Incremental Based Top-k Similarity Search Framework for Interactive-Data-Analysis Sessions

Oded Elbaz, Tova Milo, and Amit Somech
Tel Aviv University, Israel

ABSTRACT
Interactive Data Analysis (IDA) is a core knowledge-discovery process, in which data scientists explore datasets by issuing a sequence of data analysis actions (e.g., filter, aggregation, visualization), referred to as a session. Since IDA is a challenging task, special recommendation systems were devised in previous work, aimed to assist users in choosing the next analysis action to perform at each point in the session. Such systems often record previous IDA sessions and utilize them to generate next-action recommendations. To do so, a compound, dedicated session-similarity measure is employed to find the top-k sessions most similar to the session of the current user. Clearly, the efficiency of the top-k similarity search is critical to retain interactive response times. However, optimizing this search is challenging due to the non-metric nature of the session similarity measure.

To address this problem we exploit a key property of IDA, which is that the user session progresses incrementally, with the top-k similarity search performed, by the recommender system, at each step. We devise efficient top-k algorithms that harness the incremental nature of the problem to speed up the similarity search, employing a novel, effective filter-and-refine method. Our experiments demonstrate the efficiency of our solution, obtaining a running-time speedup of over 180X compared to a sequential similarity search.

1 INTRODUCTION
Interactive Data Analysis (IDA) is an important procedure in any process of data-driven discovery. It is ubiquitously performed by data scientists and analysts who interact with their data “hands-on” by iteratively applying analysis actions (e.g., filtering, aggregations, visualizations) and manually examining the results. This is primarily done to understand the nature of the data and extract knowledge from it, yet is also fundamental for particular data scientific tasks such as data wrangling and cleaning, feature selection and engineering, and explaining decision models.

However, since IDA is long known as a complex and difficult task, extensive research has been devoted to the development of recommendation systems that assist users in choosing an appropriate next-action to perform at each point in an IDA session. Many of these IDA recommendation systems [2, 15, 16, 26, 43] rely on a similarity comparison between users’ sequences of analysis actions (denoted sessions). They follow the assumption that if two session prefixes are similar, their continuation is likely to be also similar. Hence, they utilize a repository of prior analysis sessions (of the same or other users): Given an ongoing user’s session, such systems first retrieve the top-k most similar session prefixes from the repository, then examine their continuation and use the gathered information to form a possible next-action recommendation [2, 15, 16, 43]. Since these recommender systems are interactive, the efficiency of the top-k session similarity search is critical. However, most previous work focuses on the quality and applicability of the produced IDA recommendations, rather than on their scalability and running time performance.

Our goal in this paper is thus to devise efficient, scalable algorithms for this particular top-k similarity search problem, a significant computational bottleneck in many IDA recommender systems. Specifically, we focus our attention on a dedicated similarity measure for analysis sessions [3], which we denote by SW-SIM. In a comprehensive user study [3], SW-SIM was compared, quality wise, against several alternative similarity measures, and was found to be the most suitable for the context of data analysis sessions. Consequently, it has been adopted by multiple IDA recommender systems and applications, e.g., [2, 4, 30, 41, 44]. SW-SIM is an extension of the well-known Smith-Waterman algorithm [35] for local sequence alignment. Intuitively, given a similarity metric for individual analysis-actions (e.g., filter, aggregation, visualization), SW-SIM compares two sessions s, s’ by aligning them, i.e. matching similar action pairs using an alignment matrix. The measure allows for (yet penalizes) gaps in the alignment, and gives a higher weight to the more recent actions in the session, since they are expected to have higher relevance on the current user’s intent.

Given a current user session and a sessions repository, a naive top-k search may be done by sequentially iterating over all sessions in the repository, computing the SW-SIM similarity score of the current session w.r.t. each of their prefixes, then selecting the top-k prefixes with the highest score. This simple algorithm, however, is prohibitively time consuming: Our experiments show that even when employed on a medium size repository of 10K sessions, the naive sequential search takes more than 17 seconds to complete. This is excessively high for a real-time response.

To optimize it, two key challenges must be addressed:
1. A single similarity comparison of two sessions is expensive to begin with. Computing SW-SIM requires to construct an alignment matrix, which results in a high computation time (quadratic in the mean session length).
2. Employing existing optimizations is highly non-trivial. Since SW-SIM is a non-metric similarity measure, existing optimizations for similarity search e.g. [8, 33, 37] are inadequate. Furthermore, because analysis sessions are compound sequences of complex analysis actions, they do not have a numeric vector representation, which is a requirement by other top-k optimization works [33]. (See Section 2 for an elaborated discussion.)

A key observation underlying our work is that the user session progresses incrementally with the top-k similarity search performed by the IDA recommender system at each step. We exploit this property to address the two challenges mentioned above: Our first, rather direct optimization, speeds up the processing of a single similarity comparison between two sessions by utilizing
previous-step computations. Our second, more sophisticated optimization, tackles the top-k search by employing a novel filter-and-refine technique. It employs lower and upper bounds stemming from the incremental growth of the sessions accompanied by a dedicated index structure. Our experiments demonstrate that using our optimizations, a speedup of more than 180X is obtained compared to a non-optimized sequential search. Importantly, this is done while retaining a perfect accuracy of results, since our algorithms are exact.

We next explicitly state our assumptions and focus, then summarize our contributions and paper organization.

**Paper Focus and Assumptions.**

1. Although the particular IDA settings may vary, many IDA recommender systems rely on a top-k prefixes search. IDA settings may vary between systems, in terms of, e.g., the type of allowed analysis actions, structure of the data, user expertise, etc. However, the incremental, session-like nature of the process characterizes most IDA settings. Our paper is thus aimed at solving a computational bottleneck in a growing number of IDA-dedicated recommendation systems [2, 15, 16, 26, 43] that rely on finding the top-k most similar session prefixes in order to generate recommendations. Our solution is generic, suitable for a wide range of IDA platforms, regardless of their particular settings.

2. **SW-SIM is currently the most-suitable measure for IDA sessions similarity.** A multitude of definitions exist for measuring the similarity between two arbitrary sequences, such as *Dynamic Time Warping* for time series and *Smith-Waterman* for DNA sequences. In the context of data analysis sessions, SW-SIM is considered the most comprehensive and suitable (See [3] for a comparative study).

3. **The focus of this paper is on performance and scalability, rather than on the quality of the produced recommendations.** Different IDA recommender systems may use the selected top-k sessions in different ways in order to generate recommendations, possibly using additional information such as the data properties, user profiles, etc. But in all of them - response time optimization of the top-k search is clearly a critical issue. We therefore focus on performance and scalability rather than discussing the quality of the recommendations produced. For the latter, we refer the reader to works e.g. [2, 15, 16, 26, 43] in which the focus is on the quality of recommendations produced.

**Technical Contributions & Paper Organization.**

**Section 3:** We provide a simple, generic model for representing IDA sessions, and describe the alignment-based session similarity notion. Based on this model, we develop a formal definition for the incremental top-k similarity search problem.

**Section 4:** We describe an optimization for the incremental construction of a session alignment matrix and means to derive prefixes similarity scores from it.

**Section 5:** We present a novel threshold-based algorithm for the incremental top-k search, which utilizes effective similarity lower and upper bounds, also stemming from the incremental nature of the search problem.

**Section 6:** We demonstrate the efficiency of our solution via extensive experiments on artificial and real-life session repositories.

We begin by reviewing related work (Section 2), and finally present our conclusions and limitations in Section 7.

## 2 RELATED WORK

There is vast literature on sequences similarity, alignment, and top-k search. We survey works in numerous application domains, explaining why existing optimizations are inadequate for the context of IDA.

**Bioinformatics-based optimizations for sequence alignment.** Local alignment techniques are known to be extremely useful in the bioinformatics domain for tasks such as DNA and protein sequencing [22, 25, 39]. In this application domain, alignment is often performed between a query sequence and an extremely long *reference* string (e.g. a DNA genome), often from a fixed, small size alphabet (comprises, e.g., six nucleotides). Consequently, biology-driven optimizations such as [14, 18, 28, 40] exploit this particular property (small alphabet size) in order to index the large reference string. These solutions typically use a prefix/suffix tree or hash tables mapping small, common substrings to their location in the reference string. In contrast, in our context the “alphabet” (namely the analysis-actions space) is unbounded and, as explained in the next section, there is a predefined notion of distance between letters (analysis actions in our context). Such solutions, therefore, cannot be directly employed in our case.

