any value of c. The proof adapts that of Theorem 5.6, while guaranteeing "shallowness" of the obtained specification (details omitted for lack of space). But, in contrast, for exp-c-optimality we can show the following positive result.

THEOREM 5.7. *For every natural number c, there exists an exp-c-optimal lookahead algorithm over $I_{shallow}(c)$*

PROOF SKETCH. We first present a lookahead algorithm, and then prove that it is exp-c-optimal over $I_{shallow}(c)$. As explained in the proof of theorem 5.6, different lookahead instructions are better in different scenarios, and specifically the algorithm must choose between computations for possible user choices that are more "immediate", and those that are more likely. This entails that there are multiple "equally good strategies" i.e. lookahead instruction sets among which none is best for all users. Our proof will show that in $I_{shallow}(c)$, we can bound the number of equally good strategies and split the computation between them in a fair way.

We define the following Lookahead algorithm LA. The algorithm maintains an ordered list $Inters$ of interactions, ordered by their *pLikelihood* value. Denote the input node of LA by n, then the algorithm inserts to $Inters$ all interactions $[n, n']$ such that $n' \in \psi(n)$. While the most popular interaction p_{max} in $Inters$ is also the shortest one, LA instructs $(end(p_{max}), \infty)$, removes p_{max} from $Inters$ and adds to $Inters$ all p' such that $p_{max} \rightarrow p'$. If p_{max} is not the shortest, then we create a set *candidates*, containing, for each interaction length l, the best weighed interaction p_l of length l, if $pLikelihood(p_l) > pLikelihood(p_{l-1})$. By definition of $I_{shallow}(c)$, $|candidates| \leq c$. We can show that while top-k computations for end-nodes of interactions in *candidates* were not completed, then the "best" lookahead algorithm for this case first performs computations for these nodes. We employ a round-robin method for dividing the lookahead computations evenly between all these nodes. We may show that the worst case occurs when the best algorithm performs all computations over a single such node, while we perform only a fraction of $\frac{1}{c}$ of these computations. □

Fairness-c-optimality. We next consider fairness-c-optimality. We assume in the sequel that the ratio between each two time intervals Δ_i, Δ_j allowed for lookahead, is bounded by some natural number d independent of the input size. This is reasonable as these time intervals are dictated by the delay in user choices, and not by the application structure.

Furthermore, for every natural number i, denote $I_{balanced}(i)$ as the domain of all input instances s.t. (1) $graph(s)$ is a tree and (2) $\forall n_1, n_2 \in graph(s) \mid \psi(n_1) \mid - \mid \psi(n_2) \mid < i$ for some natural number i. Namely, the difference in the fan-out of each two nodes is bounded by i. In general, such bound is natural as the fan-out of every node is bounded by some (small) constant, describing the total number of choices that a user can make at a single point of the interaction. For instance, in shopping applications, only a bounded number of choices are simultaneously presented on the screen; moreover, this number is typically low, to avoid visual overload.

The following theorem holds.

THEOREM 5.8. *For every natural number c, there exists a fairness-c-optimal algorithm $LA \in \mathcal{A}_{LA}$ when the input domain is restricted to $I_{balanced}(c)$.*

PROOF. Given a node n, our algorithm LA simply employs a round-robin of instructions for computation of top-k, up to a single data access starting at each node $n' \in \psi(n)$. We claim that LA maximizes $MinUtil(LA, A, S)$ up to a constant fraction for every A and for every $S \in I_{balanced}(i)$. Denote the interaction p_{max} (p_{min}) for which the utilization of LA, divided by the interaction length, is highest (lowest) by p_{max} (p_{min}), and its length by L_1 (L_2). We further denote $|\psi(n_i)| = x_i$ ($|\psi(n'_i)| = x'_i$) where n_i (n'_i) is the i'th node along p_{max} (p_{min}). We claim that the ratio between $util(LA, A, S, p_{max})$ and $util(LA, A, S, p_{min})$ is bounded by c, where c is the balance factor of S. We obtain: $\frac{util(LA,A,S,p_{max})}{util(LA,A,S,p_{min})} = \frac{\frac{\Delta_0}{x_0}+\ldots+\frac{\Delta_{n_1-1}}{x_{n_1-1}}}{n_1} \div \frac{\frac{\Delta_0}{x'_0}+\ldots+\frac{\Delta_{n_2-1}}{x'_{n_2-1}}}{n_2} \leq \frac{\frac{1}{x_0}+\ldots+\frac{1}{x_{n_1-1}}}{\frac{1}{x'_0}+\ldots+\frac{1}{x'_{n_2-1}}} \cdot \frac{n_2}{n_1} \cdot d$, where d is the maximal ratio in-between two time intervals Δ_i and Δ_j.

In the worst case, all x_i values are x and all x'_i values are $c \cdot x$, for some value of x (as c is the balance factor of s). We get $\frac{\frac{1}{x}+\ldots+\frac{1}{x}}{\frac{1}{cx}+\ldots+\frac{1}{cx}} \cdot \frac{n_2}{n_1} \cdot d = \frac{n_1 \frac{1}{x}}{n_2 \frac{1}{cx}} \cdot \frac{n_2}{n_1} = \frac{\frac{1}{x}}{\frac{1}{cx}} = c \cdot d$. Now, it is easy to show that $util(LA, A, S, p_{max}) \geq util(LA', A, S, p'_{min})$ for every algorithm LA', where p'_{min} is the "worst" interaction for LA' (utilization-wise). This concludes the proof. □

6. RELATED WORK

Top-k algorithms were studied in different contexts (e.g. [13, 18, 20, 17, 4, 6]), Specifically, the seminal work of [13] introduced the notions of *instance-optimality*, and had many follow-ups (e.g. [20, 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, 12]). 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* and *fairness*, used to measure the quality of lookahead algorithms, is inspired by notions from the theory of communication networks [15, 14]. As in networks, we aim to optimize the quality of service [14] provided to users; however, research in this area does not consider (interactive) top-k analysis of applications, rendering the problems and algorithms completely different.

7. 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 operating with such memory bound is c-optimal (even for input domains where c-optimality was previously achievable). 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 challenging future work includes the analysis of further restricted, practical cases that allow for c-optimal solutions.

8. REFERENCES

[1] S. Abiteboul, P. Bourhis, and B. Marinoiu. 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] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959.

[12] D. Eppstein. Finding the k shortest paths. In *FOCS*, 1994.

[13] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.

[14] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet*. Addison-wesley, 1998.

[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] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[20] 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.