SeDA: Bridging the Gap between Efficient Syntactic and Precise Semantic Search of Similar Passages in Large Text Corpora

Pranay Mundra*
pranay.mundra@yale.edu
Yale University

Daniel Kocher
dkocher@cs.sbg.ac.at
University of Salzburg

Martin Schäler
martin.schaeler@sbg.ac.at
University of Salzburg

Nikolaus Augsten
nikolaus.augsten@plus.ac.at
University of Salzburg

ABSTRACT

A two-stage pipeline is commonly used to identify similar text passages in large document corpora: First, a fast approach produces potential matches, which are then examined in detail. Existing approaches for the first step consider only syntactic information and miss semantically similar passages that are syntactically dissimilar. To address this, we define the novel problem of semantic document alignment as a semantic set-similarity problem on k-width windows. For two documents S and T, an exhaustive baseline that evaluates all $|S|\times|T|$ window pairs is computationally infeasible since assessing the similarity of a single pair requires $O(k^3)$ time.

We propose SeDA, which combines a sophisticated candidate generation technique with a bound cascade to drastically reduce the number of expensive window comparisons. It further exploits overlapping windows to efficiently compute both the bounds and the final similarity scores. Our empirical results on three large document corpora indicate that SeDA prunes over 99% of the window similarity computations, resulting in response-time improvements of 1.5–3 orders of magnitude over the baseline solution and 2–5 orders of magnitude over SBERT. Compared to purely syntactic competitors, SeDA provides competitive runtimes and achieves superior result quality, i.e., near-optimal F1-Score of precision/recall and matching the performance of purely semantic methods such as SBERT.

PVLDB Reference Format:
Pranay Mundra, Daniel Kocher, Martin Schäler, and Nikolaus Augsten. SeDA: Bridging the Gap between Efficient Syntactic and Precise Semantic Search of Similar Passages in Large Text Corpora. PVLDB, 19(6): 1332 - 1344, 2026.
doi:10.14778/3797919.3797938

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mschaeler/seda.

1 INTRODUCTION

In the current era of increasing global information exchange, precise and scalable semantic document search is crucial for locating analogous document passages. Existing methods are commonly categorized into syntactic and semantic methods. Semantic methods leverage contextual embeddings or deep language models to capture meaning and yield precise matches. However, this expressiveness incurs substantial computational overhead, making them difficult to scale. Contrastingly, syntactic methods are computationally efficient but fail to detect semantically similar documents that differ syntactically. To balance efficiency and semantic expressiveness, most systems adopt a two-stage pipeline: initially, a document query is matched using a fast syntactic technique to identify a set of potentially similar documents, which are then rigorously evaluated with time-intensive semantic methods [15, 17]. Since the syntactic matching stage completely ignores semantic relationships, it systematically filters out semantically similar documents that are syntactically different. Considering semantics is vital in text analysis, as documents often contain subtle nuances, deep semantic ties, translations (e.g., into contemporary youth language), and paraphrases. Example 1.1 demonstrates this challenge in two biblical texts.

Example 1.1 (Bible Verse Alignment). Consider two passages from different Bible revisions, Book of Ester 1:2, that span numerous historical eras: S, T stem from the English Standard Version and the King James Bible, respectively. S', T' show the sequences after common preprocessing steps (stop-word removal; stemming). For a human reader, the two sequences are similar, as one would expect for the same Bible verse. However, standard alignment faces two major challenges: (1) The word order may differ between text passages, e.g., "throne". (2) Syntactically different words may be synonyms, e.g., syntactic matching of "citadel" and "palace" is virtually impossible.

S = "*in those days when King Ahasuerus sat on his royal throne in Susa, the citadel,*"
T = "*That in those days, when the king Ahasuerus sat on the throne of his kingdom which was in Shushan the palace,*"

T' = "*day king ahasuerus sat royal throne susa citadel*"
S' = "*day king ahasuerus sat throne kingdom shushan palace*"

To handle positional variations of words, syntactic approaches [8, 34, 35] employ windowing techniques that disregard word order within each window. As such methods lack semantic awareness, they rely on very low similarity thresholds θ to avoid missing potential matches. This inevitably leads to low-quality matches, sacrificing precision for recall. As a result, the two-stage strategy of syntactic preprocessing followed by semantic verification struggles to scale for two main reasons. First, lower precision produces a large number of irrelevant document pairs that must still undergo costly semantic

*Work done during an internship at the University of Salzburg.

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797938

verification. Second, exact syntactic matching becomes inefficient at low similarity thresholds. For example, systems like TxtAlign [37], achieve their best matching quality with Jaccard-based token similarity at thresholds as low as $0.3 \leq \theta \leq 0.4$. In contrast, works on efficient set similarity search focus on Jaccard thresholds of $\theta \geq 0.5$ [20, 39] as common filtering techniques [2, 9, 29, 36, 38] degrade significantly for lower thresholds. Hence, solutions targeting at low similarities often resort to approximations [23, 37], further reducing result quality.

Semantic Document Alignment. To this end, we introduce and formally define the novel problem of *semantic document alignment*, which aims to identify pairs of similar text passages of user-defined length k (referred to as *windows*) between two documents. Unlike syntactic approaches that essentially count identical words within windows, our method is based on the concept of *semantic similarity of sets*. This enables us to effectively handle both positional variations in word order and paraphrased text passages.

To assess the semantic similarity between two windows, a maximum matching between the semantic word similarities in the windows is established, which requires $O(k^3)$ time. An exact exhaustive baseline for solving the semantic document alignment problem between two documents S and T must compute this expensive matching for $|S| \times |T|$ window pairs, which is impractical for large documents.

The SeDA Algorithm. We propose the SeDA algorithm to scale semantic document alignment to large documents. Scaling a semantic solution is more challenging than scaling syntactic, set-based approaches, which leverage exact word matchings and disregard semantic similarities of words. SeDA (a) efficiently computes *exactly the same* alignment matrix as the exhaustive solution, while (b) substantially improving the result quality w.r.t. precision and recall.

Conceptually, SeDA computes an alignment matrix A that stores similarity scores for all pairs of k-length windows from sequences S and T. Each window pair corresponds to a cell in A, and we are interested only in cells whose similarity exceeds a user-defined threshold θ. To avoid processing the quadratic number of matrix cells, SeDA employs sophisticated pruning techniques and optimizes the matching computation. Its key features are as follows:

(1) A *candidate generation technique* avoids iterating over all cells of A. Empirical results show that fewer than one-sixth of the cells of A are proposed as candidates for similarity $\theta \geq 0.7$.
(2) A *cascade of bounds* groups adjacent candidate cells into *candidate runs* to leverage overlapping windows, enabling incremental computation of the tightest $O(k^2)$ bound pruning 99% of the cells in linear $O(k)$ time.
(3) To *jump-start* the similarity computation of the remaining cells, we efficiently reuse results computed for the bound cascade, which avoids a quadratic $O(k^2)$ overhead per cell pair and substantially improves practical performance.

Contributions. Summarizing, our contributions are as follows.

- We introduce the *semantic document alignment* problem to identify pairs of semantically similar text passages in documents. Unlike syntactic approaches, our method effectively captures semantic similarity, including paraphrased text.
- Our *SeDA algorithm* efficiently solves the semantic document alignment problem. SeDA leverages our pruning techniques to avoid most of the quadratic number of expensive similarity computations required by the baseline solution.
- *High-quality Matches.* Experiments across three different use cases demonstrate result quality competitive with the best existing approaches. On a well-established plagiarism detection benchmark, we achieve an F1-Score of 0.93, clearly outperforming the best syntactic alignment (F1 = 0.83) [23].
- *Efficiency.* In our extensive empirical evaluation on three corpora with varying avg. text lengths, SeDA consistently outperforms the baseline solution by 1.5–3 orders of magnitude and achieves similar or superior runtimes compared to a purely syntactic state-of-the-art method.