**Edit Distance and Dynamic Time Warping (DTW).** Edit distance and DTW are popular alignment-based measures that are used in numerous application domains such as textual autocorrection, speech recognition and the comparison of time series. However, as noted in [3], these measures lack important properties for the context of IDA sessions. Classical edit-distance, which originated from the comparison of textual strings, assumes that letters are either *identical* or *mismatched*, which is inadequate for the context of IDA. This is because the alphabet, containing individual actions, is much larger and more complex. Also, the compound similarity of individual “letters” (actions) should be taken into consideration. DTW, on the other hand, does support the employment of a designated distance metric for the sequences’ objects, yet it is not compatible with the notion of *gaps* [32]. Also, both measures do not support the important time-discount feature [3], which allows giving a higher weight to the more recent actions in the session. Optimization works for these measures are typically based on specific properties of the measures or application domain [19, 38].

**General solutions for similarity search in a non-metric space.** SW-SIM is a non-metric similarity measure, thus many optimizations that rely on the *triangle inequality* property (e.g. [8, 27]) cannot be used here. Other solutions for non-metric spaces, e.g. [6, 7, 19, 34], assume that the sequences’ objects are numeric vectors (e.g. as in Minkowski distance, Cosine distance), as opposed to sequences of abstract objects (analysis actions in our case). But even ignoring this, employing any such solutions in our context requires indexing each session prefix, which may be prohibitively large (typically by an order of magnitude compared to the number of sessions). Existing solutions for optimized top-k search supporting arbitrary, non-metric similarity measure are Constant Shift Embedding (CSE) [31] and NM-Tree [34]. We examine these solutions in our experimental evaluation (Section 6.2) and show that our algorithm is consistently superior, obtaining a performance improvement of at least two orders of magnitude.

**Incremental keyword query autocompletion.** Incremental computation is used for optimization in a variety of database applications, e.g., interactive association-rules and sequence mining [29], but most notably for incremental text autocompletion [10, 17, 24], the latter solutions efficiently perform a top-k keyword
similarity search at each key stroke of the user (using string edit distance). However, such solutions also exploit the rather small alphabet size and use a trie-index, a dedicated prefix tree for strings. As explained above, in our case the “alphabet” (i.e., the analysis-actions space) is complex and unbounded. Last, [21] and [20] suggest algorithms for incrementally computing the string edit distance of Ax and B (where x is an additional character) in O(|A| + |B|) by using the distance matrix computed for A and B. We employ similar principles for our single-pair incremental alignment computation (Section 4), yet, importantly, augment them with novel techniques for efficient top-k similarity search.

**Similarity search of scientific/business workflows.** Similarity search has also been considered in the context of workflows [12, 23, 36] in two main contexts - searching for workflow specifications, and searching for workflow traces. They employ, however, a different notions of similarity focusing mainly on the structure/nesting of the workflow-specification components (different from the the unstructured, free-form nature of IDA sessions). Also, to our knowledge, these works do not address the incremental nature of the top-k similarity search problem, that is more typical for IDA sessions than for workflows.

3 PRELIMINARIES

We start by providing basic definitions for IDA, then define the incremental top-k similarity search problem.

**Interactive Analysis Sessions.** In the process of IDA, users investigate datasets via an interactive user interface which allows them to formulate and issue analysis actions (e.g., filter, grouping, aggregation, visualization, mining) and to examine their results. Formally, we assume an infinite domain \( s \) of analysis actions and model an analysis session as a sequence of actions \( s = (q_1, q_2, \ldots, q_n) \) \( q_i \in S \). We use \( s[i] \) to denote the \( i \) ’th action in \( s \) (\( q_i \)) and \( s_t \) to denote the session’s prefix up to \( q_i \), i.e. \( s_t = (q_1, q_2, \ldots, q_i) \). Therefore, \( s = s_n\), and \( s_0 \) is the empty session. An ongoing user session, denoted \( u \), is built incrementally. At time \( t \) the user issues an action \( q_t \) then analyzes its results and decides whether to issue a next action \( q_{t+1} \) or to terminate the session. The session (prefix) \( u_t \) at time \( t \) thus consists of the actions \( (q_1, \ldots, q_t) \), where the following actions in the session, i.e., \( q_{t+1} \ldots q_n \) are not yet known.

In this work we consider the case of comparing the similarity of the user session \( u \) to other sessions (and parts thereof) in a given repository, denoted \( S \), containing prior analysis sessions performed by the same or other users. To formally define sessions similarity, let us first consider the similarity of individual analysis actions, then generalize to sessions.

**Individual Actions Similarity.** Given a distance metric for individual analysis actions \( \Delta : Q \times Q \rightarrow [0, 1] \) over the actions domain, the action similarity function \( \sigma(q_1, q_2) \) is the complement function defined by \( \sigma(q_1, q_2) = 1 - \Delta(q_1, q_2) \) for all \( q_1, q_2 \in Q \). Several distance/similarity measures for many kinds of analysis actions, e.g. for SQL and OLAP queries visualizations, and web-based analysis actions have been proposed in the literature (e.g. [3, 16, 26]) and our framework can employ any of them as long as the corresponding measure defines a metric space.

**Session Similarity** As mentioned in Section 2, there are several ways to lift the similarity of individual elements into similarity of sequences [11]. To assess which measure is the most suitable

\[\begin{array}{cccccc|cccccc}
\hline
a & b & c & A & B & a & b & c & A & B \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.24 & 0.19 & 0.13 & 0.66 & 0.38 & \mathbf{0.48} & 0.43 & 0.37 & 0.30 & 0.23 \\
0 & 0.19 & 0.53 & 0.47 & 0.58 & 1.47 & \mathbf{0.43} & 1.67 & 1.00 & 0.93 & 0.85 \\
0 & 0.30 & 0.47 & 0.53 & 1.28 & 1.35 & \mathbf{0.59} & 1.60 & 1.43 & 1.35 & 1.26 \\
0 & 0.23 & 0.66 & 0.58 & 1.19 & 2.26 & \mathbf{0.52} & 1.32 & 1.35 & 1.32 & 1.78 \\
\hline
\end{array}\]

(a) Session \( \phi \)

(b) Session \( \psi \)

**Figure 1: Alignment Matrices for \( u_t = "ABAB" \)**

for comparing analysis sessions, the authors of [3] formulated desiderata for an ideal session similarity measure, based on an in-depth user study:

- It should take the actions’ order of execution into consideration, i.e. two sessions are similar if they contain a similar set of actions, performed in a similar order.
- “Gaps” (i.e subsequences of non-matching actions) should be allowed yet penalized.
- Long matching subsequences should be better rewarded than shorter matching subsequences.
- Recent actions in the sessions are more relevant than old ones.

They conclude that the only measure respecting all of the above mentioned desiderata is the Smith-Waterman similarity measure [35], a popular measure for local sequence alignment, and propose an extension suitable for the context of analysis actions, denoted \( SW-SIM \) in our work. We next describe the measure.

The similarity score of two sessions is defined recursively, on increasingly growing prefixes, with the base of the recursion being the empty prefix. At each point, the similarity of the pair of actions at the end of the prefixes is considered, and the best option, score-wise, is chosen. The two actions may either be matched, in which case an award proportional to their similarity is added to the accumulated score for their preceding prefixes, or, alternatively, one of the actions is skipped over. In this case a linear gap penalty \( 0 \leq \delta \leq 1 \) is deducted from the accumulated score. To reflect the fact that the matching/skipping of older actions is less important than that of recent ones, rewards/penalties are multiplied by a decay factor \( 0 \leq \beta \leq 1 \) with an exponent reflecting how distant the actions are from the sessions’ end. More formally, the sessions similarity is defined using an alignment matrix.

**Definition 3.1 (Alignment Matrix, Similarity Score).** Given sessions \( s, s' \) of lengths \( n, m \) resp., their alignment matrix \( A_{s,s'} \in [0,1]^{(n+1)\times(m+1)} \) is recursively defined as follows. For \( 0 \leq i \leq n, 0 \leq j \leq m \):

\[
A_{s,s'}[i,j] = \begin{cases} 
0, & \text{if } i = 0 \lor j = 0 \text{, else:} \\
A_{s,s'}[i-1,j-1] + \sigma(s[i], s'[j])\beta^{(n-i)+(m-j)} & \\
\max \left( A_{s,s'}[i-1,j] - \delta^{(n-i)+(m-j)}, A_{s,s'}[i,j-1] - \delta^{(n-i)+(m-j)}, A_{s,s'}[i-1,j-1] - \delta^{(n-i)+(m-j)} \right) 
\end{cases}
\]

The similarity score between \( s \) and \( s' \) is defined as:

\[
Sim(s, s') := A_{s,s'}[n,m]
\]

Note that even though the distance between individual actions forms a metric space, as noted in the introduction, \( SW-SIM \) does not induce a metric on IDA sessions.
The optimal values for the decay factor $\beta$ and the gap penalty $\delta$ are typically chosen by the system administrator using an extrinsic evaluation, as explained in Section 5.1. The following example illustrates the definition.

