Next-Step Suggestions for Modern Interactive Data Analysis Platforms
Technical Report

February 12, 2018

Abstract

Modern Interactive Data Analysis (IDA) platforms, such as Kibana, Splunk, and Tableau, are gradually replacing traditional OLAP/SQL tools, as they allow for easy-to-use data exploration, visualization, and mining, even for users lacking SQL and programming skills. Nevertheless, data analysis is still a difficult task, especially for non-expert users. To that end we present `REACT`, a recommender system designed for modern IDA platforms. In these platforms, analysis sessions interweave high-level actions of *multiple types* and operate over *diverse datasets*. `REACT` identifies and generalizes relevant (previous) sessions to generate *personalized* next-action suggestions to the user.

We model the user's *analysis context* using a generic tree based model, where the edges represent the user's recent actions, and the nodes represent their result "screens". A dedicated context-similarity metric is employed for efficient indexing and retrieval of relevant candidate next-actions. These are then generalized to *abstract actions* that convey common fragments, then adapted to the specific user context. To prove the utility of `REACT` we performed an extensive online and offline experimental evaluation over real-world analysis logs from the cyber security domain, which we also publish to serve as a benchmark dataset for future work.

1 Introduction

Data analysis is fundamentally an interactive, iterative process in which a user issues an analysis action (i.e. query), receives a results set, and decides if and which action to issue next. Until recent years, analysis tasks required thorough expertise in SQL and programming, as well as mathematics and statistics. However, since the advent of the Big Data era, the infrastructures and support for Interactive Data Analysis (IDA) have greatly developed: Novel, often web-based platforms such as Tableau, Kibana (ELK), and Splunk, are gradually replacing traditional tools, allowing easy-to-use data exploration, visualization, and mining, even for users lacking knowledge of SQL and programming languages. Yet, IDA is still a difficult process, especially for inexperienced users, as it requires a deep understanding of the investigated domain and the particular context. Users may therefore skip significant analysis actions and overlook important aspects of the data [21].

To assist users, previous work suggested the use of *recommender systems* in the domain of data analysis. These works mostly focus on traditional SQL/OLAP environments, and roughly employ two approaches: *collaborative filtering* [20, 18, 22, 6], and *data-driven* [25, 17, 19, 27]. In the collaborative filtering approach, systems use a repository of (prior) queries of the same or other users, to generate recommendations according to the following assumption: *if users are posing similar sequences of queries, they are likely interested in the same subpart of the dataset.* Hence, queries of one user can be provided as recommendations to the other. In the data-driven approach, systems examine the data at hand and provide recommendations based on the potential interestingness of the query result. (See overview of related work in Section 5.) While all these works make a notable contribution, they scarcely address two challenges, central to current IDA platforms:

(1) IDA platforms facilitate composite analysis processes, interweaving actions of *multiple types* (e.g. SQL-like operators, OLAP multidimensional aggregations, visualization) while providing a simplified syntax. In contrast, previous work typically focuses only on one class of actions, thus not capturing the dependency of one action on the results of previous actions of *different types.*

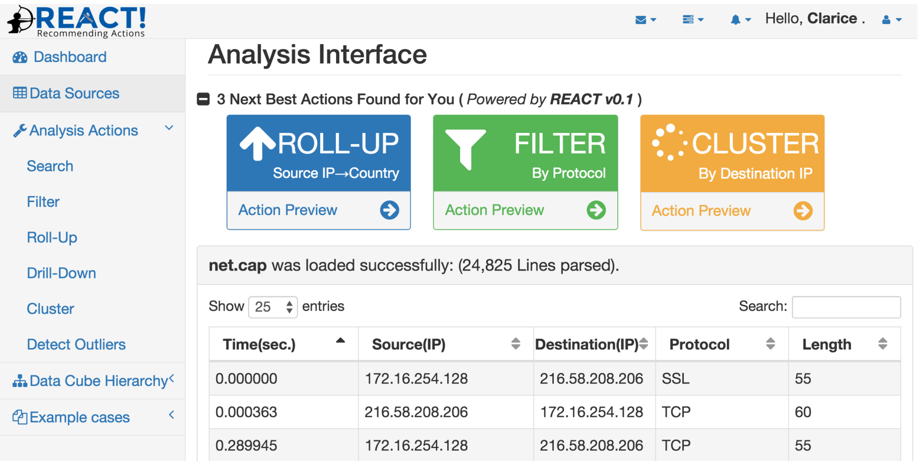
(2) In common IDA business environments, users (even of the same department) often examine *different datasets*, for different purposes. In contrast, previous work generally assumes that users are investigating the same database/cube. Thus, to generate recommendations, these systems search for users who made similar queries on *the exact same dataset*, and use their explicit queries as recommendations. This scenario is mostly impracticable in a varying-dataset IDA environment.

Thus, as advocated in [21], a more holistic approach is required to fully capture the essence of the IDA work, and to be able to provide meaningful recommendations in adequate times. To that end, we present `REACT`, a recommender system designated for modern IDA platforms, which particularly tackles the new challenges that they pose. Given the specific context of the user (e.g., the analysis actions of all types performed thus far by the user, the results obtained, the properties of the data set at hand, etc.), `REACT` processes and adapts previous experience of other analysts working with the same or related datasets, in order to present the user with *personalized* next-step suggestions.

Our key contributions in this work can be summarized as follows.

Efficient Retrieval of Similar Analysis "Contexts" As in existing analysis recommender systems, we also record previous analysis activity by the system's users. Given the state of the current user within an analysis process, we first search for the top-k similar analysis "contexts". The main questions we address are: (1) How does one define "context" in modern IDA platforms? Previous work suggests using either an a-priori profile [12] (containing e.g. the user's role, current assignment etc), her past actions sequence [20], or the data tuples examined [13]. Since modern platforms display more than just the resulted tuples (e.g., also visualizations, data mining results), can such properties be uniformly incorporated in the analysis context? (2) How does one efficiently capture and compare contexts in a way that grasp their commonalities, even when users are examining different datasets?

`REACT` models analysis context using a tree based model, where the edges



Analysis Interface

3 Next Best Actions Found for You (*Powered by* ***REACT v0.1***)

net.cap was loaded successfully: (24,825 Lines parsed).

Time(sec.)	Source(IP)	Destination(IP)	Protocol	Length
0.000000	172.16.254.128	216.58.208.206	SSL	55
0.000363	216.58.208.206	172.16.254.128	TCP	60
0.289945	172.16.254.128	216.58.208.206	TCP	55

Figure 1: Web-Based Analysis UI Powered by REACT

represent the user's recent actions, and the nodes represent their results "screens" (denoted *displays*). Context similarity is then measured using *tree edit distance*, plugged with two novel ground metrics, measuring the distance between analysis actions and displays. Last, contexts are stored in an index structure exploiting the contexts' metric space and the characteristics of our problem settings. From these similar contexts, we efficiently retrieve a set of candidate "next-actions".

Actions Generalization In order to derive an appropriate next-action recommendation to a user, we examine next-actions performed by other users in similar contexts. Note however that these actions may be diverse and operate on distinct datasets, thus they are not useful *as is*, and must be processed to become recommendations. We therefore developed a routine for *generalizing* multiple actions to *abstract actions* (to be formally defined in the paper), conveying common action fragments. Out of many possible generalizations, we derive and present the most relevant ones to the user as next step "suggestions" (See Figure 1 for an example). This process allows `REACT` to overcome the well-known *sparsity* problem in recommender systems [5], where goals of different users rarely overlap.

Real Life Experiments We evaluated our system using real-life IDA logs, acquired from over 50 experienced analysts in the domain of cyber security. We performed both an offline predictive evaluation as well as a live experiment, demonstrating that `REACT` effectively reduces analysis times by 30% on average. Since to our knowledge, there is no publicly available benchmark datasets for modern IDA recommender systems, we publish ours [2], to be used by the research community in future work.

The paper is organized as follows. In Section 2 we describe our model for analysis context, and provide the necessary definitions for the generalization of actions. Section 3 describes the components and workflow of our recommendations framework. Our experiments are detailed in Section 4. Last, we overview related work in Section 5, and conclude in Section 6.

2 Model and Definitions

We begin by describing our model for the current IDA problem setting and analysis context, then lay the groundwork for generating next-action recommendations, in a multiple datasets environment, through a novel action abstraction mechanism.

2.1 A Data Model for Modern IDA Platforms

Dataset A typical IDA process starts when a user loads a particular dataset to an analysis UI (typically web-based). She then executes a series of analysis actions, after each one she examines the results and decides if to execute a new action. We abstractly model a dataset by a triplet $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \mathcal{H})$, where $\mathcal{O}$ is a set of data objects, $\mathcal{A}$ is a domain of attribute names and $\mathcal{H}$ is a (possibly empty) set of semantic hierarchies one per attribute name, that defines an abstraction refinement order on the attribute values. For example, consider the data subset displayed in Figure 1, conveying network traffic data: Semantic hierarchies that can be employed for the *time* and *IP* attributes are: *hours* $\leq$ *minutes* $\leq$ *seconds* and *country* $\leq$ *city* $\leq$ *IP*.

Analysis Actions and Displays, Analysis Tree Inspired by [8], we assume in this work that users perform analysis actions that fall into three main categories: ***data retrieval*** actions, performed to select and filter the relevant data objects for the current assignment (e.g. `FILTER` by 'protocol'$=$"*HTTP*"); ***data representation*** operations are performed to alter the point-of-view of the data objects and include OLAP cube exploration (e.g., `ROLL-UP` 'time' FROM *'seconds'* TO *'hours'*) and ***data mining*** tasks such as clustering and outliers detection (e.g., `CLUSTER` by 'Source_IP').