Paper Outline. Section 2 introduces semantic document alignment with an exhaustive solution in Section 3, and pruning techniques in Section 4. Section 5 presents SeDA, including candidate generation, bound cascade, and Hungarian algorithm optimization. After empirically evaluating SeDA against the state of the art in Section 6, we discuss related work in Section 7 and conclude in Section 8.

2 PROBLEM DEFINITION

In this section, we formally define the problem and formulate our objective. Next, we model the problem as dynamic bipartite graph matching and illustrate it with a running example.

2.1 Formal Problem Definition

Definition 2.1 (k-width Window). Given a sequence $S = \{s_1,...,s_m\}$ of m tokens, a *k-width window* $W_{S_i} \subseteq S$ is a *multiset* of k consecutive tokens in S, $W_{S_i} = \{s_i, s_{i+1},...,s_{i+k-1}\}$, $1 \leq i \leq m-k+1$.

Let $W_S = \{W_{S_1}...,W_{S_{m-k+1}}\}$ and $W_T = \{W_{T_1},...,W_{T_{n-k+1}}\}$ denote the set of all k-width windows of S and T, respectively. Note that a sequence of m tokens has $m-k+1$ (partially overlapping) windows.

Definition 2.2 (Semantic Similarity of k-width Windows). Given two k-width windows $W_{S_i} \in W_S, W_{T_j} \in W_T$ and a symmetric similarity function $sim(s_p, t_r)$, $s_p \in W_{S_i}$, $t_r \in W_{T_j}$, which returns 1 if tokens s_p and t_r are identical, and a value in $[0,1]$ otherwise. Let $\mathcal{M}: W_{S_i} \rightarrow W_{T_j}$ be a one-to-one optional matching that maps each $s_p \in W_{S_i}$ to $\mathcal{M}(s_p) \in W_{T_j}$. Then, the semantic similarity of W_{S_i} and W_{T_j} is:

$$\mathcal{WO}\left(W_{S_i}, W_{T_j}\right) = \max_{\mathcal{M}} \sum_{s_p \in W_{S_i}} sim(s_p, \mathcal{M}(s_p))$$

We use pre-trained general-purpose word embeddings in the remainder, but any symmetric function returning 1 if two tokens are equal and a value $\in [0,1]$ is possible. We emphasize that the semantic token similarity function falls back to syntactic similarity if no semantic is found, e.g., for out-of-vocabulary words. Moreover, we rely on overlapping $k-$width windows, to be able to detect minor changes within sentences as they frequently occur in one use cases in the evaluation [4]. Technically, we could also rely on sentences or paragraphs and the sequences do not need to have the same length. This holds for all presented solutions and optimization.

Based on the semantic similarity of two passages with k tokens, we define the semantic document alignment as finding *all pairs of semantically similar passages* between two documents S, T. The result is an alignment matrix A that can be postprocessed to align the two documents, e.g., with the Smith-Waterman algorithm [32].

Definition 2.3 (Semantic Document Alignment Problem). Given a similarity threshold $\theta > 0$, the alignment matrix A of two sequences S and T stores the pairwise *normalized* semantic similarity between any pair of k-width windows $(W_{S_i}, W_{T_j}) \in W_S \times W_T$ if the similarity is at least θ, or zero otherwise.

$$A[i,j] = \begin{cases} \frac{\mathcal{WO}(W_{S_i}, W_{T_j})}{k}, & \text{if } \frac{\mathcal{WO}(W_{S_i}, W_{T_j})}{k} \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

The *semantic document alignment problem* is to compute all pairs (i,j) such that $A[i,j] > 0$.

Algorithmic Objective. We aim at a novel algorithm that efficiently solves the semantic document alignment problem, i.e., it produces exactly the same alignment matrix as the exhaustive solution.

2.2 Semantic Similarity with Bipartite Graphs

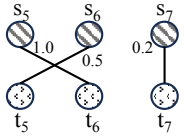
We model the semantic document alignment of two sequences S and T as a matching problem on subgraphs of a bipartite graph. Let $G = (V, E)$ be a weighted bipartite graph with $V = S \cup T$, $S \cap T = \emptyset$, and $E = S \times T$, where S and T form the two parts of the bipartition, and the weight of an edge $(s_p, t_r) \in E$ is $sim(s_p, t_r) \in [0,1]$. Each pair of k-width windows $W_{S_i} \subseteq S$ and $W_{T_j} \subseteq T$ induces a subgraph $G_{ij} = (V_{ij}, E_{ij})$, where $V_{ij} = W_{S_i} \cup W_{T_j} \subseteq V$ and $E_{ij} = W_{S_i} \times W_{T_j} \subseteq E$. Figure 1 illustrates the relation between two windows and the induced subgraph.

We define *windowed bipartite graph matching* as the problem of computing a maximum weight matching for every subgraph G_{ij} of G. Semantic document alignment is a special case that finds subgraphs whose k-normalized matching weight exceeds a threshold $\theta > 0$.

Maximum-weight bipartite graph matching, which generalizes the classical assignment problem, can be computed in $O(k^3)$ time for each subgraph G_{ij} induced by two k-width windows [11]. Neighboring subgraphs G_{ij} and $G_{i+1,j}$ differ only by a single vertex and its incident edges: s_i is removed and s_{i+k} is inserted. This locality suggests that one might update the matching incrementally. However, a naive update that simply assigns the new vertex s_{i+k} to its highest-weight incident edge can violate the one-to-one matching constraint. Restoring the constraint may require reassigning other vertices, causing non-trivial changes to the matching. Although neighboring subgraphs often share much of their matching in practice, we are not aware of a general incremental algorithm that is guaranteed to maintain correctness, resulting in an overall runtime of $O(|S| \cdot |T| \cdot k^3)$ for the windowed bipartite graph matching problem.

$sim(s_i, t_j)$	t_5 royal	t_6 throne	t_7 susa
throne (s_5)	0.6	1.0	0.0
kingdom (s_6)	0.5	0.5	0.1
shushan (s_7)	0.4	0.1	0.2

(a) Similarity matrix SM; colored cells are matched.



(b) Weighted bipartite graph; 1-to-1 matching.

Figure 1: Similarity matrix for window pair (W_{S_5}, W_{T_5}) and maximum weight matching in induced subgraph $G_{5,5}$.

2.3 Illustrative Example

Consider again the Bible sequences S' and T' from Example 1.1. Further, let $k = 3$ and $\theta = 0.5$. Then, S' and T' have the following k-width window sets (cf. Table 1): $W'_S = \{W_{S'_1}, ..., W_{S'_6}\}$ and $W'_T = \{W_{T'_1}, ..., W_{T'_6}\}$.