Example 3.2. For simplicity, assume that the action space is represented by the English letters, both uppercase and lowercase, and assume the following action similarity for any two distinct uppercase letters $X, Y$ and their corresponding lowercase versions $x, y$.

\[
\begin{align*}
\sigma(X, X) &= 1 \\
\sigma(x, x) &= 1 \\
\sigma(X, Y) &= \sigma(x, y) = 0.5 \\
\sigma(Y, x) &= \sigma(Y, y) = 0.1 \\
\sigma(X, y) &= \sigma(Y, X) = 0
\end{align*}
\]

Identical letters are maximally similar, and two instances of the same letter, but in different case (upper/lower), are 0.5 similar. Different letters but of the same case are 0.1 similar. Let $u_t = "ABAB"$ be the current user session, and consider two repository sessions $\phi = "abcAB\"$ and $\psi = "ABAb\"$. Intuitively, according to the desiderata above, we expect $\phi$ to be more similar to $u_t$ than $\psi$, as their most recent suffix ("AB") is identical. This is reflected also in the alignment matrices of $\phi$ and $\psi$, depicted in Figure 1a and Figure 1b (resp.), when setting the gap penalty $\delta = 0.1$ and decay factor $\beta = 0.9$. The bottom-right cell in each matrix reflects the alignment score of the two sessions. The highlighted cells describes the alignment "trace", namely the cells chosen among the three options in alignment formula when advancing to the next step in the final score computation. Specifically, when the highlighted trace moves vertically/horizontally we have a gap, and when it moves diagonally the corresponding actions are matched. As we can see, the similarity score $\text{Sim}(u_4, \phi)$ is higher than $\text{Sim}(u_4, \psi)$.

Our definition of SW-SIM follows that of [3, 35] with a minor adaptation to our context. First, we define the similarity score as the bottom-right cell of the alignment matrix, as opposed to the maximal cell value in [3, 35]. This is because we focus on measuring the similarity of the current session to all session prefixes in the repository\(^2\). Second, we apply a decay factor to both actions matches and gaps, as opposed to only matches in [3], since their significance decreases as the session advances. Also, note that [3] suggests dynamically setting the decay factor and gap penalty, which are both constants in our case. As we shall see, the use of constant parameter values facilitates effective optimization via computation factorization.

Problem Definition. To assist the user in choosing an appropriate next-action, IDA recommender systems search the repository $S$ to identify session prefixes that are most similar to $u_t$ - the current, ongoing user session at time $t$. The continuation of the retrieved sessions is then processed by the recommender systems and used to derive a next-action recommendation for $u_t$.\(^3\)

Let $\text{Prefix}(S)$ be the set consisting of all session prefixes in repository $S$, i.e. $\text{Prefix}(S) = \{u_i | u \in S, 1 \leq i \leq |S|\}$. Since the user session $u = (q_1, q_2, \ldots)$ is built incrementally, at each step $t = 1, \ldots, |u|$ we are given $u_t$ and wish to identify its top-k most similar session prefixes in the repository.

\(^2\)This is done by IDA recommendation systems in order to process the continuation of matching session-prefixes to next-action recommendations.

\(^3\)Other information is often used as well, e.g. the intervisits or frequency of the actions to be recommended, the user profile, etc.

\[u \setminus \phi = \begin{bmatrix} \text{a} & \text{ab} & \text{abc} & \text{abcA} & \text{abcAB} \\ 0.35 & 0.90 & 0.71 & 1.32 & 2.28 \end{bmatrix}; \begin{bmatrix} \text{u} & \phi \end{bmatrix} = \begin{bmatrix} \text{A} & \text{AB} & \text{ABA} & \text{ABAB} & \text{ABABC} \\ 0.80 & 1.81 & 1.67 & 2.09 & 1.78 \end{bmatrix} \]

\[u \setminus \psi = \begin{bmatrix} \text{A} & \text{AB} & \text{ABA} & \text{ABAB} & \text{ABABC} \\ 0.62 & 1.53 & 1.47 & 1.78 & 2.19 \end{bmatrix} \]

Figure 2: Similarity vectors for $u_4$ and $u_5$

Definition 3.3 (Incremental Top-k Similarity Search). Given a user session $u_t$ at time $t$ and a sessions repository $S$, the set $\text{top}_k(u_t, S) \subseteq \text{Prefix}(S)$ consists of $k$ session prefixes s.t. $\forall s \in \text{top}_k(u_t, S), \forall s' \in \text{Prefix}(S) \setminus \text{top}_k(u_t, S): \text{Sim}(u_t, s) \geq \text{Sim}(u_t, s')$. The incremental search problem is to compute, at each $t = 1, \ldots, |u|$, the set $\text{top}_k(u_t, S)$.

For simplicity, we assume that in the course of a user session $u$, the repository $S$ is unchanged. In Section 5.5 we discuss the minor changes required in our framework to support the case of a dynamic repository where sessions are incremented or added.

Last, note that while session prefixes may be long and contain "historical" actions of less importance, SW-SIM (as explained above) favors recent, later actions over old ones. Also, it is easy to show that in SW-SIM, the similarity of a prefix $u_i = (q_1, q_2, \ldots, q_i)$ is always higher (or equal) than the similarity of any shorter sub-session of $s$ ending in $q_i$.

4 Incremental Similarity Computation

We first optimize the similarity calculations between the current user session $u$ and all prefixes of a single repository session $s$. Rather than computing the alignment matrix from scratch for each prefix of $s$, we show that: (1) it is sufficient to compute one alignment matrix between two sessions, then use it to simultaneously derive all prefixes similarity scores, and (2) the alignment matrix $A_{u_t, s}$ at time $t$ can be efficiently constructed by reusing the previous matrix $A_{u_{t-1}, s}$ computed at time $t - 1$.

Given a current user session $u_t$ and a repository session $s$ of size $|s|$, we define a similarity vector, $\sim(u_t, s)$ containing the similarity score of $u_t$ and each prefix of $s$, i.e.,

\[\sim(u_t, s) = [\text{Sim}(u_t, s_0), \text{Sim}(u_t, s_1), \ldots, \text{Sim}(u_t, s_{|s|})] \]

We use $\sim(u_t, j)$, where $j = 1, \ldots, |s|$, to denote the $j$ element in the vector. The similarity vector $\sim(u_t, s)$ can be derived from the alignment matrix $A_{u_t, s}$ using the following observation:

Observation 4.1. Given an alignment matrix $A_{u_t, s}$

\[0 \leq j \leq |s| : \sim(u_t, s)[j] = \frac{A_{u_t, s}[t, j]}{\beta^{|s|-j}} \]

Namely, the similarity score of $u_t$ and prefix $s_j$ is derived from their corresponding element in the alignment matrix, $A_{u_t, s}[t, j]$, by readjusting the decay factor $\beta$. The proof is derived from the more general observation that for arbitrary prefixes $s_i$ and $s'_j$ of session $s$ and $s'$:

\[A_{s_i, s'_j}[i, j] = \frac{A_{s_i, s}[i, j]}{\beta^{|s'_j|-j}} \]

Example 4.2. To continue with our running example, the similarity vectors $\sim(u_4, \phi)$ and $\sim(u_5, \psi)$ are given in the first row of the tables in Figure 2a, and Figure 2b, resp. According to Observation 4.1, an element in the similarity vector, e.g. $\sim(u_4, \phi)[3]$, may be computed directly from the corresponding cell in the
alignment matrix of $u_4$ and $\phi$: $\tilde{\text{Sim}}(u_4, \phi)[3] = \frac{A_{u_4, \phi}[4, 3]}{p^3} = 0.58, 0.81 = 0.71$.

Further exploiting the incremental nature of the problem, we next show how to efficiently construct a similarity vector $\tilde{\text{Sim}}(u_t, s)$ at time $t$, from that computed at time $t - 1$, thereby avoiding explicitly building the entire alignment matrix. To do so, we generalize the dynamic programming construction of the alignment matrix (Definition 3.1). Vector entries at time $t$ are computed by reusing entries in the previous similarity vector $\tilde{\text{Sim}}(u_{t-1}, s)$, computed at time $t - 1$ (see the colored parts of the formula in Proposition 4.3).