As common in web applications, we represent the actions and their parameters by a set of key-value pairs (KVP) $\langle k, v\rangle$ s.t. k denotes the parameter type, and v denotes its value. For example, the parameters of the action `FILTER` by 'Protocol'$=$"*SSL*" may be represented by $\{\langle$ 'type',*`FILTER`*$\rangle$, $\langle$'attr',*'Protocol'*$\rangle$, $\langle$'opr',*'='*$\rangle$, $\langle$'term',*'SSL'*$\rangle\}$.

The execution of an analysis action generates a *Results Display* $d = (\langle O, A\rangle, G, M)$, representing a multifaceted "results screen", where: $\langle O, A\rangle$ represents the basic **data layer**, indicating the subsets of data objects $O \subseteq \mathcal{O}$, and attributes $A \subseteq \mathcal{A}$ of the input dataset currently being displayed. G is the **granularity layer**, identifying one abstraction level per attribute (and possibly a corresponding aggregate function), thus representing the current cube point-of-view (e.g. in the current example G can be $\{\langle$*Time*:Minuetes;IP:country $\rangle$, 'avg_length':`AVG`(*length*)$\}$). Finally M is the **mining layer**, associates each object $o_i \in O$ a (possibly empty) set of labels, representing data mining results (e.g. clustering, outliers, and association rules). Other types of actions (and corresponding layers, e.g. the **data visualization layer**) can be added in a similar manner, yet are omitted here for simplicity.

The interactive analysis process in IDA platforms works in intuitively like website navigation - at each point one may invoke an action or backtrack to a previous display and take an alternative navigation path. We thus model the

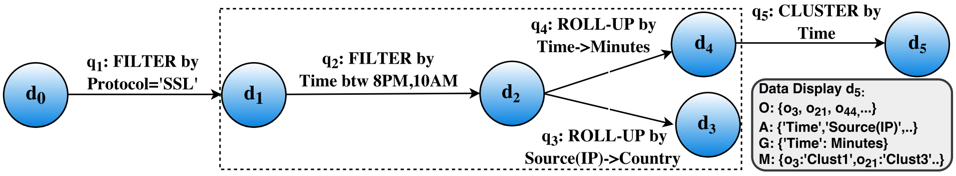

q_1: FILTER by Protocol='SSL'
q_2: FILTER by Time btw 8PM,10AM
q_4: ROLL-UP by Time->Minutes
q_3: ROLL-UP by Source(IP)->Country
q_5: CLUSTER by Time
d_0
d_1
d_2
d_3
d_4
d_5
Data Display d_5:
O: {o_3, o_{21}, o_{44},...}
A: {'Time','Source(IP)',..}
G: {'Time': Minutes}
M: {o_3:'Clust1',o_{21}:'Clust3'..}

Figure 2: Analysis Tree.

IDA process over dataset $\mathcal{D}$ as an ordered labeled tree[1] whose nodes represent displays, and the edges outgoing each node are labeled by the performed action and lead to the resulting display node. The order captures the execution timeline. We use a tree, in contrast to a serial trace, to better model points in the exploration where analysts perform multiple actions on the same display in order to examine multiple facets.

Definition 2.1 (Analysis Tree)**.** *Given a dataset $\mathcal{D}$, and a sequence of analysis actions $q_1, q_2, \ldots, q_m$ of a given user, that starts from an initial display d_0, then generates displays $d_1, \ldots, d_m$, an analysis tree $T = (r, V, E, \prec)$ is a tree containing $m+1$ nodes, s.t. each node $v_i \in V$ represents a display d_i, and each edge $e_i = (v_j, v_i) \in E$ represents an action q_i, operating on d_j, and resulting in d_i. $\prec$ denotes the preorder traversal which captures the execution timeline, namely, $v_i \prec v_j$ iff q_i was executed before q_j*

See Figure 2 for an illustration of an analysis tree (ignore for now the dashed line). A sample of the content of display d_5 is provided in the bottom-right corner.

Analysis Context A *context* is generally defined as the circumstances that form the setting for an event. Inspired by the well known n-gram model, we define the *n-context* of an action q, by the last n displays preceding q. Formally:

Definition 2.2 ($n-$context of analysis action)**.** *For analysis action (edge) q in an analysis tree T, we define its n-context, denoted c_q, as the minimal subtree of T that contains the $min(n, |T|)$ displays (nodes) preceding q.*

Note that c_q does not include q, as it represent the state of the user prior to its execution.

As an example, in Figure 2, the subtree in the dashed frame is the 4-context of action q_5. In Section 4 we examine the effect of different sizes of n-contexts, and show that using not too large values allows for both interactive performance and good recommendations quality.

2.2 Generalizing Analysis Actions

Since analysis actions may be performed in different contexts and on different datasets, we will be interested in *generalizing* a set of relevant actions into *abstract actions*, representing their commonalities.

An *abstraction* of a given action is obtained by generalizing (i.e., replacing) a subset of its parameters by generic variables, denoted by $*$. We define first a generalization for a single parameter, then lift it to action parameter sets.

[1] If the same display is generated twice (yet on different paths) it is represented by two different nodes

Definition 2.3 (Single Parameter Generalization)**.** *A single key-value parameter pair* $\langle k, v\rangle$ *can be generalized into* $\langle *, v\rangle$ *or to* $\langle k, *\rangle$*, where* $*$ *denotes the generic variable. These may be further generalized to* $\langle *, *\rangle$*, which forms a generalization for all possible parameters. Thus such "generalization" forms a partial order over template parameters:* $\langle *, *\rangle \leq \langle *, v\rangle$; $\langle *, *\rangle \leq \langle k, *\rangle$; $\langle *, v\rangle \leq \langle k, v\rangle$; $\langle k, *\rangle \leq \langle k, v\rangle$*. We refer to a generalized parameter as an* abstract parameter *and denote it by* $\langle \hat{k}, \hat{v}\rangle$*, where* $\hat{k}$ *and* $\hat{v}$ *stand for either a variable* $*$*, or some concrete key/value.*

Lifting these definitions to actions (i.e sets of parameters), we define an *abstract action* (or *abstraction* in short) as an action whose parameters may consist of abstract parameters. We require that the abstract actions in the set are incomparable, i.e. the set does not contain a parameter and its generalization. Now, a partial order of actions can then be defined by lifting the partial order of single parameters: Abstract action $\hat{q}_1$ "generalizes" $\hat{q}_2$ (or $\hat{q}_2$ is an "instantiation" of $\hat{q}_1$) denoted by $\hat{q}_1 \leq \hat{q}_2$, if all parameters in $\hat{q}_1$ are more general than parameters in $\hat{q}_2$. Formally:

Definition 2.4 (Abstract action, Partial Order)**.**
(1) An abstract action $\hat{q} = \{\langle \hat{k}_i, \hat{v}_i\rangle\}$ *is a set of incomparable abstract parameters, i.e.:* $\nexists \langle \hat{k}_1, \hat{v}_1\rangle, \langle \hat{k}_2, \hat{v}_2\rangle \in \hat{q}$*, s.t.* $\langle \hat{k}_1, \hat{v}_1\rangle \leq \langle \hat{k}_2, \hat{v}_2\rangle \wedge \langle \hat{k}_1, \hat{v}_1\rangle \neq \langle \hat{k}_2, \hat{v}_2\rangle$*.*
(2) $\hat{q}_1$ *generalizes* $\hat{q}_2$ *(denoted by* $\hat{q}_1 \leq \hat{q}_2$*) iff:* $\forall \langle \hat{k}_1, \hat{v}_1\rangle \in \hat{q}_1, \exists \langle \hat{k}_2, \hat{v}_2\rangle \in \hat{q}_2$ *s.t.* $\langle \hat{k}_1, \hat{v}_1\rangle \leq \langle \hat{k}_2, \hat{v}_2\rangle$

Consider the following example for an illustration of the actions' partial order.

Example 2.5. *Given the (abstract) actions:*
q_1=*{⟨ 'type',**FILTER**⟩, ⟨ 'attr','Protocol'⟩, ⟨ 'opr','='⟩, ⟨ 'term','SSL'⟩}*
$\hat{q}_1$=*{⟨ 'type',**FILTER**⟩, ⟨ 'attr','Protocol'⟩, ⟨ 'opr',∗⟩, ⟨ 'term','SSL'⟩}*
$\hat{q}_2$=*{⟨ 'type',**FILTER**⟩, ⟨ 'attr','Protocol'⟩, ⟨ 'opr','='⟩, ⟨ ∗,'SSL'⟩}*
First, observe that none of the abstractions contain a parameter and its generalization. Second, according to the partial order we have that $\hat{q}_1 \leq q_1$ *and* $\hat{q}_2 \leq q_1$*, but* $\hat{q}_1$ *and* $\hat{q}_2$ *are incomparable, i.e. neither one generalizes the other.*

We further lift the definition to a set of actions and say that an abstract action $\hat{q}$ *generalizes* a set of (abstract) actions $\hat{Q}$, iff $\forall \hat{q}' \in \hat{Q}, \hat{q} \leq \hat{q}'$. The set of all generalizations of $\hat{Q}$ is denoted $G(\hat{Q})$ Last, we say that a generalization $\hat{q} \in G(\hat{Q})$ is *minimal*, iff $\nexists \hat{q}' \in G(\hat{Q}), \hat{q} < \hat{q}'$.