Table 1: 3-width windows for S' and T'.

i, j	$W_{S'_i}$	$W_{T'_j}$
1	{day, king, ahasuerus}	{day, king, ahasuerus}
2	{king, ahasuerus, sat}	{king, ahasuerus, sat}
3	{ahasuerus, sat, throne}	{ahasuerus, sat, royal}
4	{sat, throne, kingdom}	{sat, royal, throne}
5	{throne, kingdom, shushan}	{royal, throne, susa}
6	{kingdom, shushan, palace}	{throne, susa, citadel}

Previous approaches use Jaccard or Hamming similarity [20, 26, 39] to compute the similarity between sets, i.e., $\mathcal{WO}(W_{S_i}, W_{T_j})$. These similarity measures solely rely on syntactic features such as stemmed words or q-grams. We depict the alignment matrix for S' and T' under the Jaccard similarity in Figure 2a: The first two windows, W_{S_1}/W_{T_1} and W_{S_2}/W_{T_2}, are perfect matches because the token sets are identical. For the next two windows, W_{S_3}/W_{T_3} and W_{S_4}/W_{T_4}, the similarity is only 0.5, and for windows W_{S_5}/W_{T_5} and W_{S_6}/W_{T_6} the cell similarity does not exceed θ. In contrast, Figure 2b shows the corresponding alignment matrix using semantic similarity based on pre-trained word embeddings. We observe that $\mathcal{WO}(W_{S_1}, W_{T_1})$ and $\mathcal{WO}(W_{S_2}, W_{T_2})$ equal the respective Jaccard similarity because the sets are identical. For other window pairs, we observe a difference: For instance, the semantic similarity for the pairs W_{S_5}/W_{T_5} (cf. Figure 1) covers the similarity of 'kingdom' and 'royal' such that $\mathcal{WO} = 0.6$, as opposed to 0 when using Jaccard similarity.

A	$W_{T'_1}$	$W_{T'_2}$	$W_{T'_3}$	$W_{T'_4}$	$W_{T'_5}$	$W_{T'_6}$
$W_{S'_1}$	1.0	0.5	0.0	0.0	0.0	0.0
$W_{S'_2}$	0.5	1.0	0.5	0.0	0.0	0.0
$W_{S'_3}$	0.0	0.5	0.5	0.5	0.0	0.0
$W_{S'_4}$	0.0	0.0	0.0	0.5	0.0	0.0
$W_{S'_5}$	0.0	0.0	0.0	0.0	0.0	0.0
$W_{S'_6}$	0.0	0.0	0.0	0.0	0.0	0.0

(a) Jaccard similarity.

A	$W_{T'_1}$	$W_{T'_2}$	$W_{T'_3}$	$W_{T'_4}$	$W_{T'_5}$	$W_{T'_6}$
$W_{S'_1}$	1.0	0.7	0.6	0.0	0.0	0.0
$W_{S'_2}$	0.7	1.0	0.9	0.5	0.0	0.0
$W_{S'_3}$	0.6	0.9	0.8	0.7	0.0	0.0
$W_{S'_4}$	0.0	0.5	0.5	0.8	0.5	0.0
$W_{S'_5}$	0.0	0.0	0.0	0.6	0.6	0.5
$W_{S'_6}$	0.0	0.0	0.0	0.0	0.0	0.0

(b) Semantic similarity.

Figure 2: Alignment matrices of two documents S', T'; $\theta = 0.5$.

3 AN EXHAUSTIVE SOLUTION

This section introduces the Hungarian algorithm [22], which is used to solve the assignment problem. Then, we show a first exhaustive baseline solution that solves the semantic document alignment problem (cf. Definition 2.3) in $O(|S| \cdot |T| \cdot k^3)$ time.

3.1 The Hungarian Algorithm

The Hungarian algorithm [22] is the most efficient exact method for solving the assignment problem in bipartite graphs. Its runtime is

$O(k^3)$, where k is the number of vertices in each partition. The input is a *similarity matrix* SM[1] of size $|W_{S_i}| \cdot |W_{T_j}|$, i.e., $k \times k$. Each element $SM[i,j]$ indicates the similarity of two tokens $p \in W_{S_i}$ and $r \in W_{T_j}$. Importantly, the algorithm uses a heuristic to greedily match tokens, and thus aims at reducing the cubic complexity. This heuristic *updates* the similarity matrix SM in $3 \cdot k^2$ steps, i.e., $O(k^2)$ time.

3.2 Exhaustive Baseline

An exhaustive solution computes each cell of the alignment matrix A by employing the Hungarian algorithm on the corresponding bipartite graph. Given the $O(k^3)$ runtime of the Hungarian algorithm and the maximum size $|S| \cdot |T|$ of A, the total computational complexity is $O(|S| \cdot |T| \cdot k^3)$. To reduce the computational overhead for recurring similarity computations, we cache the token similarities such that each value $sim(s_i, t_j)$ is computed only once.

Ring-Buffered Similarity Matrix. The exhaustive solution fills each cell of the similarity matrix SM (i.e., the input to the Hungarian algorithm) from scratch, causing redundant data movement. For two adjacent k-width window pairs (W_{S_i}, W_{T_j}) and $(W_{S_i}, W_{T_{j+1}})$, SM differs by one column, i.e., an incoming token replaces one token of the current window. Hence, we implement a ring-buffered SM and overwrite the column of the leaving vertex with new data. This reduces the data movement effort per cell of A from $O(k^2)$ to $O(k)$. Note that the Hungarian algorithm still copies the values of SM into its local buffer B as the values of B are modified to jump-start the algorithm.

Discussion. Despite our ring-buffer optimization, our experiments confirm that the exhaustive solution exhibits prohibitively high runtimes, in particular for large text corpora. However, we observe that we do not have to compute all matrix cells to derive the final result. To this end, we propose upper bounds to prune matrix cells and avoid expensive invocations of the Hungarian algorithm.

4 BOUND-BASED CELL PRUNING

Establishing bounds on the maximum matching score of k-window pairs (i.e., cells of the alignment matrix A) presents a crucial strategy to prune cells and reduce the number of calls to the Hungarian algorithm. This section introduces two types of bounds: (1) Bounds that address individual matrix cells and are independent of neighboring cells, and (2) bounds that exploit the sliding-window property.

4.1 Effective Matrix Cell Pruning

We define four bounds on individual matrix cells, prove their correctness, and summarize their intuitions in Table 2.

The *Maximum-Similarity-Value* (MSV) bound is k times the largest similarity value in the similarity matrix SM, i.e., the input of each cell to the Hungarian algorithm; $MSV = k \cdot c$ with $c = \max\{c' \mid c' \in SM\}$. This intuitively forms a bound as we use the largest similarity of two tokens from W_{S_i}, W_{T_j} for each of the k mappings we aim to find.

For the *Maximum-Row-Sum* (MRS) bound, we sum up the max. similarity values of *each row*; $MRS = \sum_{\text{Row } i \in SM} \max\{SM[i]\}$. Analogously, we introduce the *Maximum-Column-Sum* (MCS) bound on the maximum of *each column*; $MCS = \sum_{\text{Col } j \in SM} \max\{SM[*][j]\}$. Notably, the MRS and MCS bound have similar tightness, but differ for specific pairs of k-width windows. Hence, we introduce the M^3 bound as the minimum of MRS and MCS bound. By definition, the following holds for the expected tightness of the bounds:

$$\mathcal{WO}(W_{S_i}, W_{T_j}) \leq M^3 \leq MRS \leq MSV$$

All bounds exhibit $O(k^2)$ time complexity, i.e., we need a complete scan of SM. Thus, we use a single function $getBounds(SM)$ computing the bounds and normalizing them by k, i.e., to $[0,1]$.

Correctness of the M^3 Bound. We prove that MRS and MCS bound do *not* underestimate the true window similarity $\mathcal{WO}(W_{S_i}, W_{T_j})$. This proves the correctness of the M^3 bound, $M^3 = \min\{MRS, MCS\}$.

Lemma 4.1. *The MRS bound is a correct upper bound, i.e., the following holds:* $\mathcal{WO}(W_{S_i}, W_{T_j}) \leq MRS$.

Proof. Let r_i denote a maximum element in the i-th row of SM and M a one-to-one matching according to Definition 2.2. From the one-to-one property of M, it follows that the matching cost c_i of each token is from a different row i in SM, and thus $c_i \leq r_i$ holds for all rows. Consequently, $\mathcal{WO}(W_{S_i}, W_{T_j}) = \sum_i c_i \leq MRS = \sum_i r_i$ also holds. □

Lemma 4.2. *The MCS bound is a correct upper bound, i.e., the following holds:* $\mathcal{WO}(W_{S_i}, W_{T_j}) \leq MCS$.