**Proposition 4.3.** For every repository session $s$, user session $u$ and time $t > 1$,

$$\tilde{\text{Sim}}(u_t, s)[j] = \begin{cases} 0, & \text{if } j = 0, \\ \max & 0, \text{if } j \neq 0, \text{else}: \\ \tilde{\text{Sim}}(u_{t-1}, s)[j-1] \beta^2 + \sigma(u_t[t], s[j]) \\ \tilde{\text{Sim}}(u_{t-1}, s)[j-1] \beta - \delta \\ \tilde{\text{Sim}}(u_t, s)[j] \beta - \delta \\ 0 \end{cases}$$

The proof is obtained by employing Equation 1 on each case of the conditional definition.

**Example 4.4.** Continuing with our example, assume that the user now issues a new action “c”, i.e., at $t = 5, u_5 = "ABABc"$. The new similarity vectors $\tilde{\text{Sim}}(u_5, \phi)$ and $\tilde{\text{Sim}}(u_5, \psi)$, are depicted in the bottom row of Figures 2a and 2b, resp. As before, the similarity scores $\tilde{\text{Sim}}(u_5, \phi)$ and $\tilde{\text{Sim}}(u_5, \psi)$, appear in the right-most cell of the vectors. Using Proposition 4.5 we can derive each value in the new vectors from the previous corresponding similarity vectors at time $t = 4$. For example, the fourth element in $\tilde{\text{Sim}}(u_5, \psi)$ is given by:

$$\tilde{\text{Sim}}(u_5, \psi)[4] = \max \left\{ \tilde{\text{Sim}}(u_4, \psi)[3] \beta^2 + \sigma("c", "b") \right\}$$

5 INCREMENTAL TOP-K ALGORITHMS

We are now ready to address the incremental top-k problem - given an ongoing user session $u$, retrieve the set $\text{top}_k(u_t, S)$ of similar prefixes at each time $t = 1, \ldots, |u_t|$. We first present a simple algorithm, denoted I-TopK, that iterates over the repository $S$ and computes the similarity vector for each session. We then show how a much faster variant, denoted T-TopK, is obtained by employing a novel, incremental-based filter-and-refine approach.

**Algorithm 1 T-TopK($u_t, S, k$).**

- **Input:** $u_t$ - current user session,
  $S$ - session repository,
  $k$ - size of the top-k set.
- **Output:** $\text{top}_k(u_t, S)$ - a set of top-k most similar session prefixes to $u_t$.

1. $top \leftarrow \text{MaxHeap}(k)$
2. if $t = 1$ then
3. Use I-TopK to obtain $\text{top}_k(u_t, S)$
else
5. Compute $\text{inf}^t$, the lower-bound similarity threshold
6. $C \leftarrow \{ s | s \in S \land \sup^t(s) \geq \text{inf}^t \}$
7. for session $s \in C$ do
8. $m \leftarrow$ latest time $t' < t$ that $\tilde{\text{Sim}}(u_{t'}, s)$ was computed.
9. for $(i = m + 1; i \leq t; i++)$ do
10. Compute $\tilde{\text{Sim}}(u_t, s)$ using Proposition 4.3
11. for $(j = 1; j \leq |s|; j++)$ do
12. $\text{top}.\text{push}(\tilde{\text{Sim}}(u_t, s)[j], s_j)$
13. if $|\text{top}| = k$ then
14. $C \leftarrow C \setminus \{ s | \sup^t(s) < \min\text{Score}(\text{top}) \}$
15. return $\text{top}$

5.1 Iterative Top-k Algorithm (I-TopK)

Algorithm I-TopK takes the following as input: the current session $u_t$, the sessions repository $S$, and the desired size $k$ of the top-k set. It iterates over the repository, computing the similarity vector $\tilde{\text{Sim}}(u_t, s)$ for each $s \in S$ by employing Proposition 4.3, i.e. by using the similarity vector $\tilde{\text{Sim}}(u_{t-1}, s)$ calculated at the previous iteration ($t - 1$). The top-k similarity scores (along with their corresponding prefixes) are maintained in a max-heap of size $k$, which will contain the exact set $\text{top}_k(u_t, S)$ of the top-k most similar prefixes to $u_t$ at the end of the loop.

The time complexity of I-TopK is $O(n|S|\lambda)$, since it iterates over a session repository of size $|S|$ and computes a similarity vector in $O(|S|\lambda)$. The max-heap is maintained in a negligible cost of $O(\log k)$, as we assume that $k << |S|$. As for space complexity, I-TopK requires storing the previous similarity vectors computed at $t - 1$, therefore it requires $O(|S|\bar{|S|})$ space, where $\bar{|S|}$ is the average session size.

5.2 Threshold-Based Algorithm (T-TopK)

The T-TopK algorithm follows a filter-and-refine paradigm, s.t. in the filter step, repository sessions are pruned according to an overestimation of their similarity scores. Then, during the refinement step, the similarity vectors are computed only for candidate sessions passing the filter step. Importantly, T-TopK is guaranteed to be exact since the filter step always overestimates the true similarity scores and underestimates the filter thresholds. The novelty and efficiency of the algorithm stem from exploiting the incremental nature of the search problem.

We first describe the outline of the T-TopK algorithm, then in Sections 5.3 we present the techniques used for its filter step, i.e., the efficient calculation of the similarity lower and upper bounds.

**T-TopK Algorithm Outline.** Algorithm 1 depicts the outline of the T-TopK algorithm. For a given user session $u_t$, a sessions repository $S$ and a number $k$, the prefixes set $\text{top}_k(u_t, S)$ is retrieved in the following manner. First, if $u_t$ contains only one
action, the I-TopK algorithm will be used, since we have no previous prefix to rely on. Otherwise we employ a filter-and-refine process as follows.

Filter Step. We first find candidate sessions that are likely to contain prefixes in the top-k set \( \text{top}_{k}(u_{t}, S) \):

1. **Form a global similarity threshold.** We first compute an initial lower bound threshold, denoted \( \inf^{t} \), for the similarity score of a prefix to be a member of \( \text{top}_{k}(u_{t}, S) \). The threshold \( \inf^{t} \) underestimates the true minimal similarity score (Line 5).

2. **Compute a similarity upper-bound for each session.** We then form a similarity upper-bound for each session \( s \), denoted \( \sup^{t}(s) \), that overestimates the maximal similarity of a prefix in \( s \) to the current user session \( u_{t} \).

The algorithm then filters the repository \( S \) by retrieving all sessions having similarity upper-bounds greater than \( \inf^{t} \) (Line 6). Importantly, since the lower bound is underestimated and the upper bounds are overestimated - no false negatives can occur. In Section 5.3 we describe how the similarity lower and upper bounds are defined and efficiently calculated.

Refinement Step. We employ a refinement step over the candidate sessions and calculate the exact similarity scores using their similarity-vectors. Recall that to efficiently construct a similarity vector (Proposition 4.3) at time \( t \), we need to have the vector of time \( t-1 \). However, we may not have calculated a similarity vector at time \( t-1 \) if a session was pruned in the filter step. Thus, for each such candidate session we reconstruct, if necessary, its previous vectors from time \( t \) (We discuss below how this computation can be performed in user idle-times, allowing T-TopK a further speed-up), then compute the current similarity vector \( \Sim(u_{t}, s) \) (LINES 9-10).

Last, we iteratively push each element in the similarity vector into a max-heap \( \text{top} \) of size \( k \) (LINES 11-12), and further prune candidate sessions if their upper bound is lower than the minimum score in \( \text{top} \) (LINES 13-14). Finally, the max-heap \( \text{top} \) holds the set \( \text{top}_{k}(u_{t}, S) \), containing the top-\( k \) most similar prefixes to \( u_{t} \).

Offline computation in user idle-times. User idle-times occur between two consecutive actions - while the user examines the results of her current action, before executing the next one (which typically takes several seconds). Our algorithm utilizes such idle-times, to compute similarity vectors skipped at previous iterations offline, so that they are already available when needed in the top-k search.

Correctness of the T-TopK algorithm. We next sketch the correctness proof, showing that the T-TopK algorithm is correct, i.e., always retrieves the exact set of top-k most similar session prefixes:

Let \( \text{top} \) denote the output of the algorithm. We need to show that \( \text{top}_{k}(u_{t}, S) = \text{top} \). We will consider \( t > 1 \) (the case of \( t=1 \) is trivial). First, we show that for an arbitrary prefix \( s_{j} \) of a repository session \( s, s_{j} \notin \text{top}_{k}(u_{t}, S) \) \( s_{j} \rightarrow \text{top} \). If \( s \notin \text{top}_{k}(u_{t}, S) \) then \( \Sim(u_{t}, s) \geq \inf^{t} \) (as \( \inf^{t} \) is the similarity lower bound).