For instance, the minimal generalization for $\hat{q}_1, \hat{q}_2$ (as in the example above), denoted $MG(\{\hat{q}_1, \hat{q}_2\})$, is {⟨ 'attr',*'Protocol'*⟩, ⟨ 'opr',∗⟩, ⟨ ∗,*'SSL'*⟩}

In Section 3.2, we present an effective algorithm that, given a multiset of actions, derives a set of "minimal, frequent generalizations", and uses them to generate next-action recommendations. We also show that for a given set of abstractions $\hat{Q}$, always exists exactly one minimal generalization $MG(\hat{Q})$.

3 Recommendation Framework

We next present the components and workflow of `REACT`. As explained in the introduction, at each point of a user's analysis process, we would like to generate relevant next-action recommendations. For that we first find a set of "candidate" actions, performed by other users in *similar* n-contexts (Definition 2.2). We then generalize and process the candidate actions, to form a set of of next-action recommendations, tailored to the current user. In what comes next, in Section 3.1 we first describe a distance metric for n-contexts, then an optimized solution to efficiently retrieve the topmost similar n-contexts. In Section 3.2 we state the desired properties of the recommended actions. We provide efficient methods for the *generalization* process of the candidate actions, and finally, discuss how can one further *materialize* them, if concrete action recommendations are desired.

3.1 Finding Actions with Similar $n-$contexts

Let $c_\star$ be the current user's n-context *before* she decides on a new action. Given a repository of previous analysis trees, our first goal is to efficiently search the repository for the top-most *similar* other n-contexts to $c_\star$. We first briefly discuss how to measure the similarity of n-contexts, then explain how to efficiently identify such contexts.

Determining Context Similarity There are several ways to determine the distance of labeled ordered trees (e.g. [10]). We use the common *tree edit distance* to estimate the distance of any given n-contexts c, c', denoted $\delta(c, c')$. Briefly, the distance is defined by the minimum-cost sequence of *edit operations* $ES = es_1, es_2, ...$ required to transform c into c'. The edit operations allowed are *delete/add* a node or an edge, and *alter* the label of a node or an edge. We use a cost function $\gamma(es_i)$ that gives add/delete operations the *unit cost*, whereas the cost of *alter* for node/edge labels is proportional to the similarity between the data displays and analysis actions that they represent, respectively, bounded by the cost of deletion plus addition. Finally, the distance is the commutative cost of the minimal ES, namely $\delta(c, c') = min\{\sum_{i=1}^{N} \gamma(es_i)\}$.

Next, we intuitively define the distance notions for actions and displays. For a full description, see Appendix A.

Display Distance. Recall that different analysts may operate on distinct datasets (as well as on distinct data subsets), yet we would like to benefit from the users' experience across datasets and sources. Given two displays, we first match, and align their sets of attributes to a global schema (e.g. matching attribute names such as 'ip_address' and 'ip_addr' to a global attribute 'IP'), using a schema matching and alignment tool (e.g. [1]).

Then, we compare the content of the matched attributes w.r.t. each layer (data, granularity, mining, etc). Intuitively, our distance metric captures the following (for explicit formulas refer to Appendix A): (1) Their schema similarity (2) The structure of the data: we compare structural properties of each matching columns such as its value entropy, # of unique values and # of nulls. (3) Grouping/cube differences, including the difference in the level of abstraction in both displays (w.r.t. the attributes hierarchy), as well as differences in the resulted groupings e.g. the number of groups and their size variance. (4) Differences in the mining label sets (by employing a simple variation of the generalized Jaccard distance for multisets).

The final distance score between displays is the sum of the above mentioned factors, normalized to obtain values in $[0, 1]$.

Action Distance. We measure the distance between two analysis actions q, q' by the sum of their "distances" from their minimal generalization $MG(\{q, q'\})$, as defined in Section 2.2, normalized by their distances to the most general abstraction $\{\langle *, * \rangle\}$. As in the display distance, since actions may operate on different datasets, we first align attributes to the global schema. Then, before measuring the distance, we replace attribute names (i.e. in the parameter $\langle$ 'attr',$a\rangle$), by their corresponding global schema ones.

Following Example 2.5, the distance between $\hat{q}_1$ and $\hat{q}_2$ is 0.2, as their distances from $MG(\{\hat{q}_1, \hat{q}_2\})$ and $\{\langle *, * \rangle\}$ are 1 and 5, resp.

Last, to obtain distance values between $[0, 1]$ (while preserving the metric properties) we employ the Steinhaus transform [16] on the context distance scores.

Our efficient similarity search described next is based on the following observation:

Observation 3.1. *If the actions/displays distance notions are proper metrics, then δ is a proper distance metric. I.e. non-negative, symmetric and subadditive s.t. $\delta(T, T') = 0$ only if $T = T'$.*

We show in Appendix [**?**] that our action and display distance functions are proper *metrics.*

Context Indexing, Similarity search Analysis platforms and databases often store a *query log*, usually recording the syntax of past executed queries. Correspondingly, in our system we maintain a repository of *analysis trees*, denoted $\mathcal{T}$ that includes for each tree $T \in \mathcal{T}$ the actions and displays it comprises. Since storing the entire results display is costly, we only store in our repository a compact summary of each displays, containing the statistical meta-data required for the displays distance comparison as previously mentioned.

Now, our goal is to efficiently find the top-k most similar n-contexts, to the current user's context $c_\star$. A straight-forward solution would be to employ a subtree similarity search (e.g. [9]), to find all subtrees in $\mathcal{T}$ similar to the current $c_\star$ (then filter the output set for valid n-contexts according to definition 2.2).

However, this solution requires the traversal of the entire repository[2] , which may take too long for an interactive recommender system. In `REACT`, we extract the n-contexts from the analysis tree repository and store them in a *metric tree* [14] index. We briefly describe the properties of the metric-tree, then explain how we use it in our setting.

A metric-tree is a data structure designed for indexing objects residing in a metric space, for performing an efficient search by exploiting the triangle inequality property of the metric. In a metric tree, all objects are stored in its leaves, and the non-leaf nodes comprise of (1) a representative object, denoted by s, chosen among its descendant leaf nodes, (2) a "covering" radius $r(s)$, denoting the maximal distance between the representative object s and all of its descendants, and (3) for each child node of s, denoted s^i it stores the distance between their representative objects $\delta(s, s^i)$, as well as its covering radius $r(s^i)$.

In our settings, whenever a user executes a new action q, as a part of analysis tree T', we construct an object c_q that represent the current n-context and

[2] While some index structures exist for unit-cost tree edit operations[15], this is not the case in our setting.

consists of the pointer to the first and last nodes of the context subtree in T'. Then we add c_q to the metric tree w.r.t. the context distance metric δ. We denote the resulted metric tree by $\mathcal{M}_c$.

Although the metric tree facilitates an efficient *k-nearest neighbors (kNN) search*, that given a query object retrieves the top-k most similar objects, our problem settings required a different strategy. Due to the inherent sparsity of the n-contexts, it may be that some of the top-k similar sessions may be too different from the current one, thus inadequate for generating recommendations. We therefore suggest using a refinement, denoted *Bounded Top-k*, which restricts the kNN search s.t. all retrieved objects are also within a given radius.

Definition 3.2 (Bounded Top-k Search)**.** *Given the metric tree $\mathcal{M}_c$, a fixed distance bound θ, a number k, and a current $c_\star$, Bounded Top-k Search retrieve a set of k n-contexts denoted C_k that are (1) the k most similar to $c_\star$, and (2) their distance from $c_\star$ is at most θ.*

Our bounded top-k search is implemented as follows: Given $\mathcal{M}_c, \theta, k, c_\star$ we first initialize the temporary output set C_k with k "empty" n-contexts, s.t. their distance from $c_\star$ is ∞.

Then, the metric tree $\mathcal{M}_c$ is traversed from the root node. At any non-leaf node v_s, we compute $\delta(c_\star, s)$, where s is the representative object (i.e. n-context) of v_s. Let θ_k be $min(\theta, \delta_k)$, where δ_k denotes the distance between $c_\star$ and the *least* similar n-context in C_k. Then, by employing the triangle inequality, further traversal paths are excluded in cases where (1) the distance $\delta(c_\star, s) > r(s) + \theta_k$ (where $r(s)$ is the covering radius of s) or (2) any representative object s^i of a child node of v_s, s.t. $|\delta(c_\star, s) - \delta(s, s^i)| > \theta_k + r(s^i)$. When a leaf node is reached, we iteratively insert its associated n-contexts to C_k, if their distance from $c_\star$ is less than δ_k, and correspondingly remove elements from C_k that are more distant from $c_\star$

In Section 4 we examine the performance of the bounded top-k strategy, comparing to the kNN and to a naive "brute-force" search. Results show that the bounded top-k search is indeed more suitable in our setting, and allows faster execution times (by an average of more than 20%). The latter is due to the excessive computations performed in the kNN search in order to maintain a global, unbounded top-k set (i.e. as long as δ_k, the distance score of the least similar element in C_k, is lower than the bound θ).

3.2 Mining Abstract Actions

To generate recommendations, we consider the "next-action" of each of the similar n-contexts in C_k (obtained as described above). We call these *candidate actions*, denoted by $Q_k = \{q | c_q \in C_k\}$.

The question we address here is how, and to what extent can Q_k be used to generate next-actions recommendations, given that the actions may be executed on different datasets? One possibility, is to recommend actions that appear frequently in Q_k. However, since users in our setting may be examining different datasets, there may be no such frequent actions. For example, assume that the actions in Q_k are:

(1) `FILTER by 'ip_address'='192.168.1.1'`,
(2) `FILTER by 'ip_addr'='10.0.0.2'`,
(3) `FILTER by 'ip'='164.148.2.26'`

While the actions are intuitively similar, none appear more than once. To that end, we suggest finding the *commonalities* of the actions in Q_k, using the actions generalization constructs, defined in Section 2.2. As actions may be employed on attributes of different schema, we first replace the attributes in each action to its equivalent in the global schema (similarly as in the actions distance notion described in Section 3.1).