Proof. Analogously to the proof of MRS (cf. Lemma 4.1). □

Our experiments (cf. Section 6) reveal that our bounds are highly effective on all corpora: Typically, over 90% of all matrix cells are pruned. However, the runtime improves only by half an order of magnitude w.r.t. to the baseline solution. The reasons are twofold: (1) All bounds require $O(k^2)$ time and (2) are computed for individual matrix cells without leveraging the sliding-window property. In the next subsection, we investigate on (1) computationally cheaper bounds that (2) exploit the sliding-window property.

4.2 Sliding Window Bounds

Although the bounds of Section 4.1 are highly effective, their $O(k^2)$ cost makes them impractical for time-sensitive settings. To address this, we derive two looser bounds that are incrementally updated in $O(1)$ and $O(k)$ time as the window slides, respectively.

4.2.1 $O(1)$ Bound. Assume that we know the similarity of the prior matrix cell $A[i, j-1]$. Then, we can compute an upper bound on the similarity of the cell $A[i,j]$ as follows:

$$A[i,j] \leq A[i,j-1] + \frac{1.0}{k} \tag{1}$$

Specifically, we leverage the fact that the similarity of any two tokens is between 0 and 1. Recall that neighboring cells of the alignment matrix A differ only by two tokens: The oldest token is replaced by a new token. This is a special case of Lemma 4.3 with $L_{min} = 0$ and $I_{max} = 1$. For the $O(1)$ bound, we assume that the *leaving* token contributes minimum similarity: $L_{min} = 0$ and the *incoming* token contributes maximal similarity (normalized by k): $I_{max} = \frac{1.0}{k}$. Hence, $A[i,j] \leq A[i,j-1] + \frac{1.0}{k}$ is a correct bound. For small values of k, we expect the bound to be rather loose. However, with increasing k, the tightness of the bound improves, which is particularly valuable as the runtime of the Hungarian algorithm is cubic in k.

[1] SM contains the semantic similarities of two k-width windows W_{S_i} and W_{T_j}, whereas the alignment matrix A contains the semantic similarities of *all* window pairs.

Table 2: Intuition of all bounds (example SM, $k=3$). True max. matching cells in blue, max. value bound in yellow.

sim	t_i	t_{i+1}	t_{i+2}	max
s_i	0.9	1.0	0.2	1
s_{i+1}	0.0	0.85	0.2	0.85
s_{i+2}	0.0	0.2	0.75	0.75
max	**0.9**	**1.0**	**0.75**	$\mathcal{WO}=2.5$

Bound	Value
MSV	3
MCS	2.65
MRS	2.6
M^3	2.6

4.2.2 $O(k)$ Bound. Consider the similarity matrices $SM_{i,j-1}$ and $SM_{i,j}$ of two subsequent matrix cells $A[i,j-1]$ and $A[i,j]$. Further, let the leaving and incoming matrix columns be $L=SM_{i,j-1}[*][0]$ and $I=SM_{i,j}[*][k-1]$, respectively. $SM_{i,j-1}$ and $SM_{i,j}$ only differ in a single column and all other matrix cells remain unchanged. We exploit this fact to establish an upper bound in Lemma 4.3.

Lemma 4.3. *Assume two subsequent matrix cells $A[i,j-1]$ and $A[i,j]$ and their corresponding similarity matrices $SM_{i,j-1}$ and $SM_{i,j}$. Let $L_{min}=\min\{SM_{i,j-1}[*][0]\}$ be the min. value of the leaving column in $SM_{i,j}$ and $I_{max}=\max\{SM_{i,j}[*][k-1]\}$ be the max. value of the incoming column. Then, the following inequality holds:*

$$A[i,j] \leq A[i,j-1] - \frac{L_{min}}{k} + \frac{I_{max}}{k} \tag{2}$$

Proof. Consider a maximum matching $M_{i,j-1}$ of a window pair $(W_{S_i}, W_{T_{j-1}})$. The corresponding similarity $\mathcal{WO}$ w.r.t. to Def 2.2 is:

$$\mathcal{WO}_{i,j-1}\left(W_{S_i}, W_{T_{j-1}}\right) = \sum_{s \in W_{S_i}} sim\left(s, M_{i,j-1}(s)\right)$$

Any other valid matching $M' \in \mathcal{M}$ must not have a larger value:

$$\sum_{s \in W_{S_i}} sim\left(s, M_{i,j-1}(s)\right) \geq \sum_{s \in W_{S_i}} sim\left(s, M'_{i,j-1}(s)\right) = \mathcal{WO}'_{i,j-1}$$

Deleting the leaving vertex from any matching, we lose at least L_{min} similarity. This is an upper bound on the matching of the tokens in $(W_{S_i}, W_{T_{j-1}} \cap W_{T_j})$: $\mathcal{WO}_{i,j-1} - L_{min} \geq \mathcal{WO}'_{i,j-1} - L_{min}$.

To arrive at a bound for the subsequent window pair (W_{S_i}, W_{T_j}), we add the largest value I_{max} in I to all the matchings. This is an upper bound, because we ignore the one-to-one matching property:

$$\mathcal{WO}_{i,j} \leq \mathcal{WO}_{i,j-1} - L_{min} + I_{max} \geq \mathcal{WO}'_{i,j-1} - L_{min} + I_{max}$$

Normalizing, $\mathcal{WO}_{i,j}$ to $A[i,j]$ according to (see Def 2.3) creates:

$$A[i,j] \leq A[i,j-1] - \frac{L_{\min}}{k} + \frac{I_{\max}}{k}$$

□

While the $O(1)$ and $O(k)$ bounds are computationally less expensive, they are expected to be looser than, e.g., the MCS bound. However, we can leverage this fact to consecutively apply bounds with increasing tightness and runtime on matrix cells. Next, we propose SeDA, which implements a bound cascade to effectively translate pruning power into significant runtime improvements.

Algorithm 1: SeDA: Semantic Document Alignment

Input: (S,T) : sequences, k : window width, θ : threshold, sim : similarity function, W_s : k-width window of S
Output: A : alignment matrix

```
1 B ← getCandidatesBitVector(S,T,k,θ,sim)
2 A ← alignment matrix of size |W_S|×|W_T|
3 for W_{S_i} ∈ W_S do
4     let C be an all-zero bit vector of size |W_T|
5     for s ∈ W_{S_i} do C ← C | B_s                    // bitwise OR
6     let R be a list of all candidate runs (r_b,r_e) ∈ C
7     for (r_b,r_e) ∈ R do validateCandRun(S,T,k,θ,A_i,(r_b,r_e))
8 return A
```

5 SEDA: TIME-EFFICIENT SEMANTIC DOCUMENT ALIGNMENT

SeDA is a time-efficient solution for semantic document alignment. We provide a high-level overview of SeDA's core components below and discuss them in detail in the subsequent subsections: (1) A *candidate generation technique* avoids iterating over all cells of the quadric-size alignment matrix. (2) A *bound cascade* routes candidates through multiple bounds with increasing tightness. (3) A *zero-copy heuristic* for the Hungarian algorithm exploits information from the bound computations to avoid expensive initialization steps.