Hence, for the upper bound \( \sup^{t}(s) \) of session \( s \) we can easily see that \( \sup^{t}(s) \geq \inf^{t} \), thus \( s \in C \), i.e. \( s \) is retrieved and processed as a candidate session. The proof for the case that \( s_{j} \in \text{top}_{k}(u_{t}, S) \) \( s_{j} \rightarrow \text{top} \) follows similar lines.

To complete the picture we still need to explain how (1) the similarity lower bound \( \inf^{t} \), and (2) the upper bounds \( \sup^{t}(s) \) are computed and compared.

### 5.3 Incremental-Based Similarity Bounds

We first explain how the similarity lower and upper bounds are defined, then describe how the candidate sessions are efficiently retrieved.

#### 5.3.1 Similarity Threshold (Lower Bound)

The threshold \( \inf^{t} \) forms a lower-bound (underestimated) for the similarity score of a prefix to be a member of \( \text{top}_{k}(u_{t}, S) \). Intuitively, we use the sessions in the (already computed) top-k set of time \( t-1 \) as a potential representative for the top-k set of time \( t \), and define our threshold w.r.t. them. Let \( S_{\text{top}}^{t-1} \subseteq S \) denote the set of sessions with prefixes in \( \text{top}_{k}(u_{t-1}, S) \). The similarity threshold is defined by:

\[
\inf^{t} = \min \text{Score}(\text{top}_{k}(u_{t}, S_{\text{top}}^{t-1}))
\]

As the size of \( S_{\text{top}}^{t-1} \) at most \( k \), \( \inf^{t} \) is computed efficiently.

The proof that \( \inf^{t} \) always underestimates the exact lower-bound can be immediately obtained from the simple observation that:

\[
\forall S' \subseteq S, \min \text{Score}(\text{top}_{k}(u_{t}, S')) \leq \min \text{Score}(\text{top}_{k}(u_{t}, S'))
\]

This stems from the fact that the lowest possible \( \inf^{t} \) is obtained when the last query in \( u_{t} \) is not aligned in \( any \) of the sessions in \( S_{\text{top}}^{t-1} \) (hence a gap penalty is absorbed).

**Example 5.2.** Assume that \( k = 1 \) (i.e. we are interested in the top-1 similar prefixes), and consider the sessions \( u, \psi, \phi \) from our running example. Considering the similarity vectors at \( t = 4 \) (detailed in Example 4.2), the most similar prefix is \( \phi_{5} \) and thus \( \text{top}_{k}(u_{t-1}, (\phi, \psi)) = \{\phi_{5}\} \). Consequently, its corresponding session is in the set \( S_{\text{top}}^{t-1} = \{\phi\} \). For computing the threshold \( \inf^{t} = \min \text{Score}(\text{top}_{k}(u_{t}, S_{\text{top}}^{t-1})) \), we construct the similarity vector \( \Sim(u_{t}, \phi) \) (bottom row in Figure 2a), and take the maximal score among its elements, thus \( \inf^{t} = 1.95 \).

#### 5.3.2 Upper-Bounding Similarity Scores

Let \( \sup^{t}(s) = \max_{1 \leq j \leq |s|} \Sim(u_{t}, s_{j}) \) denote the tight, exact upper bound for the similarity score of all prefixes in \( s \). Recall that for sessions in \( S_{\text{top}}^{t-1} \), we already computed their similarity vectors when computing the lower bound \( \inf^{t} \), therefore their tight bound is already at hand. Nevertheless, for the rest of the sessions, i.e. \( s \in S \setminus S_{\text{top}}^{t-1} \) we show that the incremental nature of the computation can be used to form \( \sup^{t}(s) \), an effective over-approximation to \( \sup^{t}(s) \). An important observation is that we are only interested in upper-bounding the scores of sessions containing at least one prefix with a similarity scores higher than \( \inf^{t} \) (otherwise it can be safely pruned). We therefore define a proper similarity upper bound as follows:

**Definition 5.3 (Proper upper bound).** \( \sup^{t}(s) \) is a proper upper bound w.r.t. \( u_{t} \) and \( S \), if for every session \( s \) having some prefix \( s_{j} \) where \( \Sim(u_{t}, s_{j}) > \inf^{t} \), we have that \( \sup^{t}(s) \geq \sup^{t}(s_{j}) \).

A second important observation, is that unless the last action in \( u_{t} \), denoted \( q_{t} \), is aligned to an action in \( s \in S \setminus S_{\text{top}}^{t-1} \), \( s \) can be safely pruned.
Observation 5.4. For a session \( s \in S \setminus S^{t-1} \) if computing \( \text{Sim}(u_t, s) \) the action \( q_t \) is not matched with any of \( s \)'s actions, then for every prefix \( s_j \) of \( s \), \( \text{Sim}(u_t, s_j) \leq \inf^t \).

This is because if \( q_t \) is not matched with an action in \( s_j \) it absorbs a gap penalty, i.e., \( \text{Sim}(u_t, s_j) = \text{Sim}(u_{t-1, s_j}, \beta) - \delta \) (Proposition 4.3). On the other hand, since \( s_j \notin \text{top}(u_{t-1}, S) \) we know that \( \text{Sim}(u_{t-1, s_j}) \leq \minScore(\text{top}_k(u_{t-1}, S)) \). Then, using Observation 5.1 we can see that \( \text{Sim}(u_t, s_j) \) can never exceed \( \inf^t \).

Consequently, in the analysis below we ignore sessions where \( q_t \) is not matched with any of \( s \)'s actions, and for brevity, unless stated otherwise, whenever we refer to a session \( s \) we mean one in \( S \setminus S^{t-1} \) where \( q_t \) has a match.

We next provide two ways to overestimate \( \sup^t(s) \). In our efficient implementation of the filter procedure for candidate session retrieval (to be detailed in the next subsection), we use the first bound to quickly prune irrelevant sessions, then use the second one for further pruning the candidates set.

First bound (\( B_1 \)). The following observation shows that we can bound \( \sup^t(s) \) from above using the top-\( k \) set of the previous iteration and the maximal similarity of the session’s individual actions to the new action \( q_t \).

Observation 5.5.

\[
\sup^t(s) \leq \minScore(\text{top}_k(u_{t-1}, S))\beta^2 + \max_{q \in S} \{ \sigma(q, q_t) \}
\]

Intuitively, this is because for every prefix not in \( \text{top}_k(u_{t-1}, S) \), the similarity to \( u_{t-1} \) is smaller than \( \minScore(\text{top}_k(u_{t-1}, S)) \). Consequently, even if the newly added action \( q_t \) is matched to the most similar action in \( s \), by the definition of the similarity measure, the cumulative score cannot be greater than the prefix similarity (multiplied by the decay factor, following the arrival of a new action) plus the similarity of the best match (Proof omitted). We therefore use the following as our first upper-bound:

\[
B_1(s) = \minScore(\text{top}_k(u_{t-1}, S))\beta^2 + \max_{q \in S} \{ \sigma(q, q_t) \}
\]

Note that since \( \text{top}_k(u_{t-1}, S) \) is already computed, to calculate the bound we only need the similarity of \( q_t \) and the actions in \( s \). As individual actions do reside in a metric space (unlike sessions/prefixes), we show in Section 5.4 how this is done efficiently.

Second bound (\( B_2 \)). An alternative, less intuitive upper-bound for \( \sup^t(s) \) is achieved by dividing \( u_t \) into three disjoint subsequences w.r.t. a session \( s \) and separately bounding the similarity to each part: (a) the segment of \( u_t \) that was already compared to \( s \) in previous iterations, denoted \( u_t \); (b) the segment containing only the last added action \( q_t \), and (c) the segment in between these two. By adding up the bounds for each segment we obtain a bound for the whole sequence:

Proposition 5.6.

\[
\sup^t(s) \leq \max_{1 \leq j \leq |s|} \{ \text{sim}_{u_t, u_s}(j) \} \beta^{2(t-r)} + \frac{\beta^{2(t-r)} - \beta^2}{\beta^2 - 1} + \max_{q \in S} \{ \sigma(u_t[1], q) \}
\]

The latter proposition holds for \( \beta < 1^4 \).

Proof Sketch. We prove by induction. We denote \( s_t := \max_{1 \leq j \leq |s|} \{ \text{sim}_{u_t, u_s}(j) \} \). The base case is when \( t = \tau + 1 \).