Continuing with the example, assuming that the schema alignment maps the attributes in (1), (2), and (3) to 'IP', we can derive that the *abstract* action "*Employ a Filter on the 'IP' column by SOME value*" is a meaningful next-step suggestion to the current user.

Of course, not all such generalized, abstract actions may be useful (e.g. the most general abstract action $\{\langle *, * \rangle\}$, while frequent in Q_k, is clearly uninformative). We next explain how our system allows to restrict attention only to meaningful abstract actions, and efficiently extracts them from Q_k.

Recommendation candidates Given the action multiset Q_k, our goal here is to extract, as *recommendation candidates*, a set of abstract actions that generalize actions in Q_k. Intuitively, we are interested in abstract actions that are, first of all, "frequent", i.e., correspond to action fragments that are frequent in Q_k. Also, to extract the most information from Q_k, we require the frequent abstractions to also be *minimal* (as in Section 2.2), i.e. as specific as possible. Last, since we are interested in providing only meaningful, *coherent* recommendations, we will allow a set of predefined constraints for recommendation candidates, e.g. *"Must contain an attribute name"*.

Formally, let $\hat{Q}_k$ be the set of all abstractions corresponding to Q_k, namely $\hat{Q}_k = \{\hat{q} | \exists q \in Q_k, \hat{q} \leq q\}$. Out of the large set $\hat{Q}_k$, we are interested in finding a (small) subset $\hat{R} \subset \hat{Q}_k$ of *recommendation candidates* s.t. each $\hat{r} \in \hat{R}$ has the following properties:

1. Frequent in Q_k. Given a threshold θ_f, an abstract action $\hat{q}$ is *frequent* iff $\frac{|\{q | q \in Q_k, \hat{q} \leq q\}|}{|Q_k|} \geq \theta_f$. We require that each $\hat{r} \in \hat{R}$ is frequent.

2. A Minimal Generalization. Minimal generalizations preserve the most "information" about the essence of their corresponding actions. We thus require that each $\hat{r} \in \hat{R}$, is both frequent and minimal, i.e. $\nexists \hat{q} \in \hat{Q}'_k$ s.t. $\hat{r} < \hat{q}$ and $\hat{q}$ is *frequent* in Q_k.

3. Coherent. We allow a predefined set of coherency constraints, namely a set of abstractions $\hat{T}$, requiring any $\hat{r} \in \hat{R}$ is an instantiation of at least one of them, i.e. $\exists \hat{t} \in \hat{T}$ s.t. $\hat{t} < \hat{r}$. For example, we can restrict our process to return recommendations that only contain attributes that are present in the currently examined dataset, i.e. $\hat{T} = \{\langle$ 'attr',$a_1\rangle$, $\langle$ 'attr',$a_2\rangle$,...$\}$ where a_i is an attribute name (in the global schema).

Extracting recommendation candidates Given the action multiset Q_k, a frequency threshold θ and a set of coherency constraints $\hat{T}$, we will now describe how to extract $\hat{R}$, the corresponding set of recommendation candidates.

A naive solution would be to generate $\hat{Q}_k$, the set of all possible abstractions, then iterate through all $\hat{q} \in \hat{Q}$, and output $\hat{q}$ only if it is a valid recommendation candidate, as defined above (i.e., frequent, minimal and coherent). However, this is computationally expensive since the size of $\hat{Q}_k$ may be exponential in $|Q_k|$, and determining the frequency of a template demands a traversal over the action multiset Q_k.

The efficient mining of abstract actions is enabled due to a reduction from our problem to the *frequent itemsets mining* (FIM) problem under *monotonic constraints* [23]. We first provide necessary propositions, then explain the reduction and describe how any state-of-the-art FIM algorithm may be used to extract $\hat{R}$ from Q_k, without materializing the large set $\hat{Q}$.

First, we show the following essential properties:

Proposition 3.3. *(1) The multiset $\hat{Q}_k$ forms a semi-lattice, under the generalization partial order. (2) Any minimal generalization $MG(\hat{Q})$ is a least-upper-bound for $\hat{Q}$: For any two actions $\hat{q}_1, \hat{q}_2$ exists a single minimal generalization $MG(\{\hat{q}_1, \hat{q}_2\})$.*

(sketch). We need to show that for any two (abstract) actions $\hat{q}_1, \hat{q}_2$ exists a single MG (i.e. least upper bound). Existence of an MG can be proven by construction. Given action $\hat{q}_1, \hat{q}_2$, we first expand each of their parameters sets to include all "ancestors", i.e. given action $\hat{q}_i$, create $\hat{q}_i^* = \{\langle\hat{k}, \hat{v}\rangle | \exists\langle\hat{k}', \hat{v}'\rangle \in \hat{q}_i, \langle\hat{k}, \hat{v}\rangle \leq \langle\hat{k}', \hat{v}'\rangle\}$. Thus $MG(\hat{q}_1, \hat{q}_2)$ is obtained by $\hat{q}_1^* \cap \hat{q}_2^*$, then pruning redundant ancestors, that are more general than others (See Definition 2.3). As for uniqueness, we can prove that any $MG(\{\hat{q}_1, \hat{q}_2\})$ is derived from the intersection $\hat{q}_1^* \cap \hat{q}_2^*$, which is naturally unique. □

Now, we explain how to use an FIM algorithm in our settings. First, recall from the literature that an FIM algorithm is given a catalog of items, and a set of transactions (each comprises a multiset of items), then it mines all *maximal-frequent itemsets* (MFI): i.e. sets of items that appear frequently in the transactions, yet none of their supersets are frequent. We denote by $FIM(X, \theta_f)$ the output MFIs of a given FIM algorithm on a multiset X of transaction with frequency above θ_f.

Applying it to our settings, let Q_k^* be the extension of Q_k, obtained by embedding in each action $q \in Q_k$ the ancestors in the single-parameter partial order. Namely, $q^* := q \cup \{(\hat{k}, \hat{v}) | \; \exists(k, v) \in q, (\hat{k}, \hat{v}) \leq (k, v)\}$. Intuitively, each action $q^* \in Q_k^*$ is equivalent to a *transaction*, and its parameters (including ancestors) to *items*. We can prove the following proposition, stating that our desired set of abstractions are, in fact, MFIs:

Proposition 3.4. $\hat{R} \subseteq FIM(Q_k^*, \theta_f)$

We now state the procedure for computing $\hat{R}$.

Algorithm: (sketch) Given the actions Q_k, a constraints set $\hat{T}$ and a frequency threshold θ_f, we generate $\hat{R}$ as follows: (1) using Q_k we construct Q_k^* by embedding the parameters' ancestors in each $q \in Q_k$. (2) We apply an FIM algorithm to obtain the MFI set $FIM(Q_k^*, \theta_f)$. (3) We prune any $\hat{q} \in FIM(Q_k^*, \theta_f)$ if: (1) $\hat{q}$ is not a legal abstract action, that contains a parameter and its ancestor, i.e. $\exists\langle\hat{k}, \hat{v}\rangle, \langle\hat{k'}, \hat{v'}\rangle$ s.t. $\langle\hat{k'}, \hat{v'}\rangle \leq \langle\hat{k}, \hat{v}\rangle$. (2) $\hat{q}$ does not meet any of the constraints in $\hat{T}$, namely $\nexists\hat{t} \in \hat{T}$ s.t. $\hat{t} \leq \hat{q}$. The output of the procedure is the pruned set $FIM(Q_k^*, \theta_f)$, which is essentially the set of recommendation candidates $\hat{R}$.

Last, note that our input set for the FIM algorithm induces relatively short running times, comparing to the typical itemset mining scenario, due to the following:

(1) The number of "transactions" is relatively small, as it contains only k actions. In Section 4 we show that optimal recommendation quality is obtained when setting k between 15 to 30. (2) The size of each transaction, i.e. an extended action q^*, is at most $3|q|$, as to each parameter $\langle k, v\rangle \in q$ we add up to two generalizations - $\langle$ k,$*\rangle$ and $\langle$ $*$,$v\rangle$. (3) The coherency constraints in our settings are *monotonic*, i.e. if apply to a given action $\hat{q}$ then they also apply for any of its instantiations $\hat{q}'$ s.t. $\hat{q} \leq \hat{q}'$. The latter facilitates a further speed-up in the FIM process as suggested e.g. in [23].

We return the recommendations in $\hat{R}$, sorted by their frequency in Q_k, while replacing the global attributes with their local equivalents in the current user's dataset.

Materializing Abstractions to Concrete Recommendations The generalization procedure produces a set of frequent and minimal abstract actions, that are returned to the user as next step "suggestions". Some of the suggested abstract actions in $\hat{R}$ may not yet be executable actions (e.g. $\{\langle$ 'type',***AGGREGATE***$\rangle$, $\langle$ 'attr',*'Length'*$\rangle$ $\langle$ 'func',$*\rangle\}$, namely, "Employ *some* aggregation function on the column 'Length'", without specifying the aggregation function). As we demonstrate in our experimental evaluation (Section 4), these recommendations are most useful, and assisted real users to reduce analysis times by an average of 30%. Nevertheless, if desired, such abstract suggestions can be further *materialized* to obtain executable actions by assigning real values to the $*$ parameters. To do so, we harness as a complementary module a set of *data-driven tools* each designated for a specific action type (See [19] for drill-down operations, and [29] for data visualizations, [17] for FILTER, etc.). These tools consider the data subset at hand, and automatically select action parameters that maximizes the *interestingness* of its corresponding results set.