5.1 Algorithm Outline

Our objective is to compute all cells of the alignment matrix A that exceed the similarity threshold θ. A consists of $O(|S| \cdot |T|)$ cells, and the cell similarity is computed with the Hungarian algorithm in $O(k^3)$ time. As a result, the overall complexity of the baseline solution is $O(|S| \cdot |T| \cdot k^3)$. This asks for further improvements: (1) A reduction of the number of cells for which we have to compute the similarity and (2) an efficient similarity and bound computation. In SeDA, we reduce the number of cells with a candidate generation technique to avoid the $O(|S| \cdot |T|)$ complexity w.r.t. the sequence lengths. All other optimizations target at reducing the cost of pruning a candidate cell and computing the cell similarity.

Conceptually, SeDA computes A row-wise in three steps (cf. Figure 3): (1) For each row in A, SeDA determines so-called *candidate cells*. Candidates are stored as *runs* of adjacent cells in the same row. This is related to the observation that the bound used for candidate generation produces candidate-cell runs of k cells, which might overlap with the next run. (2) We exclude candidate cells from individual runs that do not satisfy our bounds. For each cell in a run, we use a cascade of three bounds to avoid computing the exact cell similarity. The bounds are applied in increasing order of tightness and computational complexity. We stop the cell-similarity computation once a bound returns a value $v < \theta$ and continue with the next cell. For an efficient bound computation, we heavily reuse intermediate results of looser bounds and the bounds of previous cells. (3) SeDA computes the exact cell similarity for candidate cells that survive the bound cascade using an improved, zero-copy Hungarian algorithm.

To this end, SeDA introduces techniques that accelerate computation in practical settings, but it does not aim to improve the

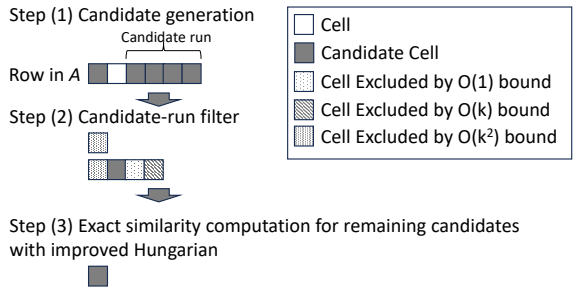


Figure 3: Conceptual overview of SeDA in three major steps.

inherent worst-case runtime of $O(|S||T|k^3)$ for the semantic document alignment problem. Consider, for example, a user-defined similarity threshold θ that lies below all pairwise window similarities. In such a case, any method that solves the semantic document alignment problem (cf. Definition 2.3) must compute all $|S| \times |T|$ similarity values in the alignment matrix A, each of which in turn requires computing an exact window similarity.

Running Example. We illustrate the algorithmic details of SeDA using T' of Example 1.1 and a shortened version of S', namely $S'' = [s_1 = \text{throne}, s_2 = \text{kingdom}, s_3 = \text{shushan}, s_4 = \text{kingdom}]$, i.e., $s_2 = s_4$. Further, we use window size $k = 3$ and similarity threshold $\theta = 0.6$.

Parameter Setting. We recommend to set the parameters k and θ as follows: We propose to select k according to the avg. sentence length. For θ, the selection depends on the used semantics: Sample the pair-wise token similarity of the used semantics to approximate the probability distribution. Next, select a use-case-specific fraction of similar token pairs one conceives similar and set θ accordingly.

5.2 Step 1: Candidate Generation

Computing all cells of A in quadratic time w.r.t. the sequence length renders the baseline solution infeasible. Hence, we advocate for a candidate generation technique that substantially reduces the computation load: For two sequences S, T, our technique efficiently generates a list of potential candidate cells. A *candidate* is a cell in A whose corresponding windows W_{S_i}, W_{T_j} contain at least one token pair $(s \in W_{S_i}, t \in W_{T_j})$ with $sim(s,t) \geq \theta$. To this end, we exploit the token-based *MSV* bound.

Algorithm 2: getCandidatesBitVector

Input: (S,T) : sequences, k : window width, θ : threshold, sim : similarity function
Output: B : Bit vector for $s \in S$ with candidates

```
1 B ← ∅; W_T ← computeSlidingWindows(T,k)
2 for each unique s ∈ S do
3     let N_s be a list of all token t ∈ T with sim(s,t) ≥ θ
4     let B_s be an all-zero bit vector of size |W_T|
5     for W_Tj ∈ W_T do let B_s[j] ← 1 if W_Tj ∩ N_s ≠ ∅
6     B[s] ← B_s
7 return B
```

Table 3: Results of Algorithm 2 on seq. T', S'', $k = 3$, and $\theta = 0.6$. One bit-vector index B_i for each unique token in S''.

N_i	B_i	W_{T_1}	W_{T_2}	W_{T_3}	W_{T_4}	W_{T_5}	W_{T_6}
$N_1 = \{t_5, t_6\}$	B_1	0	0	1	1	1	1
$N_2 = \{t_1\}$	B_2	1	0	0	0	0	0
$N_3 = \emptyset$	B_3	0	0	0	0	0	0

Table 4: Complete candidate-runs for T',S'', $k = 3$. One list of candidate runs R for every k-width window in S ($s_2 = s_4$).

W_{S_i}	C	W_{T_1}	W_{T_2}	W_{T_3}	W_{T_4}	W_{T_5}	W_{T_6}	R
$\{s_1, s_2, s_3\}$	$B_1 \mid B_2 \mid B_3$	1	0	1	1	1	1	(1,1),(3,6)
$\{s_2, s_3, s_4\}$	$B_2 \mid B_3 \mid B_2$	1	0	0	0	0	0	(1,1)

Conceptually, we compute candidate cells of A row-wise and prevent repetitive computations by pre-computing additional information for overlapping windows. Next, we describe both aspects in detail, and Algorithm 2 shows the complete candidate generation.

5.2.1 Step 1.1: Neighborhood Bit-Vectors. Table 3 illustrates the preprocessing step. Initially, we compute a *neighborhood index* N_s for each unique token $s \in S$. N_s is the list of all tokens $t \in T$ with similarity $sim(s,t) \geq \theta$, i.e., token pairs that produce candidates. In our example, S'' consists of three unique tokens (s_1, $s_2 = s_4$, s_3), and Table 3 shows the corresponding neighborhood indexes N_1-N_3. With each unique token s_i, we associate an initially all-zero bit-vector B_i of size $|W_T|$. If window $W_{T_j} \in W_T$ contains a token from N_s, we set bit $B_s[j]$ indicating that W_{T_j} is a candidate. Note that every non-border token occurs in k consecutive windows.

Example 5.1. For B_1 in Table 3, we set bits 3–5 as $t_5 \in N_1$ and bits 4–6 since $t_6 \in N_1$, which illustrates overlapping candidate runs.

5.2.2 Step 1.2: Candidate Determination per Row. SeDA generates *all* candidate runs for a specific row in A. Recall that a row represents one k-width window W_{S_i} (cf. Table 4). We compute a *candidate bit-vector* C as the bit-wise OR of the corresponding bit-vectors, e.g., B_1-B_3 for window $W_{S_1} = \{s_1, s_2, s_3\}$. SeDA condenses C to a list of candidate runs R as shown in Table 4 (cf. Algorithm 1, Lines 5–7). Next, in Step 2, we reduce the number of candidates by keeping only candidate runs that pass our bound cascade.

5.3 Step 2: The Bound Cascade

The candidate generation improves the runtime by reducing the number of Hungarian algorithm invocations for determining the correct cell similarity $\mathcal{WO}(W_{S_i}, W_{T_j})$. To further improve the runtime, each candidate cell has to pass a *cascade* of bounds with increasing tightness and runtime complexity, as shown in Algorithm 3. Specifically, the cascade uses the $O(1)$ and $O(k)$ bounds, exploiting the sliding window property (cf. Section 4.2). Finally, we use the *MCS* bound from Section 4.1, originally having a complexity of $O(k^2)$. By reusing information computed for preceding bounds of the cascade, we can reduce its complexity to $O(k)$, as we show next. We illustrate the interplay of the bound cascade with our running example.