We can easily show that:

\[
\sup^t+1(s) \leq s_t \beta^2 + \max_{q \in S} \{ \sigma(u_t[1], q) \}
\]

We know that:

\[
\sup^t+1(s) \leq \sup^t(s) \beta^2 + \max_{q \in S} \{ \sigma(u_t[1], q) \}
\]

(2)

Assuming the inequality holds for \( t \), we can bound \( \sup^t(s) \beta^2 \):

\[
\sup^t(s) \beta^2 \leq \beta^2(s_t \beta^{2(t-r)} + \frac{\beta^{2(t-r)} - \beta^2}{\beta^2 - 1} + \max_{q \in S} \{ \sigma(u_t[1], q) \})
\]

Since \( \max_{q \in S} \{ \sigma(u_t[1], q) \} \leq 1 \) we have:

\[
\sup^t(s) \beta^2 \leq \beta^2(1 - \tau) - \beta^2 + \beta^2(\beta^2 - 1)
\]

We can use the latter expression in (2) to obtain:

\[
\sup^t+1(s) \leq s_t \beta^2(1 - \tau) - \beta^2 + \beta^2(\beta^2 - 1)
\]

Hence the inequality holds for \( t + 1 \) \( \square \)

Importantly, note that this bound requires no action-similarity computations besides those already performed for \( B_1 \), and that all other values are either predefined constants or already computed in previous iterations. The proper upper bound \( \sup^t(s) \) is defined as the minimum of the two bounds: \( \sup^t(s) = \min(B_1(s), B_2(s)) \)

5.4 Efficient Retrieval of Candidate Sessions

As mentioned above, we use the first bound \( B_1 \) to quickly identify relevant sessions, then compute the full bound only for the retrieved sessions, for further pruning. We use the following observation, which follows immediately from Observation 5.5.

Observation 5.7. For \( \sup^t(s) \) to be not smaller than \( \inf^t \), \( s \) must contain some action \( q \) s.t.

\[
\sigma(q, q_t) \geq \inf^t - \minScore(\text{top}_k(u_{t-1}, S))\beta^2
\]

To identify sessions that contain such actions, we employ an index structure that uses the fact that the action similarity measure defines a metric space (See Section 3). Specifically, in our implementation we use a metric-tree \( [8] \) - a popular index structure that harnesses the triangle inequality property of a metric space to facilitate a fast similarity search.

Sessions selection via the actions metric-tree. Individual actions are stored in a metric tree, with pointers from each action to the session it appears in\(^5\). From Observation 5.7 and our definition of the action distance notion, it follows that all sessions that satisfy the bound \( B_1 \) must contain some action \( q \) s.t.:

\[
\Delta(q, q_t) \leq 1 - \inf^t + \minScore(\text{top}_k(u_{t-1}, S))\beta^2
\]

We thus use the metric tree for a fast retrieval of all such actions \( q \), and follow their associated pointers to identify the relevant sessions. Now we only need to compute Bound \( B_2 \) for the retrieved sessions to select those with upper bound greater than \( \inf^t \).

Example 5.8. For our running example, at \( t = 5 \) we retrieve candidate sessions via the metric tree, w.r.t. the lower bound 1.95 (computed as in Example 5.2). Recall that our goal is to retrieve sessions having an action \( q \) s.t. \( \Delta(q, q_t) \leq 1 - \inf^5 + \minScore(\text{top}_1(u_t, \{ \phi, \psi \}))\beta^2 \), namely actions similar to \( q_5 \) "c" with distance no more than 1 – 1.95 + 2.28 x 0.81 = 0.897. Only

\(^5\)The distance function used is the complement of our predefined action similarity function, i.e. \( \Delta(q, q') = 1 - \sigma(q, q') \)

103
the letters "C" and "c" meet this constraint, therefore sessions \( \psi \)
(that contains "c") is retrieved and added to the initial candidate
sessions set. Then, the upper bound for \( \psi \) is given by
\( sup_{\psi}(\psi) = \min(B_1(\psi), B_2(\psi)) = \min(2.84, 3.09) = 2.84 \) (full computation
omitted), which is greater than the lower bound 1.95, therefore
the final set of candidate sessions is \( \{\psi\} \).

### 5.5 Analysis of Pruning Effectiveness

Intuitively, the effectiveness of T-TopK is dependent on (1) the
amount of reduction in the examined candidate sessions, on which
the exact similarity scores are calculated, as well as on (2) the
cost of retrieving the candidate sessions.

First, let us assess the expected time complexity of T-TopK. Let
\( |C| \) be the expected number of candidate sessions meeting the
bounds (we assume \( |C| >> k \)), and let \( \tau \) be the expected number of
missing similarity vectors that need to be computed for each
candidate session. Also, denote by \( \alpha \) the cost of the metric-tree
search. Thus the average time complexity of T-TopK is given by
\( O(\alpha + \lambda |C|\tau) \).

The parameters \( \lambda \) (mean session size) and \( \lambda \) (cost of individ-
ual action similarity operation) are not controllable nor affected
by the implementation of the T-TopK algorithm. We therefore
analyze the performance of T-TopK according to the following
parameters: \( \alpha \) - the cost of the actions metric tree search, \( |C| \) -
the expected number of candidate sessions, and \( \tau \) the expected
number of missing vectors per candidate session.

#### Cost of searching the actions metric tree (\( \alpha \)).

Recall that the individual actions metric tree is used to efficiently retrieve all
repository sessions that satisfy Bound \( B_1 \). Based on the cost model
for metric trees proposed in [9], \( \alpha \) is given by:
\( \sum_{l=1}^{L} m_{l} t_{l} \) where:
\( F(x) \) is the action-distance probability distribution
(i.e., the probability that the distance of two arbitrary actions
\( \leq \chi \)), \( L \) is the number of levels in the tree, and
\( m_{l} \) is the number of nodes in level \( l \) of the tree and \( t_{l} \) is the mean covering radius
of nodes at level \( l \), and \( T_{l} \) is the search range (in our case, \( T_{l} =
1 - inf_{l} + minScore(topl(u_{l-1}, S))r_{l}^{2} \)). Also note that \( m_{l+1} \) is
the total number of individual actions stored in the metric tree.
Based on this cost model, we can see that (1) naturally, increasing
the size of the actions domain and the cost of action-distance
computation increases \( \alpha \). (2) More importantly, decreasing the
range \( r_{l} \) decreases \( \alpha \). The latter is greatly affected by the lower
bound \( inf_{l} \) and the upper bound \( B_1 \), as described next.

#### Expected number of candidate sessions (\( |C| \)).

The number of candidate sessions directly stems from the number of ses-
sions that satisfy both bounds \( B_1 \) and \( B_2 \). Intuitively, \( |C| \) de-
pend on three factors: (1) \( \frac{\text{How high is the lower-bound}}{\text{Natu-}} \)

#### 6 EXPERIMENTAL STUDY

As mentioned above, since the speedup achieved by our opti-
mizations may be affected by the underlying structure of the
session repositories, one would ideally like to evaluate their per-
formance on a multitude of real-life session repositories with
different characteristics.

However, the only publicly available repository (to our knowl-
edge) of real-life analysis sessions is rather small [1]. Therefore,
we first performed experimental evaluations on a variety of artifi-
cially crafted session repositories, each having different internal
characteristics that may effect the performance of our solutions.

Then, we used the session repository of [1] to verify that the
same performance trends hold on real life sessions as well.

Last, let us recall that our experiments focus on running-
time performance. For an evaluation of the quality of recom-
endations generated based on \( SW-SIM \) we refer the reader to,
e.g., [2, 3].
In what comes next, we describe the experimental setup and the construction method for the artificial repositories in Section 6.1, then show the performance of our solution compared to numerous baseline approaches in Section 6.2. Last, we examine the scalability and performance trends w.r.t. numerous parameters of the search problem and session repositories (Section 6.3).

6.1 Experimental Datasets & Setup

Artificial Session Repositories Construction. In the absence of real-life session repositories that are large enough and publicly available, we constructed a multitude of artificial repositories, each with different underlying characteristics that may affect the performance of our solution. Each repository was constructed by first building an action (metric-) space with certain, configurable properties, then constructing repository sessions by drawing actions from this space (also w.r.t. a configurable setting). The controllable parameters and the ranges we use are depicted in Table 1.