4 Experimental Results

We performed an extensive experiments set over real-world analysis logs that we acquired. We performed first an *offline evaluation*, as common in recommender systems, by measuring the ability of generated recommendations to *predict* the users' analysis actions. This allowed us to tune the system parameters, then compare its predictive performance with several baseline approaches. Second, we performed a "live" experiment with real users, testing if REACT can be practically used to reduce analysis times. We further evaluated the scalability of REACT using large, synthetic analysis workloads, and finally, we examined the effect of the system's parameters on the predictive performance as well as on execution times.

4.1 Experimental Setup

Implementation The prototype of REACT is implemented in Python 3.4+MySQL, using the metric tree described in [14], tree edit distance algorithm of [31], and schema matching and alignment tool from [1]. All experiments described below

Parameter	Value range	Default Conf.
n-context Size	$[3, 14]$	8
Dist. Threshold θ_δ	$[0.1, 0.8]$	0.45
Freq. Threshold θ_f	$[0.1, 0.3]$	0.1
k	$[5, 40]$	25

Table 1: Parameters Grid Search

were conducted on a MacBook Pro machine with 8 cores and 16GB RAM, out of which 6 were utilized for our system. In both the offline and online evaluations described in the sequel, we used REACT with a single constraint $\{\langle$ 'type',$*\rangle\}$, that restricts all recommendations to contain an action type, and set a limit of 3 on the number of recommendations presented to the user at each point.

Real-world dataset To our knowledge, there are no publicly available repositories of analysis actions performed on modern IDA platforms. We therefore recruited 56 analysts, specializing in the domain of cyber-security (via dedicated forums, network security firms, and volunteer senior students from the Israeli National Cyber-Security Program), and asked them to analyze 4 different datasets using a prototype web-based analysis platform that we developed (as in Figure 1). Each dataset, provided by the Honeynet Project [28], contains between 350 to $13K$ rows of raw network logs that may reveal a distinct security event, e.g. malware communication hidden in network traffic, hacking activity inside a local network, an IP range/port scan, etc. (there is no connection between the tuples of different datasets). The analysts were asked to perform as many analysis actions as required to reveal the details of the underlying security event of each dataset. All actions (total of 1152), displays, and corresponding n-contexts were recorded as mentioned in Section 3.1, and the attributes of the 4 datasets aligned to a global schema comprising 15 columns, obtained from [3]. Our acquired action log is publicly available [2] for use in future work.

4.2 Offline Evaluation

Predictive accuracy is a common method for measuring the utility of a recommender system. We describe first the evaluation process and metrics, then overview the results.

Evaluation technique We simulated the recorded analysis sessions one by one, and at each point in a session, used REACT to generate recommendations (with the given session omitted from the repository), then examined whether the recommendations indeed correspond to the actual next-action performed by the user, at this point. We used the following standard evaluation metrics in Information Retrieval: First, we evaluated "high-level" accuracy, by considering a recommendation *relevant* if it has the same 'type' and 'attr' values as the true action q. We then used **(1) R@3**, i.e. Recall at 3, which counts the number of *relevant* recommendations, i.e. yields 1 if one of the three recommendations was relevant and 0 otherwise. To further account for the order of recommendations, we used **(2) MRR** (Mean Reciprocal Rank) score, which discounts the score according to the position of the relevant recommendation. Second, we evaluated precision and recall w.r.t. the complete set of parameters in each action, by using the **(3) Macro F1-score** which is the harmonic mean of the precision

Baseline	Multi-dataset			Cross-Dataset		
	MRR	**R@3**	**F1**	**MRR**	**R@3**	**F1**
RANDOM	0.03	0.05	0.3	0.03	0.05	0.3
NO-GEN.	0.55	0.65	0.78	0.52	0.62	0.77
REACT:DO	0.67	0.75	0.85	0.69	0.77	0.85
REACT:AO	0.73	0.8	0.89	0.72	0.79	0.89
REACT	**0.76**	**0.84**	**0.9**	**0.75**	**0.81**	**0.91**

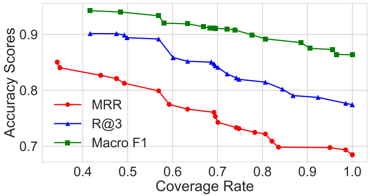


Figure 3: Skylines of Coverage/Accuracy

and recall. We aggregated the results for all prediction cases, when the system was able to produce at least 1 recommendation, and correspondingly measured the **(4) coverage** rate, i.e. the proportion of cases where REACT produced recommendations out of the total number of cases.

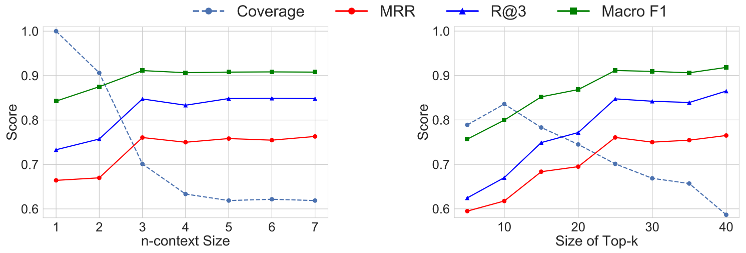
Evaluation scenarios. Recall that we captured analysis activity in 4 different (yet related) datasets. We thus evaluated our solution in two scenarios: (1) **Multi-dataset** scenario, where REACT utilizes analysis actions performed on *all* datasets, and (2) **Cross-dataset** scenario, in which we examined if our solution can provide recommendations for a user analyzing a given dataset, using only sessions performed on the other datasets than the one she examines.

Parameters Selection We used a standard *grid search* consisting of more than $11.5K$ unique settings. Table 1 depicts the system parameters and their range. To choose an optimal configuration we calculated the *skyline* to obtain a set of dominant configurations w.r.t. the coverage, and each of the accuracy scores (R@3, MRR and F1). Figure 3 depicts the 2-dimensional skyline for the coverage and each of the accuracy measures, in the multi-dataset scenario (similar trends were examined for the cross-dataset experiment, thus omitted).

Naturally, there is a tradeoff between optimal coverage and predictive accuracy (See Section 4.5 for a discussion). For the experiments described next, we selected a default configuration (See Table 1) that obtains an MRR, R@3 and F1 scores of $0.76, 0.84, 0.9$ (resp.) while retaining a coverage of 70%.

Baselines Comparison We next compare the predictive performance of our system to several baselines.

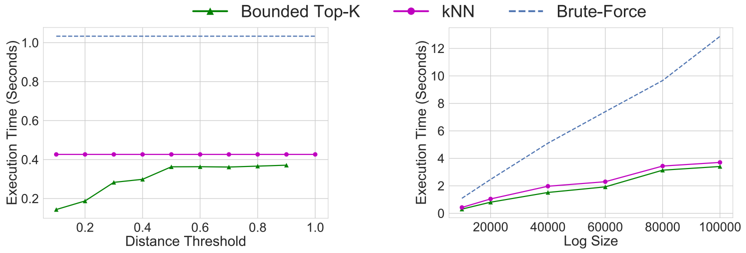
Since, to our knowledge, previous work does not support the multi-facet IDA setting that we consider, (see Section 5 for discussion), we do not benchmark against them. The following baselines were used: **(1) NO-GEN.** This baseline directly applies collaborative techniques, and recommends previous queries from the log, without our generalization mechanism: For each case, it generates recommendations by finding the top-3 similar n-contexts (using our context distance metric), then returns their corresponding actions. We use this baseline to examine the need in our generalization process. **(2) REACT:AO** (actions only)



(a) PA: Context Size

(b) PA: Size of Top-k

Figure 4: Predictive Accuracy



(a) ET: Dist. Threshold

(b) ET: Log Size

Figure 5: Execution Times

and **(3)** **REACT:DO** (displays only) are variants of `REACT`, that use a restricted version of the similarity metric which only considers the actions syntax/displays summary (resp.) It is used to test whether considering both actions and displays improves the predictive accuracy. **(3) Random.** This simple baseline generates 3 random actions and returns them as recommendations. This one is used to test whether one can simply "guess" the next action. We compared the baselines in both the multi-datasets and the cross-datasets scenarios. For baselines (2) and (3), we used the same parameters selection routine (as described earlier in this section) to find their best configuration that yields above 70% coverage.

Table 2 depicts the Accuracy scores of all baselines, in the multi-dataset and cross-dataset scenarios. First, we observe that the random baseline fails to produce adequate recommendations, as the space of possible actions is rather large. Second, we see that `REACT` outperforms both the *actions only* and *displays only* variant which ignores the results-set/action syntax similarity. Third, we observe that the *collaborative* baseline is substantially inferior to `REACT`, demonstrating the importance of our generalization process in the IDA diverse environment. Importantly, observe that `REACT` obtained the best performance also in the *cross-dataset* scenario, i.e. when it has to provide recommendations only based on actions performed on *different* datasets.

4.3 Online Evaluation

We next demonstrate the effectiveness of `REACT` for assisting users, in practice. Since modern IDA platforms provide a simplified interface that does not require prior SQL/programming knowledge, our goal here is to assist inexperienced analysts who are not domain experts, (as opposed to assisting domain experts lacking SQL knowledge, as examined in [18, 20]).

To that end, we recruited 20 computer science graduate students, all familiar with data analysis practices, but who are not experts in cyber security. The participants were asked to perform two distinct analysis tasks on two (different) datasets from the Honeynet challenges collection, such that in one task they are unassisted, and in the other they are supported by REACT. In each task, the goal was to identify the underlying security event hidden in the particular dataset. For each participant, we measured the number of actions, and the amount of time required to successfully analyze the dataset. To neutralize external effects (e.g. which of the challenges was solved with/without REACT, and whether REACT was used first or second), we considered all four combinations, splitting the users into four equal size groups. Figure 6 depicts the average number of actions it took users to complete the tasks, as well as the variance bar, w.r.t. the sessions order (with REACT or unassisted), and the examined datasets.