MCS Bound in $O(k)$ Time. We use the *MCS* bound as the final bound in our cascade (cf. Section 4.1). This bound is slightly looser than the M^3 bound, but its complexity can be reduced from $O(k^2)$ to $O(k)$ if the bound for the prior cell is known.

Lemma 5.2. *Given two bipartite graphs of consecutive windows (W_{S_i}, W_{T_j}) and $(W_{S_i}, W_{T_{j+1}})$ with similarity matrices SM_j and SM_{j+1}, respectively. The MCS bound for SM_{j+1} can be computed in $O(k)$ time, if the MCS bound of SM_j is known; otherwise, the MCS bound must be computed from scratch, which requires $O(k^2)$ time.*

Proof. The proof follows from the *MCS* bound definition. The *MCS* bound on SM_j is the sum of the max. values of each column in SM_j, i.e., we read all cells in $O(k^2)$ time. SM_j and SM_{j+1} share $k-1$ columns, L is the leaving column, and I is the incoming column. For the *MCS* bound on SM_{j+1}, we compute the max. value of I, add it to the bound of SM_j, and subtract the cached min. value of L. L's min. and max. values are computed when L is an incoming column. □

5.4 Step 3: Hungarian Algorithm Optimization

The Hungarian algorithm (cf. Section 3.1) has two steps: (1) A heuristic jump-starts it and computes a partial mapping. (2) Incrementally extend the partial mapping to derive the final mapping (main loop).

Our analysis reveals that the jump-start heuristic has a large share of runtime as it loops three times over SM, i.e., has $O(k^2)$ time complexity. In contrast, the number of iterations in the main loop is small. Moreover, the jump-start heuristic copies SM into a buffer of size $k \times k$ as it updates the values. In particular for small values of k, we observe that the jump-start heuristic incurs significant overhead.

We introduce a novel heuristic that (1) leverages information from the bound computations to derive (2) a zero-copy heuristic, but (3) sacrifices some tightness compared to the jump-start heuristic. After preprocessing, we are interested in cells with similarity 0. Hence, our heuristic considers only a subset of SM, i.e., cells of SM with a value equal to the respective column maxima, which are already known from the $O(k)$ and the *MCS* bound. This results in our efficient *zero-copy* heuristic that avoids copies of SM (no updates) and loops over SM only once (instead of three times).

Algorithm 3: ValidateCandRun

Input: (S,T,k,θ) : as before, A_i : line of s_i in A, (r_b,r_e) : begin & end of candidate run
Output: A_i : alignment matrix line

```
1  E_{j-1} ← 1.0 // Bound of previous cell, 1.0 means there is no bound
2  Initialize SM with edge weights of W_{S_i} × W_{T_j}
3  for j ∈ [r_b, r_s) do
4      L_min ← min(SM[*, (j-1) mod k])   // Leaving column min.
5      for l ∈ [0,k) do                  // Update column (j-1) mod k of SM
6          SM[l, (j-1) mod k] ← sim(s_{i+l}, t_{j+k-1})
7      ub ← E_{j-1} + 1.0/k              // O(1) bound
8      if ub ≥ θ then
9          I_max ← max(SM[*, (j-1) mod k])   // Incoming max.
10         ub ← E_{j-1} ← E_{j-1} - L_min + I_max   // O(k) bound
11         if ub ≥ θ then
12             ub ← E_{j-1} ← getBounds(SM)   // O(k²) bound
13             if ub ≥ θ then
14                 c = HungarianSolve(SM) / k
15                 if c ≥ θ then A[i,j] ← E_{j-1} ← c
16 return A_i
```

Table 5: Absolute bounds for run (3,6) of $W_{S_1''}$; $k \cdot \theta = 1.8$.

	$O(1)$	$O(k)$	$O(k^2)$	$\mathcal{WO}$	$b(\mathcal{WO})$
$(W_{S_1''}, W_{T_3})$	-	-	1.0	-	1.0
$(W_{S_1''}, W_{T_4})$	2.0	2.0	1.9	1.7	1.7
$(W_{S_1''}, W_{T_5})$	2.7	1.8	1.7	-	1.7
$(W_{S_1''}, W_{T_6})$	2.7	1.7	-	-	1.7

6 EMPIRICAL EVALUATION

We conduct two sets of experiments that investigate (1) the result quality of SeDA on three different use cases, and (2) its efficiency.

Competitors. We compare **SeDA** to the competitors listed in Table 6. From the domain of purely *syntactic* competitors, locality sensitive hashing (LSH) schemes approximating Jaccard similarity of token windows using MinHashes [3] are state of the art [23, 37]. In our result quality experiments, we use the *exact* **Jaccard** similarity as an upper bound on the result quality of purely syntactic competitors. Additionally, we consider the most recent approach that computes all one-permutation MinHashes efficiently, namely **OPH** [23]. In the runtime experiments, we also consider a syntactic approach named **TxtAlign** [37] that compresses MinHashes into compact windows. From the domain of semantic competitors, we use **FastText** as a well-established procedure averaging word embeddings [18]. Finally, we use one of the most recent **SBERT** variants (bge/m3) [5] as an instance of present-day sentence encoders and LLMs [27], and the winner of PAN14, **SEF** [28], as the gold standard for the plagiarism detection use case. Recall that the envisioned application of SeDA (and all purely syntactic competitors) is to serve as candidate generation mechanism, whose results are analyzed by slow AI-based tools or humans. To this end, the results of SBERT form an upper bound on the result quality in Section 6.1.

Experimental Setup & Datasets. All experiments have been conducted single-threaded on a machine equipped with an AMD Epyc 7313 16-core processor (512 KiB L1-, 2 MiB L2-, and 64 MiB L3-cache) and 64 GB of main memory, running Debian 12 (`bookworm 6.1.158-1`) and using `g++` 12.2.0 with optimization level `-O3`.

In our experiments, we use three datasets with different characteristics. The **Bible** corpus [12] is composed of German bible revisions of the Book of Ester, including (i) the original translation to German of Martin Luther, (ii) two canonical editions of Schlachter and Elberfelder, (iii) a canonical edition of the New Evangelican version with simplified language, and (iv) the non-canonical Volx Bible presenting the biblical texts in modern youth language. It consists of 5 documents with an avg. length of 5497.8, and 2697 unique tokens (27,489 tokens in total). All texts are editions of the same text. Hence, the high average window similarity is expected to challenge our bounds. The **PAN11** corpus is a widely adopted benchmark to evaluate plagiarism detection systems [23, 24, 37]. It contains 31 documents with

Table 6: Competitors (Syn. = syntactic, Sem. = semantic).

	Name	Remark	Used in
syn.	Jaccard	exact k-windowed Jaccard	Sec 6.1
	OPH [23]	MinHashes (LSH)	Sec 6
	TxtAlign [37]	compressed MinHashes (LSH)	Sec 6.2
	SEF [28]	Plagiarism detection tool	Sec 6.1.1
sem.	FastText [18]	Averages Word Embeddings	Sec 6
	SBERT [27]	Sentence Transformer	Sec 6

Table 7: Plagiarism Use Case: Best observed result.