Generating the individual action space. Recall that our algorithms model analysis actions as abstract objects in a given metric space. We represent here actions as points in an $N$-dimensional Euclidean space, with the Euclidean distance as the distance metric. To obtain values in $[0, 1]$ range, each element is restricted to the range $[0, \frac{1}{\sqrt{N}}]$. We the simulate a controlled degree of similarity between actions as follows: first draw a (controllable) number of action points uniformly at random (u.a.r.), to serve as cluster centers, then generate additional actions around each center using a (multi-variant) normal distribution, with the cluster-center as mean. In our experiments we varied the number $N$ of action-space dimensions, the number of action clusters, and their radius (standard deviation), obtaining different degrees of underlying action-similarity in the repository session.

Generating repository sessions. The sessions repository is constructed in an analogous manner. We first generate a set of “seed” sessions, then use them to generate sessions with varying degrees of similarity to the seed. For each session (seed or other) we draw its length from a given normal distribution. To construct a seed-session we select a sequence of actions of the given length. The rest of the repository sessions are constructed based on one of the seed-sessions. To further control the similarity of an arbitrary session $s$ to its corresponding seed, we set a fraction $p$ of “random” actions in $s$. Namely, $p \cdot l$ actions in $s$ are randomly drawn (u.a.r) from the entire action space. The rest $(1 - p) \cdot l$ actions in $s$ are chosen from the same action-clusters as the actions in the corresponding seed session. Intuitively, setting a high value of $p$ of “random” actions significantly decreases the potential of two sessions to be similar.

In the experiments below we used multiple repository configurations, as depicted in Table 1.

REACT-IDA Session Repository. We used the only publicly available repository (to our knowledge) of real-life sessions [1], collected as part of the experimental evaluation of an existing IDA recommender system [26]. The user sessions were performed by 56 analysts who used a web-based IDA interface in order to explore four different datasets from the cyber-security domain. The repository contains 1100 distinct actions. The average session length is 8.5 actions, and the median user idle-time is 40 seconds.

Action Similarity. Recall that both algorithms require a measure for action similarity. When using the real-life repository, we employed the similarity measure of the IDA recommender system [26] whose analysis UI was used to record the sessions. For the artificial repositories, where the actions reside in a multi-dimensional metric-space, we used Euclidean distance.

Default Parameters Selection. The search problem parameters (namely, the decay factor $\beta$, the gap penalty $\delta$, and the size of $k$) may affect the performance of our solution. However, to capture the real-life setting where the algorithms are embedded in actual IDA recommender systems, the default values are set to optimize the quality of the SW-SIM measure, rather than the performance of our optimizations. To do so, we embedded SW-SIM as the top-k search component in the IDA next-step recommender system [26] (code available in [1]), and performed the hyper-parameters selection routine as described in [26]. Briefly, this is done by executing a grid search for the systems’ parameters (including the top-k problem parameters), and selecting the configuration that allows the recommender system to achieve the highest qualitative performance (as determined in a predictive accuracy evaluation).

As for the artificial repository construction, intermediate values were chosen as default values, as stated in Table 1. Nevertheless, in Section 6.3 we examine the effect of varying each problem/repository parameter on the running time performance of our solution.

Hardware and Software Specifications. We implemented the algorithms presented in the previous sections in Java 8, using Guava (https://github.com/google/guava) for the max-heap. For the metric-tree we used an M-tree [8] Java implementation (https://github.com/erdavila/M-Tree), employing Minimum Sum of Radii split policy (See [8]), and a maximum node capacity of 20 objects. All experiments were conducted over Intel Core i7-4790, 3.6GHz machine (4 dual cores), equipped with 8GB RAM.

6.2 Baseline Comparison

We compared the performance of the T-TopK algorithm to the following baseline algorithms, each computing the exact set $top_k(u_t, S)$ according to the alignment based similarity measure (Definition 3.1). Among these, we show the results of the following baselines: (1) Naive Sequential Search (NSS) that retrieves the set of similar prefixes by iteratively comparing $u_t$ to each prefix of each repository session in $S$ using a direct implementation of Definition 3.1 with no further optimizations, then selects

---

Table 1: Problem & Repository Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decay Factor $\beta$</td>
<td>0.1</td>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>Gap Penalty $\delta$</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>Output Size $k$</td>
<td>4</td>
<td>24</td>
<td>12</td>
</tr>
</tbody>
</table>

| Action Dim.                   | 5   | 500 | 25     |
| #Action Clusters              | 3K  | 24K | 6K      |
| Cluster Rad. (std)            | 0.0075 | 0.012 | 0.003 |
| #Seed Sessions                | 6%  | 16% | 10%     |
| %Random Actions (p)           | 0%  | 100%| 80%     |
| Idle Time (s)                 | 0   | 5   | 1       |

<table>
<thead>
<tr>
<th>Repository Scale Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#Sessions</td>
<td>Session len.</td>
</tr>
<tr>
<td></td>
<td>$N(4,3^2)$</td>
</tr>
<tr>
<td></td>
<td>$N(32, 12^2)$</td>
</tr>
<tr>
<td></td>
<td>$N(16, 3^2)$</td>
</tr>
</tbody>
</table>

---

The similarity notion considers both the action syntax and (a signature of) the results. (See [26]).
the top-k similar prefixes. (2) Optimized Sequential Search (OSS), which employs the first optimization described in Section 4, instead of computing the alignment matrix for each prefix. Namely, it iterates over all sessions in \( s \in S \), constructs a single alignment matrix \( A_{u_t,s} \), and uses Observation 4.1 to derive the similarity scores of \( u_t \) and each prefix of \( s \). (3) I-TopK, the iterative algorithm depicted in Section 5.1 which employs both of the optimizations in Section 4. (4) T-TopK, the optimized algorithm described in Sections 5.2-5.4. (5) Constant Shift Embedding (CSE)[31] and (6) NM-Tree [34], are general purpose solutions for top-k search in a non-metric space. Both use metrization techniques: CSE use a simple solution that increments each distance score by a predefined constant, so the triangle inequality is enforced. Then, all session prefixes in \( S \) are stored in a metric tree (w.r.t. the new metric space). When given the ongoing session \( u_t \), the metric tree is traversed, using the new metric to obtain the exact set of top-k similar prefixes. NM-tree uses a more sophisticated similarity-preserving transformation on the original (non-metric) distance measure, then employs an extension of the metric tree to index and query the transformed metric space.

Finally, recall from Section 2 that optimization techniques dedicated to other similarity measures (e.g., DTW, Global Sequence Alignment) are inadequate for \( SW-SIM \), therefore could not serve as baselines. Also, approximation-based solutions were omitted as well since this work concerns with the case of exact computation of the similarity scores.

**Evaluation Process.** We evaluated all baselines using a multitude of configurations for the top-k search problem (i.e. the constants \( \beta, \delta \) of the similarity measure, and \( k \)), on various configurations of artificial repositories. The evaluation process is as follows. Given a session repository \( S \), in each trial we draw a random session prefix as \( u_t \), then employ each baseline to retrieve \( \text{top}_k(u_t, S) \). We performed 100 trials for each repository, capturing the average execution time and memory consumption, and the average number of action similarity operations performed in each run.

**Results.** Table 2 presents a representative sample of the results, showing the average running time and the number of action similarity operations (denoted \#ops) obtained by each baseline. The comparison is presented for the REACT-IDA repository (Containing 1.1K actions) as well as Repo-10 and Repo-100, which are two artificial session repositories (with the default configuration stated in Table 1) containing 10K and 100K sessions (resp., 160K and 1.6M actions). First, for both Repo-10 and Repo-100, the performance of NM-Tree and CSE is almost on-par with NSS (the naive sequential search). This could be due to the fact that transforming the non-metric space may induce high overlap between the metric-tree nodes, therefore the search deteriorates to sequential search plus additional overhead induced by the metric tree (See [34]). Second, note that OSS improves running times by 5X (and \#ops by 9X) over NSS due to its efficient use of Observation 4.1. However, a more significant improvement of 15X in running times (and 9X in terms of \#ops) is achieved by I-TopK which utilizes the incremental computation. Finally, an additional 2-4X speedup is obtained by T-TopK, resulting in an overall improvement of 189X to 291X over the NSS baseline.

The performance trends on the REACT-IDA repository (comprising the smaller collection of real-life sessions) are similar, with some minor variation. We can first see that NM-Tree performs slightly better than NSS and CSE. This is due to the underlying structure of the dataset that allows the NM-Tree to perform (and store) the distance-preserving transformations more efficiently. However, both our optimized algorithms perform significantly better, with T-TopK dominating. Interestingly, while the REACT-IDA is fairly small, the pruning-based optimizations of T-TopK are still highly effective. T-TopK retains a 2X speedup compared to I-TopK, and significantly outperforms the other baselines.