We can see that in all cases, with REACT, the number of analysis actions required to complete the task was reduced by approximately 50%. Correspondingly, the overall analysis time was also reduced by an average of 30%, regardless of the dataset and the order of tasks.

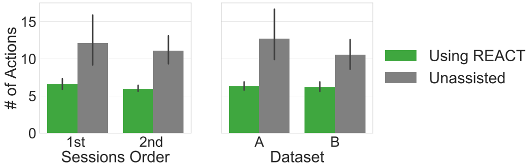


Figure 6: Online Evaluation: With/Without REACT

4.4 Running Time & Scalability

Our next experiment examines the system's response time as a function of the log size. In order to examine performance over a larger analysis log than our real-world dataset, we generated synthetic workloads of analysis sessions, following [24], by first producing a set of random "seed" contexts and then constructing a set of variants for each seed. This was done by randomly generating edit scripts for the given tree up to a distance bound of 0.3.

Figure 5b displays the overall execution times of REACT (in green) for log size of $10K$ to $100K$. To further examine the efficiency of our contexts index mechanism and similarity search strategy, we provide the execution time of two variants of REACT, using alternative similarity search techniques: (1) the traditional metric tree kNN search, and (2) *Brute-force* similarity search, which retrieves the top-k contexts by performing a sequential pass on all n-contexts.

First, see that REACT - using the Bounded Top-k similarity search - outperforms the unbounded kNN by an average of 17%, and the brute-force by 71%. For a moderately large log of $10K$ actions, REACT produces recommendations in less than 0.5 seconds, while for a significantly larger one ($100K$) it takes 3.4 seconds, which, given the time the users need to examine the results display before they consider the next action to perform, is a reasonable time (We measured a median time of 40 between consecutive queries). Also note that the generalization process took no longer than 8 milliseconds, in all tested configuration. Last, recall that our prototype is implemented in Python, and utilizes only 6 cores. Hence execution times could be greatly improved if added more processors or if using a compiled/JIT language such as *C* or *Java*.

4.5 System Parameters Effect

Last, we provide a deeper dive into the effect of the system parameters on the predictive accuracy, and on execution times. In each experiment, we varied one parameter while keeping the others in their default values (as in Table 1). We provide here only a brief overview of our findings, and refer the reader to Appendix [**?**] for further details.

Briefly, we observe the following trends: (1) The accuracy/coverage tradeoff: n, k, and θ (the context size, top-k size, and distance threshold, resp.) control the amount of "information" considered in the similarity comparison and in the generalization process. For instance, choosing larger values of n and k dictates `REACT` to retrieve a *larger* set of *longer* similar n-contexts. This naturally increases accuracy, yet decreases the coverage, as there are fewer cases where a large set of similar n-contexts can be found (See Figures 4a and 4b). (2) The distance threshold θ has a similar effect on accuracy/coverage (namely, tighter threshold induces higher accuracy, and respectively, lower coverage), however as depicted in Figure 5a, it reduces execution times. This is due to the Bounded Top-k strategy (described in Section 3.1) which employs the tight threshold for pruning more dissimilar n-contexts.

5 Related Work

The design of recommender systems that utilize the query log to assist users in data analysis has been the focus of a significant body of work. However, although the analysis paradigm shifts towards modern web-based analysis platforms, prior work mostly focuses on recommending explicit SQL/OLAP queries and generally assumes that users are investigating the same database/cube.

For SQL, works like [20, 18] suggest query recommendation, with the primary goal of assisting users lacking SQL knowledge to formulate queries w.r.t. their information needs. Considering the current user's past query sequence, they find other users with similar sequences (that contain similar query "fragments"), then return as recommendation the most suitable query (or query parts) from the log. We argue, and empirically show in our experiments (Section 4.2), that due to the diverse datasets environment as in nowadays IDA settings, presenting users with past queries, exactly as appear in the log, is not optimal. We address this by our *actions generalization* mechanism.

Similar techniques used for recommending OLAP MDX queries (See [22] for a survey). Closer to our work is [6], providing OLAP query sequence recommendations. The authors note that it is unlikely that two OLAP sessions share identical queries, therefore suggest to first find the top most similar session in the log (using a dedicated measure for OLAP sessions in [7]), and then adapt it to the current user, by matching query fragments in both sessions. However, they assume that there always exists a *single* session with high enough similarity to the current one, rather than synthesizing a recommendation based on an analysis of *multiple* similar sessions. Here too our experiments (Section 4.5) show that in the IDA environment, best results were obtained when deriving recommendations from *multiple* contexts (top-25 yielded the best results in terms of accuracy/coverage).

A complementary module that we use in REACT (as described in Section 3.2) is *data-driven* recommendations (also called *discovery-driven* in the literature) that we employ to instantiate action parameters. These tools use heuristic notions of *interestingness* and employ them, e.g., to find data subsets conveying interesting patterns ([25, 17]), choose high-utility drill-down parameters [19], data visualizations [29], and data summaries [27].

In our work, we define the analysis context as the recent actions performed thus far, and their results displays. Alternative notions of context (e.g. in [12]) consider a user's a-priori *profile*, comprises e.g. her role, analysis goals, etc. Such information can be incorporated in our system by refining the n-contexts similarity measure to also take the similarity of these parameters into consideration.

Also, our framework draws similar lines to previous work in *process mining* [30, 26], presenting recommendations based on high-level workflows extracted from application event logs. However, such works assume limited number of possible "activities" at a given state, which is not the case for the flexible process of data analysis.

Last, recent techniques from *transfer learning* and *embedding-based representation learning* [11] can be used to investigate how one may find a mapping between different analysis scenarios, and transfer knowledge between different analysis tasks and datasets. Harnessing such techniques to our context would make an exciting future research.

6 Conclusion

This work presents REACT, a recommender system designated to assist users of modern, web-based IDA platforms. REACT's generic data model supports a range of high-level action types, and can be easily extended to include new ones. Our n-context similarity metric takes into account both the actions' syntax and their corresponding multi-layered displays. We synthesize next-action suggestions by generalizing multiple actions executed in similar n-contexts, extracting abstract actions that capture meaningful commonalities. Our experiments with real-life data and users demonstrate the effectiveness of our solution. An additional contribution of our work, is the real-life IDA logs acquired in our experiments that may serve as a benchmark dataset for future work.

In future work, we intend on investigating tighter integration with *data driven* tools that may lead to a better comprehension of the dataset at hand. Similarly, process mining techniques may be used to encompass more elements in the analysis context (e.g. mouse movements, time spent on examining result sets, etc.), and embedding techniques for finding a comprehensive representation of analysis contexts, displays, and actions.

References

[1] Biggorilla: Data integration in python. `https://www.biggorilla.org/`.

[2] REACT: Ida benchmark dataset. `https://github.com/TAU-DB/REACT-IDA`.

[3] Wireshark: Network protocl analyzer wiki. `https://wiki.wireshark.org/`.

[4] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDBJ*, 2015.

[5] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems. *TKDE*, 2005.

[6] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi. A collaborative filtering approach for recommending olap sessions. *ICDSST*, 2015.

[7] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for olap sessions. *KAIS*, 39, 2014.

[8] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *INFOVIS*, 2005.

[9] N. Augsten, D. Barbosa, M. Böhlen, and T. Palpanas. Tasm: Top-k approximate subtree matching. In *ICDE*, 2010.

[10] N. Augsten, M. Böhlen, and J. Gamper. The pq-gram distance between ordered labeled trees. *TODS*, 2010.

[11] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 2013.

[12] C. Bolchini, E. Quintarelli, and L. Tanca. Context support for designing analytical queries. In *Methodologies and Technologies for Networked Enterprises*, pages 277–289. Springer, 2012.

[13] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, 2009.

[14] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[15] S. Cohen. Indexing for subtree similarity-search using edit distance. In *SIGMOD*, 2013.

[16] M. M. Deza and E. Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, pages 1–583. Springer, 2009.

[17] M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *The VLDB Journal*, 22(6), 2013.

[18] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 2014.

[19] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, 2016.

[20] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for sql. *VLDB*, 2010.

[21] J. Liu, A. Wilson, and D. Gunning. Workflow-based human-in-the-loop data analytics. In *HCBDR*, 2014.

[22] P. Marcel and E. Negre. A survey of query recommendation techniques for data warehouse exploration. In *EDA*, pages 119–134, 2011.

[23] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *KDD*, 2000.

[24] S. Rizzi and E. Gallinucci. Cubeload: a parametric generator of realistic olap workloads. In *CAISE*, 2014.

[25] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.

[26] H. Schonenberg, B. Weber, B. Van Dongen, and W. Van der Aalst. Supporting flexible processes through recommendations based on history. In *ICBPM*, 2008.

[27] M. Singh, M. J. Cafarella, and H. Jagadish. Dbexplorer: Exploratory search in databases. *EDBT*, 2016.

[28] L. Spitzner. The honeynet project: Trapping the hackers. *IEEE S&P*, 2003.

[29] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *VLDB*, 2015.

[30] S. Yang, X. Dong, L. Sun, Y. Zhou, R. A. Farneth, H. Xiong, R. S. Burd, and I. Marsic. A data-driven process recommender framework. In *KDD*, 2017.

[31] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM*, 1989.