Approach	θ	k	Precision	Recall	F1-Score
SeDA	0.63	15	0.90	0.95	0.93
SBERT	0.76	-	0.93	0.85	0.89
SEF	-	-	0.88	0.88	0.88
Jaccard	0.30	15	0.78	0.97	0.87
FastText	0.88	12	0.91	0.84	0.87
OPH	0.42	11	0.83	0.82	0.83

an avg. length of 3061.6, and 13,105 unique tokens (94,911 tokens in total). **Wiki** [1] is our largest corpus, which comprises of 2077 Wikipedia articles with an avg. length of 493.2, and 51.377 unique tokens (1,024,432 tokens in total). For the Wiki corpus, we extract and concatenate the main text (disregarding recurring sections like "References" and metadata information) from the corresponding Parquet files until we reach the desired number of tokens. All datasets are consistently preprocessed according to [37][2]. We use pre-trained general-purpose word embeddings from [16]. The problem of selecting an optimized embedding is orthogonal to our contributions, but is expected to improve SeDA's runtime and result quality.

6.1 Comparison of the Result Quality

This section evaluates the result quality of SeDA by postprocessing the alignment matrix A for three use cases with different data properties and queries: (1) Plagiarism detection, (2) alignment of Bible revisions, and (3) candidate generation for large Wikipedia articles.

6.1.1 Plagiarism Detection Experiment. A common application of document alignment is plagiarism detection [14]. For the comparison, the metrics *precision* and *recall* follow the common understanding, and their formal application to plagiarism detection as proposed in Potthast et al. [25]. We show the F1-Score of precision and recall and use the following general setup:

(1) Compute the alignment matrix A for each pair (S,T) of suspicious and source document of the PAN benchmark.
(2) A cell $A[i,j]$ is marked as a plagiarism case if (a) $A[i,j] \geq \theta$, or (b) at least one neighboring cell is marked and $A[i,j] \geq 0.5$.
(3) A window W_{S_i} is a plagiarism case if the corresponding row in A contains at least one marked cell.

Results. Table 7 shows the results for all approaches. We list the numbers for the configurations with the highest F1-Score of precision and recall for all parameter combinations of $k \in [3,15]$ and $\theta \in [0,1]$ with an increment of 0.01. Figure 4 shows the interplay of window length k and threshold θ for all approaches[3].

Compared to Jaccard and OPH, SeDA reveals a clear superiority w.r.t. result quality. Our results also suggest that SeDA outperforms the semantic competitors FastText and SBERT w.r.t. result quality, which is unexpected for SBERT. This results from the fact that SBERT operates on lexicographical sentences rather than k-width windows. The texts partially contain short sentences like 'He agrees.', which provoke false casual matches. A workaround could limit the minimum length of a plagiarized passage, but domain-specific optimization are beyond the scope of this paper. This also holds for SeDA, which suffers from *enveloping* plagiarized passages where the detected plagiarized passage starts too early and finishes too late: The first (last) windows of a plagiarized passage often contain tokens of the prior (next) sentence. Domain-specific tuning is implemented in the detection rule sets of specialized approaches like SEF [28], explaining its high F1-Score compared to the other syntactic approaches.

The plots in Figure 4 indicate that the threshold θ is the decisive factor for result quality. Increasing k has an observable positive effect only for small values of $k < 10$ and $k \in [10,15]$ is a valid choice for all approaches.

6.1.2 Bible Experiment. This experiment is motivated by one of our ongoing collaborations with Bible scientists and introduces a corpus of semantically very *similar* texts. This allows to study how our proposed solution performs in such cases.

We select one query document Q and the remaining Bible texts form the database $\mathcal{D}$. As query Q, we use the first German translation, i.e., the Luther bible. For each document pair (Q,D), $D \in \mathcal{D}$, we compute the following: For each paragraph $p \in Q$, we compute the most similar paragraph $p' \in D$, i.e., execute a top-1 query. The ground truth for correctness is the canonical numbering of the paragraphs in the Bible. We determine the mapping accuracy per document pair (Q,D) as the number of correctly detected pairs (p,p') divided by the number of paragraphs in Q. SeDA computes the similarity between two paragraphs in three steps:

(1) Compute the k-window alignment matrix A (cf Sec. 2).
(2) Determine the maximum for each row and column in A.
(3) Return the average of all row and column maxima from (2).

Results. We show the results of all canonical translations in Figure 5a. For canonical translations, all approaches are expected to achieve very good mapping accuracies close to 1. SBERT delivers optimal result quality. The next best approach is SeDA with a mapping accuracy of about 0.95, outperforming all syntactic competitors and the older semantic competitor FastText. For all three window-based approaches (SeDA, Jaccard, and FastText), the results indicate that the window length k is of minor importance.

A detailed inspection of incorrect mappings suggests that the quality of the mappings differs among the document pairs and correlates with the similarity assessed by domain experts. That is, there are few mapping errors for the Bible versions edited by Schlachter

[2] https://github.com/rutgers-db/RangeAllign-code

[3] Results of Jaccard and OPH are almost identical; thus, we show them in a single plot.

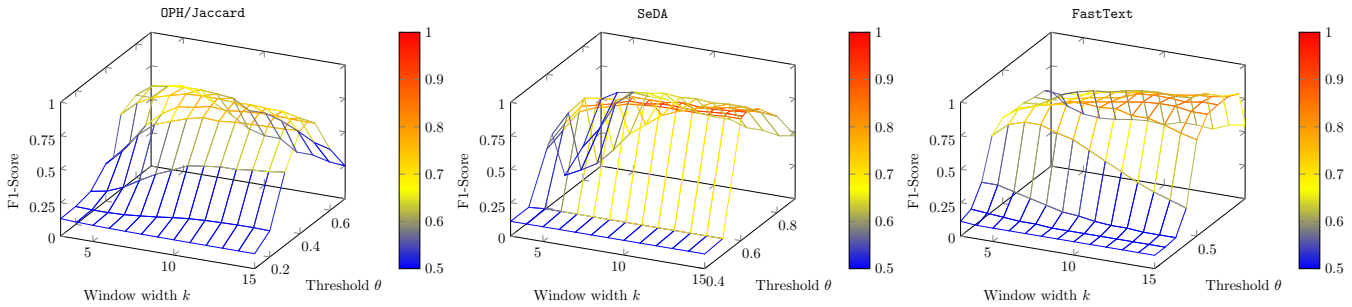
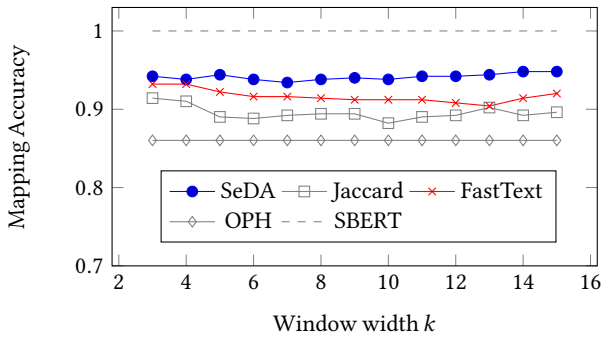
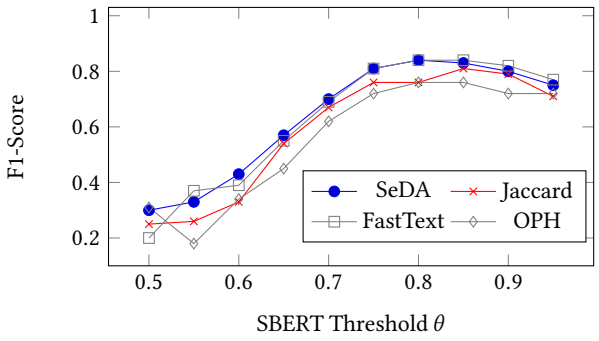


Figure 4: PAN Experiment: Interplay of threshold θ and window width k with the corresponding F1-Score of precision and recall.



(a) Mapping accuracy over increasing window width k for the Bible experiment.



(b) F1-Score of candidate cells over increasing SBERT thresholds for the Wikipedia experiment.