Next, we present a performance comparison of the algorithms when employed on a multitude of different repositories and with different problem parameters, as well as an examination of the computation segments in T-TopK. Since in all configurations, the performance of T-TopK was significantly better (by at least an order of magnitude) than the baseline algorithms, we omit them from presentation and use only I-TopK as a baseline.

### 6.3 Scalability & Parameters Effect

We next analyze the performance of the T-TopK Algorithm compared to I-TopK, by varying the problem parameters as well as the repository parameters, one at a time, while keeping the rest at their default values (As in Table 1). We measured both the running times, number of action similarity operations (\#ops), and memory consumption.

**Scalability Parameters.** Figure 3a depicts the performance of the algorithms when increasing the number of sessions in the repository. For both algorithms we can see a rather linear increase in running times (correspondingly, \#ops however the two plots overlap), but T-TopK shows a significant speed up of 3X (on average) over I-TopK. Moreover, as we can see in Figure 3b, when the mean session length increases, T-TopK performance stays stable compared to a linear increase for I-TopK. This demonstrates that...

<table>
<thead>
<tr>
<th>Baseline</th>
<th>REACT-IDA (1.1K actions)</th>
<th>Repo-10 (160K actions)</th>
<th>Repo-100 (1.6M actions)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (ms)</td>
<td>#ops</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>NM-Tree</td>
<td>42.3</td>
<td>27.5K</td>
<td>18,503</td>
</tr>
<tr>
<td>CSE</td>
<td>55.0</td>
<td>36.6K</td>
<td>18,224</td>
</tr>
<tr>
<td>OSS</td>
<td>51.9</td>
<td>35K</td>
<td>17,204</td>
</tr>
<tr>
<td>I-TopK</td>
<td>7.6</td>
<td>5.1K</td>
<td>3,242</td>
</tr>
<tr>
<td>T-TopK</td>
<td>0.5</td>
<td>641</td>
<td>59</td>
</tr>
</tbody>
</table>

Table 2: Baseline Comparison. We compared various baseline algorithms in terms of running times and the number of similarity operations.

![Figure 3: Effect of Scale Parameters](image-url)

(a) Repository Size  
(b) Mean Session Len.
the effectiveness of the threshold-based approach further grows for longer sessions. As for memory consumption, the maximal usage for T-TopK did not exceed 74MB even for a repository as big as 100K sessions (1.6M individual actions), which is negligible in practice. We therefore do not further discuss memory consumption in the next experiments.

**Problem Parameters.** Recall that the problem parameters are the gap penalty $\delta$, the decay factor $\beta$ and the output set size $k$. We examine their effect on the performance, when varying each parameter, and keeping the rest at their default values. To gain a better insight on the computation of T-TopK, we break the total #ops performed into three computational segments: (1) forming the initial similarity lower bound, (2) the metric-tree search, and (3) computing similarity vectors for candidate sessions. The corresponding running time trends are similar, therefore omitted from the figures below. Also, as Segment (3) is negligibly small when using the default idle-time, we further restrict the default user idle-time to 150ms, which naturally stresses T-TopK. The performance of I-TopK in terms of #ops is represented by a dotted flat line since it is not dependent on the problem parameters.

We observe that increasing each of the problem parameters results in a minor increase in #ops for T-TopK. Figure 4a and Figure 4c show the effect of the decay factor $\beta$ and the output size $k$ on performance, when the algorithms are employed on the REACT-IDA repository, and Figure 4d shows the effect of the gap penalty $\delta$ on the default artificial repository (the effect of $\delta$ on the REACT-IDA repository was marginal, therefore the figure is omitted). The increase in performance is expected, since the values of the problem parameters play a part in the lower/upper bound computations: (1) The decay factor $\beta$ affects the upper bound, therefore lower values induce more restrictive candidate selection, and thereby better performance. (2) Increasing the output size $k$ causes a decrease in the lower bound threshold $inf$, thus more candidate sessions are examined.

To gain a deeper insight on the performance w.r.t. the computation segments, we examine in Figure 4b, the number of sessions retrieved in the metric-tree search results (i.e. sessions satisfying bound $B_1$) and the number of sessions among them that also meet bound $B_2$ (hence satisfy the upper bound), for varying $\beta$ values. We can see that the first set grows with $\beta$ (hence the cost of Segment 2 increases), but many sessions are pruned via bound $B_2$ (thus the cost of Segment 3 does not increase).

**Repository Parameters.** As is the case for most top-k search optimizations and dedicated data structures, the underlying structure of the data points may impact the performance of the T-TopK algorithm (note that the incremental similarity optimizations presented in Section 4 are not affected by such parameters).

Therefore, to properly evaluate the effectiveness of T-TopK we constructed a multitude of artificial repositories (as described above), each with a different underlying structure that stems from a particular repository-parameters configuration. We varied each construction parameter and examined its effect on the performance of T-TopK (keeping the rest of the parameters at their default configuration). Figures 4e, 4f and 4g depict the performance of T-TopK compared to I-TopK when varying the amount of individual action clusters, percentage of “seed sessions”, and the percentage of random actions in a session, respectively. Intuitively, increasing the values of each of these parameters inflict more “randomness” on the underlying structure of the data points (sessions), therefore the mean similarity score of two arbitrary sessions decreases. In turn, lower similarity scores imply smaller lower bounds, therefore more candidate sessions are examined. With the exception of one case, in all observed settings, even when increasing these parameters, T-TopK was still effective, outperforming I-TopK. The only setting where I-TopK outperformed T-TopK was when setting a high (> 60%) percentage of random actions in the repository sessions, which significantly decreases the average session similarity score (importantly, further increasing this parameter had a negligible effect on performance). However, as we show next, given slightly higher user idle-times, (which is a very reasonable assumption) T-TopK still outperforms I-TopK, even when the percentage of random actions is high, as the computation segment of calculating similarity vectors (in green) is significantly diminished.

**User Idle-Times.** We measured how the mean user idle-time affects the performance of T-TopK. Such idle time between consecutive actions of the same user often takes several seconds. Figure 4h depicts the performance when varying the expected time ranges from 0 to 0.24 seconds. Interestingly, even the latter, which is lower than the minimal human reaction time to a visual stimulus, was already sufficient to compute all missing vectors offline. As expected, T-TopK improves when the idle-time increases, and when all missing vectors are computed offline - T-TopK obtains almost a 3X speedup over I-TopK.
7 CONCLUSION

In this work, we show (for the first time, to our knowledge) that by utilizing the progressive nature of IDA sessions, a major running-time speedup (of over 380X) can be obtained, compared to current solutions for top-k similarity search of analysis sessions. Our solution allows IDA recommendation systems to effectively rely on much larger session repositories while retaining interactive response times.

However, the scope of our work is limited in two main aspects: First, our solution is dedicated to the SW-SIM similarity measure for analysis sessions. While considered a very comprehensive, useful measure, some IDA recommendation systems use different session similarity measures. Nevertheless, we believe that the core principles of our solution may be applicable (with some tweaking and adaptations) to other similarity measures - even outside the scope of IDA. (e.g. scientific/business workloads, prescriptive analytics, mashups [5, 13, 42]).

Second, as common in many filter-and-refine based frameworks, the efficiency of the pruning techniques used by the T-TOPk algorithm may be affected by underlying characteristics of the session repository, as well as the parameters of the search problem. In the absence of large enough, publicly available real-life IDA workloads, our experimental evaluation is performed, primarily, over a multitude of carefully crafted artificial repositories, each with different characteristics. Although we have shown that similar performance trends occur on the real-life session repository as well (the only one that is publicly available), it is still left to demonstrate the performance speedup on other real-life IDA workloads (when such become publicly available). Nevertheless, recall that the efficiency of the single-alignment optimizations, and the computation of the similarity vectors (as presented in Section 4), is independent of the session repository or problem parameters - and even these alone still provide a significant speedup of $40 - 80X$ compared to the currently available solutions.

Acknowledgements. This work has been partially funded by the Israel Innovation Authority (MDM), the Israel Science Foundation, the Binational US-Israel Science foundation, Len Blavatnik and the Blavatnik Family foundation, and Intel® Al DevCloud.

REFERENCES


[34] Tomáš Skopal and Jakub Líkol. 2008. NNS-tree: Flexible approximate similarity search in metric and non-metric spaces. In DEXA.


[40] Sebastian Wandelt, Johannes Starlinger, Marc Bux, and Ulf Leser. 2013. RCSF: Scalable similarity search in thousand (s) of genomes. VLDB (2013).