A REACT: Prototype Implementation

In this Section we describe our prototype implementation of REACT, designed to work with IDA interfaces supporting the following actions: (1) Basic relational actions such as FILTER PROJECT, SORT actions, as well as GROUP-BY and AGGREGATE, (2) Data cube actions e.g. ROLL-UP and DRILL-DOWN, and (3) basic ML based actions like CLUSTER DETECT-OULIERS. We first explain what information is captured for these interface, then describe the context distance function, and finally, we present the end-to end system workflow.

A.1 Prototype IDA Platform

In principle, the recommendation engine is an intermediate plug-in for any web based IDA platform. As a proof of concept, we have developed our own web-based analysis IDA (Python 3.4, AngularJS). REACT-UIcan be loaded with any tabular dataset (and a corresponding OLAP hierarchy) provided by the user, and support the following actions: Basic data retrieval actions - FILTER, PROJECT, SORT, GROUP-BY; Data cube (OLAP) actions that alter the point-of-view of the data tuples - ROLL-UP, DRILL-DOWN; and the mining operations CLUSTER, and elemDETECT-OUTLIERS a basic data mining action.

In accordance, a display in REACT-UIcomprises three layers, denoted by $d = \{L_D, L_G, L_M\}$:

Raw data layer. It depicts the currently projected tuples and attribtues from the original dataset. Instead of maintaining the entire results set, inspired by data profiling tools, (see [4]) we keep a compact summary containing structural meta data. Formally, for a set of currently projected attributes A, the raw data layer $L_D = \{\langle I_U^a, I_N^a, I_H^a \rangle | \forall a \in A\}$ consists a triplet of indices for each column s.t. (1) I_U indicates The number of unique values, (2) I_N marks the number of NULL values, and (3) I_H stands for the normalized value entropy, given for a *categorical* column a containing the unique values $x_{1,2}, ..x_n$ as $H = -\sum\limits_{x \in a} \frac{p_x \log p_x}{\log |a|}$, where p_x denotes the *frequency* of the unique value $x \in X$.

For *numeric* columns, since they may be sparse, we first perform *equi-width* binning, dividing the value range to B different buckets ranging from the minimal value to the maximal. The normalized entropy $H = -\sum\limits_{b \in B} \frac{p_b \log p_b}{\log |a|}$ where p_b denotes the frequency of column's values associated to the bucket b.

Granularity layer. This layer describes the level of abstraction for columns that are *grouped-by*, or *aggregated* using some aggregation function (e.g., SUM, COUNT, etc.). Given a set G of grouped/aggregated attributes, We denote the granularity layer by $L_G = \langle \{I_G^a | \forall a \in G\}, I_C, I_V \rangle$, where for an aggregated column (1) $I_G^a = l(a)$, i.e. the position of the current level in the OLAP hierarchy (when such hierarchy is absent, $l(a) = 1$). If the column a is aggregated, then, $I_G^a = I_H^a$, where I_H stand for the normalized value entropy (as defined for numeric columns above). (2) I_C^a denotes the number of different groups and (3) I_V^a marks their size-variance, i.e. $I_V = \sum_{i=1}^{I_C} (p_i - \mu)^2$ where p_i stands for the probability for group i (i.e its size divided by the number of tuples), and μ is the average group size in the current display.

Mining layer. describing currently presented outcomes of data mining actions, by a multiset of labels representing particular data mining results (e.g. clustering, outliers, association rules). Formally, we denote the mining layer by $M = \{(o_i, L_i)\}$, associates each object $o_i \in O$ a (possibly empty) set of labels $L_i \subseteq \mathcal{L}$, where $\mathcal{L}$ denotes the domain of data mining labels.

A.2 Distance metric for analysis contexts

Recall that an n-contexts, as a tree-based model comprises of the actions (edges), their corresponding displays (nodes), and the particular order of execution (preorder traversal). To account for all available contextual information, recall that we base our notion of context distance, denoted δ, on *tree edit distance*, using two ground metrics for actions and displays, denoted δ_q, δ_d (resp) each in range $[0, 1]$.

We first detail the groun metrics for measuring the distance between actions, and between displays then give the explicit context distance definition.

A.2.1 Action Distance

Our distance notions for analysis actions, compares two actions (namely, KVP parameter sets) by using the actions partial order (See Section 2.2). Intuitively, given two analysis (abstract) actions $\hat{q}_1, \hat{q}_2$, we calculate the distance according to the length of the "paths" from the actions to their minimal generalization $MG(\{\hat{q}1, \hat{q}_2\})$, compared to the length of their paths to the most general template, $(*, *)$. Paths are defined on the DAG $G = (V, E)$ induced by the actions partial order, s.t. each node $v \in V$ represensts an action, and an edge exists between $\hat{q}_1$ and $\hat{q}_2$ iff $\hat{q}_1 \leq \hat{q}_2$ and $\nexists \hat{q}_3 \neq \hat{q}_1, \hat{q}_3 \neq \hat{q}_2$ s.t. $\hat{q}_1 \leq \hat{q}_3 \leq \hat{q}_2$. We denote by $|\hat{q}_1 \rightsquigarrow \hat{q}_2|$ the size of the shortest path from template $\hat{q}_1$ to $\hat{q}_2$, namely their "graph-distance". Finally, we define the distance for any two actions $\delta_q(\hat{q}_1, \hat{q}_2)$ by their graph-distance from their MG, divided by their graph-distance from $(*, *)$. More formally:

$$\delta_q(\hat{q}_1, \hat{q}_2) := \frac{|\hat{q}_1 \rightsquigarrow MG(\hat{q}_1, \hat{q}_2)| + |\hat{q}_2 \rightsquigarrow MG(\hat{q}_1, \hat{q}_2)|}{|\hat{q}_1 \rightsquigarrow \langle *, * \rangle| + |\hat{q}_2 \rightsquigarrow \langle *, * \rangle|}$$

A.2.2 Display Distance

Our notion of display distance considers the distance between each pair of corresponding layers. We currently support three types of layers: (1) the *data layer*, containing statistics about the curerntly projected tuples and attribuets, (2) *Granularity layer* which describes the level of abstraction for columns that are *grouped-by*, or *aggregated*, and (3) *Mining layer* that represents the results of ML actions. We present below the explicit representation of each layer and its corresponding distance metric, then state the overall display distance notion.

Data layer. Recall that for a set of currently projected attributes A, the data layer $L_D = \{\langle I_U^a, I_N^a, I_H^a \rangle | \forall a \in A\}$ consists of three indices for each column: (1) I_U indicates The number of unique values, (2) I_N marks the number of NULL values, and (3) I_H stands for the normalized value entropy, given for a *categorical* column a containing the unique values $x_1,_2, ..x_n$ as $H = -\sum_{x \in a} \frac{p_x \log p_x}{\log |a|}$, where p_x denotes the *frequency* of the unique value $x \in X$. For *numeric* columns, since they may be sparse, we first perform *equi-width* binning, dividing the value range to B different buckets ranging from the minimal value to the maximal. The normalized entropy $H = -\sum_{b \in B} \frac{p_b \log p_b}{\log |a|}$ where p_b denotes the frequency of column's values associated to the bucket b.

The data distance between data layers, denoted $\delta_{\langle O,A \rangle}(L_D, L'_D)$, gives higher scores if the displays differ in their projected attributes, and lower scores for columns appearing in both displays and that have similar statistics. Formally:

$$\delta_{\langle O,A \rangle}(L_D, L'_D) := |A \triangle A'| + \sum_{a \in A \cap A'} \sum_{j \in \{U,N,H\}} (\frac{1}{3} - \frac{min(I_j^a, I_j'^a)}{3max(I_j^a, I_j'^a)})$$

where $|A \triangle A'| = |(A \cup A') \setminus (A \cap A')|$.

We can show the following:

Proposition A.1 ($\delta_{\langle O,A \rangle}$ is a proper distance metric)**.**

The latter proposition holds, as one can see that $\delta_{\langle O,A \rangle}$ is an addition of two proper metrics - the Jaccard set distance, and the extended Jaccard distance for vectrs. This yields a proper metric.

To scale the distance values between $[0, 1]$ while preserving the metric characteristics, we employ the Steinhaus transform [16]:

$$\widehat{\delta_{\langle O,A \rangle}}(L_D, L'_D) = \frac{2\delta_{\langle O,A \rangle}(L_D, L'_D)}{|A| + |A'| + \delta_{\langle O,A \rangle}(L_D, L'_D)}$$

Granularity layer. As explained earlier in this section, given a set G of grouped/aggregated attributes, the granularity layer is denoted by $L_G = \langle \{I_G^a | \forall a \in G\}, I_C, I_V \rangle$, where for an aggregated column (1) $I_G^a = l(a)$, i.e. the height of the current level in the OLAP hierarchy (when such hierarchy is absent, $l(a)$ is always equal to 1). If the column a is aggregated, then, $I_G^a = I_H^a$, where I_H stand for the normalized value entropy (as defined for numeric columns above). (2) I_C^a denotes the number of different groups and (3) I_V^a marks their size-variance, i.e. $I_V = \sum_{i=1}^{I_C} (p_i - \mu)^2$ where p_i stands for the probability for group i (i.e its size divided by the number of tuples), and μ is the average group size in the current display.

Our granularity distance measure examines the differences in grouping and aggregations currently applied. if a given column is *grouped* in both displays, the difference in granularity (according to the OLAP hierarchy) is measured, and if *aggregated* in both - we compare the normalized value entropy. Last, we compare the difference in the number of present groups, and their size variance. We define first the distance measure for the granularity indices, then depict the notion for two arbitrary granularity layers. More formally, granularity distance function for a single pair of columns, is defined as follows:

$$\delta_g(I_G^a, I_G'^a) := \begin{cases} 1 - \frac{min(I_G^a, I_G'^a)}{max(I_G^a, I_G'^a)} & \text{if } I_G^a \cong I_g'^a \\ 1 & \text{otherwise} \end{cases}$$

Where $I^a_G \cong I'^a_g$ iff column a is either grouped/aggregated in both corresponding displays. Next, similarly to distance notion of the raw-data layer, we extend the above formula for grouping *sets* G and G' as:

$$\delta_g(L_G, L'_G) := |G \triangle G'| + \sum_{a \in G \cap G'} (\delta_g(I^a_G, I'^a_G))$$

and scale, similarly to $\delta_{\langle O,A \rangle}$ by: $\widehat{\delta_g}(L_G, L'_G) := \frac{2\delta_G(G,G')}{|G|+|G'|+\delta_G(G,G')}$

Last, we define the granularity distance w.r.t. the column-wise distance, the number of groups, and their size variance:

Definition A.2 (Granularity Distance)**.** *Given two granularity layers L_G, L'_G having a grouping attribute sets G and G' their distance is defined as:*

$$\delta_G(L_G, L'_G) := \sum_{j \in \{V,C\}} (\frac{1}{3} - \frac{min(I_j, I'_j)}{3max(I_j, I'_j)}) + \frac{\widehat{\delta_g}(L_G, L'_G)}{3}$$

Mining-labels Distance Recall that the mining layer, in our model, denoted by $M = \{(o_i, L_i)\}$, associates each object $o_i \in O$ a (possibly empty) set of labels $L_i \subseteq \mathcal{L}$. However, since two displays may contain different objects, rather than comparing the mining labels attached to individual objects we consider the multiset consisting of all mining labels attached to the objects of the display. We use standard multiset definitions, thereby consider a multiset as a mapping $m : \mathcal{L} \to \mathbb{Z}_{\geq 0}$, where $\mathcal{L}$ is the mining labels domain. $m(x)$ represents the multiplicity of a label $x \in \mathcal{L}$. Thus, to measure the distance between two mining layers, we simply use a multiset variation of the generalized Jaccard index. Formally:

Definition A.3 (Mining Labels Distance)**.** *Given two mining labels multisets M, M' with corresponding mappings m, m' and the labels domain $\mathcal{L}$, their distance is calculated as follows.*

$$\delta_M(M, M') := 1 - \sum_{x \in \mathcal{L}} \frac{min\{m(x), m'(x)\}}{max\{m(x), m'(x)\}}$$

Example A.4. *The mining settings for d_5 and d_{10} convey clustering results. There are 3 clusters in d_5 and only 2 in d_{10}. Their mining distance is given by $\delta_M(M_5, M_{10}) = 1 - \frac{132+86+0}{220+311+40} \approx 0.62$.*

Display Distance

Using the the distance notions defined above, we complete our definition for display distance as a weighted average of the data, granularity, and mining distances.

Definition A.5 (Display Distance)**.** *Given two distinct displays $d = (R, G, M), d' = (R', G', M')$ and a set of weights $W = \{w_R, w_G, w_M\}$ summing to 1, the distance between d and d' is defined by:*
$\delta_d(d, d') := w_R\widehat{\delta_R}(R, R') + w_G\widehat{\delta_G}(G, G') + w_M\delta_M(M, M')$.

Weights may be adjusted according to the importance of each layer in a given analysis UI. E.g., if a UI is focused on *data mining*, then w_M should be set correspondingly. As described in Section 4.2, the system parameters can be tuned by a grid search, using an offline, predictive evluation. Last, we can show that δ_d is a distance metric, and its value range is $[0, 1]$

A.2.3 n-context distance

Putting it all together, we plug the ground metrics δ_q,δ_d in the tree edit distance notion obtaining the context distance metric as follow.

The distance between two n-contexts c_1, c_2 is defined as the minimal cumulative cost of transforming c_1 to c_2 by using a sequence $ES = es_1, es_2, ...es_N$ of *edit* operations on nodes or edges:*delete/add* a node or an edge, and *alter* the label of a node or an edge. The cost of an edit operation γes_i is defined as follows

$$\gamma(es_i) := \begin{cases} 1 & \text{add/delete a node/edge} \\ \delta_q(e, e') & \text{alter edge} \\ \delta_d(v, v') & \text{alter node} \end{cases}$$

Finally, the contexts distance is given by $\delta(T, T') = min\{\gamma(ES)\}$.

B Additional Experiments

Following Section 4.5, in which we provide a brief overview regarding the effect of the system parameters on the predictive accuracy and execution time, we now provide further deatails. We first discuss the parameters effect on the predictive accuracy and coverage, then on the system response times.

B.1 Predictive Accuracy

As described in Section 4.5, we examine the effect of each system parameter on the accuracy measures and the coverage.Recall that in each experiment we varied one parameter while keeping the rest at their default values, as stated in Table 1.

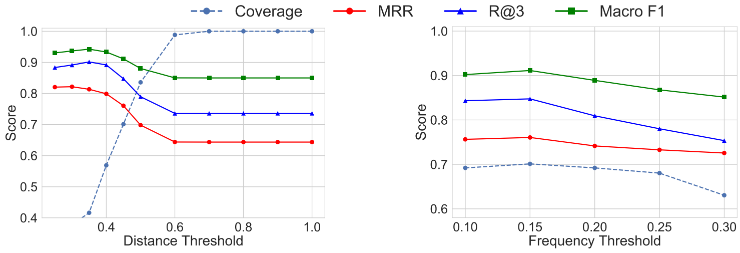
n-context. Figure 4a depicts the predictive accuracy and coverage as a factor of n, the context limit. Observe that all accuracy scores increases as we increase the n-context size, but stabilizes at $n = 3$. On the other hand, larger values of n have negative effect on the coverage, because naturally there are fewer "very similar" large n-contexts.

Size of the top-k set. As depicted in Figure 4b, when we increase k the accuracy score improves (up to the point of $k = 30$ where they stabilize), while the coverage decreases. This is expected, since the input size for the generalization procedure is large enough, the system is able to find more meaningful and relevant abstract actions. The coverage decreases, as there are fewer cases where a large set of similar n-contexts can be found.

Distance threshold. Figure 7a depicts the results when relaxing the distance threshold. Here we can see that a tighter (low) threshold leads to more accurate results, yet at the cost of covering less cases. In the sequel, we also show that setting a tighter threshold has a positive effect of running times due to our bounded top-k search strategy.

Frequency threshold. As shown in Figure 7b, a threshold of 0.1 yields optimal accuracy and coverage results. Naturally, higher thresholds decrease coverage and accuracy, as less recommendations are produced.

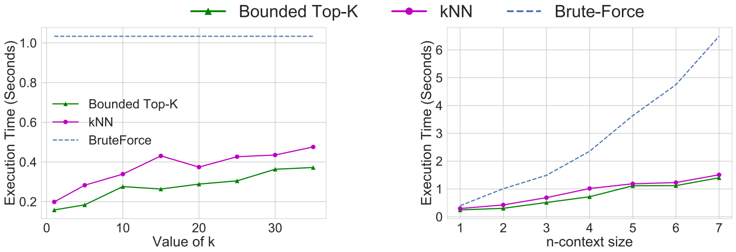
Concluding, we can see that the parameters described above control how tight are the similarity and frequency constraints of our system. Naturally, when finding a large set of highly similar contexts the system will have optimal performance in terms of predictive accuracy, at the cost of predictive accuracy. However, our default configuration retain a fairly good coverage while obtaining high accuracy scores. In the sequel, we show that REACT loaded with the default configuration, outperformed all baselines in the offline evaluation, while proven effective in the online evaluating using real users.



(a) Dist. Threshold

(b) Freq. Threshold

Figure 7: Predictive Accuracy



(a) Size of top-k

(b) Contexts Size

Figure 8: Execution Times

B.2 Execution Times

Here also, at each experiment we varied one parameter while keeping the rest at their default values as in Table 1. The default log size used is $10K$.

We next overview the results. Recall that as baselines, we give the performance of the our solution, while replacing the Bounded top-k strategy (Definition 3.2) with (1) the original metric-tree *unbounded* kNN search, and (2) *Bruteforce* similarity search, which compare the current n-context to all other n-contexts in the log.

Distance Threshold. Figure 5a depicts the execution times as a factor of the distance threshold. We can see that tighten the threshold reduces the execution time for REACT using the Bounded Top-K, while naturally the Unbounded kNN and Bruteforce are not effected. In our default setting ($\theta = 0.45$), we obtain execution time faster than the kNN by 24%.

n-context size. The results depicted in Figure 8b shows a rather linear increase in execution times of REACT as we increase the value of n. While the Brute-force baseline increases quadratically, due to the increasing cost of computing the edit distance for larger trees, REACT's time increase almost linearly. This effect stems from a more efficient traversal of the metric tree, induced by the fact that the n-contexts metric space is more evenly distributed.

Frequency threshold. While increasing the frequency threshold in our generalization process decreased execution times (as more candidate actions are infrequent, thus pruned when performing FIM), it had a marginal effect on the overall execution times (decrease in 0.8 millisecond). Graph omitted.

Size of the top-k set The results depicted in Figure 8a demonstrate a moderate increase in execution time as we increase k. This effect is expected, as retrieving a larger set of nearest neighbors requires a longer traversal in the metric tree. While negligible w.r.t. the overall execution times, varying k also effects the performance of the generalization process, as its input size increase. Indeed, we measured an increase in the running time of the generalization, from 3.2milliseconds (at $k = 10$) to 4.6milliseconds ($k = 40$).