Figure 5: Correctness results for the Bible and Wikipedia experiment, respectively.

and Elberfelder. The majority of mapping errors occur between the Luther Bible and the New Evangelic Version. This version simplifies language and partially exchanges names and places, e.g., "Xerxes" vs. "Ahasveros". Our results suggest the hypothesis that the mapping accuracy as well as the similarity of the mapped paragraphs are indicators for the relatedness and therefore, for the desired genealogy of the Bible revision. This hypothesis is supported by the mapping accuracy to a non-canonical translation like the Volxbible. The mapping accuracy deteriorates for all approaches: SBERT 0.8, SeDA 0.38, Jaccard 0.25, FastText 0.31, OPH 0.18. Concerning the Bible use case, we conclude that SeDA and our underlying problem definition is well suited because it produces the best mapping accuracy of the non-AI-based approaches.

6.1.3 Wikipedia Experiment. This use case targets at the envisioned setup: A fast approach determines candidates, which are then examined manually or with computationally expensive methods like SBERT. To this end, we study the applicability of all approaches as candidate-generation method. We measure the F1-Score of precision and recall, and use $\theta = 0.5$ as lowest SBERT threshold (i.e., the similarity of two random sentences within the Wikipedia corpus). Ideally, for an SBERT threshold $\theta \in [0.5,1]$, all candidates produced by the other approaches exceed θ (high precision) and no true candidates are discarded (high recall). We use the following general setup:

(1) Split the corpus into (overlapping) k-width windows and compute A for each approach.
(2) Per approach: Train a regression model on (A_{SBERT}, A).
(3) Determine the F1-Score of precision and recall for SBERT thresholds $\theta \in [0.5,0.95]$ with an increment of 0.05.

Notably, we train a regression model to translate the similarity values of the approaches SeDA, Jaccard, FastText, and OPH into SBERT similarity values. Since the relationship of all approaches and SBERT similarity values is not entirely linear, we train one regression model for each 0.1 interval. We use $k = 10$, a small value that has shown good quality in the other experiments and is close to the avg. length of a sentence after stop word removal.

Results. The overall result pattern of all approaches in Figure 5b is very similar: With increasing thresholds θ, the F1-Score increases. Initially, the F1-Score does not exceed 0.4 for an SBERT threshold $\theta = 0.5$. The score increases until reaching an optimum at around $\theta = 0.85$ and then marginally decreases until $\theta = 1.0$. This is a direct result of two effects: (1) The distribution of the similarity values forms a Gaussian centered at $\theta = 0.5$ and (2) the deviation of the similarity values is high for small values of θ, say $0.4 \leq \theta \leq 0.6$. Hence, the F1-Score is dominated by k-width windows that are not particularly similar or dissimilar while the similarity values between SBERT and the approaches deviate significantly. However, this does not imply a conceptual problem as we are interested only in similar text parts. The global result patterns for relevant thresholds to determine cases of plagiarism (SBERT $\theta \geq 0.83$) and similarity of the top-1 queries from the Bible use case, the correlation of similarity approaches

(after regression) and SBERT is in the range of the best F1-Scores of all approaches. For all SBERT thresholds, there is a clear hierarchy of approaches according to their F1-Score: The best two approaches are SeDA and FastText. The next best approach is Jaccard, clearly outperforming OPH, which approximates the Jaccard scores using LSH. This hierarchy results from SeDA and FastText relying on semantic information, whereas Jaccard and OPH consider only syntactic information. In summary, SeDA is among the two best approaches w.r.t. result quality and thus, is an eligible candidate generation approach.

6.2 Efficiency Experiments

We start with an intrinsic investigation on the strength of our bounds. Finally, in our extrinsic evaluation, we study response time characteristics and compare SeDA to state-of-the-art approaches.

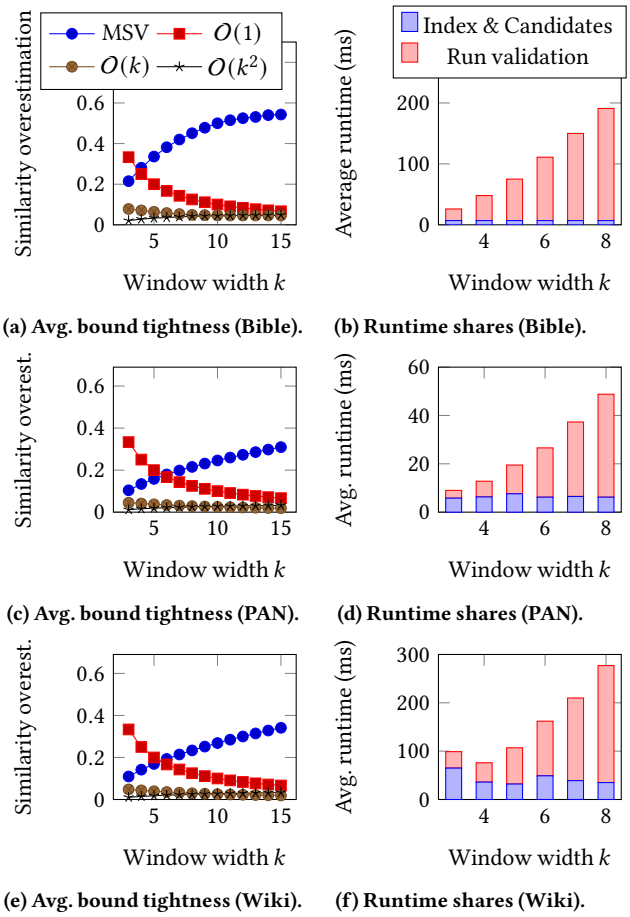
6.2.1 Bound Effectiveness. In this set of experiments, we investigate the effectiveness of our upper bounds and we are interested in two measures: (1) *Bound Tightness:* The avg. difference between a specific bound and the true cell similarity $\mathcal{WO}(W_{S_i}, W_{T_j})$ for different window sizes k. This measure is independent of θ. (2) *Bound Cascade Pruning Power:* We study the relationship between θ and the number of cells that pass a specific bound.

Average Bound Tightness. In Figure 6a,c,e, we show the avg. bound tightness for documents across all corpora. The results indicate that the avg. bound tightness of PAN11 and Wikipedia are almost identical. In contrast, the similarity overestimation for the Bible corpus is observably higher. This is a result of the different avg. similarity of the corpora. PAN11 and Wikipedia contain dissimilar texts, whereas the Bible corpus contains very similar texts. This affects the bound tightness, i.e., the more dissimilar the texts, the tighter are our bounds.

We further observe that bound-similarity overestimation highly varies among different bounds, and the pattern is the same for all corpora. As expected, computationally cheap bounds offer a looser bound. Interestingly, the $O(1)$ bound tends to converge to the tightness of the $O(k)$ and $O(k^2)$ bounds. That is, it is more effective for increasing k due to the decreasing impact of a single token for larger windows. For example, the maximum difference between two subsequent cells in the alignment matrix for $k=3$ is 0.33, whereas for $k=8$ it is 0.125.

In Figure 6, we also depict the avg. response time per alignment matrix computation for the entire bound cascade. Our results reveal that candidate-run validation is the major cost factor for reasonable values of $k \geq 5$. In other words, the overhead for on-the-fly index construction and candidate generation is negligible. This is the envisioned result and also holds for the additional memory consumption (dominated by the size of A).

Average Pruning Power. Next, we investigate how many cells of the alignment matrix A pass a specific bound. In contrast to the prior experiment, we expect that this mostly depends on θ. Table 8 contains the avg. fraction of candidates that remain after applying the corresponding bound for different values of θ and $k=10$. The results of this experiment are consistent with the results of the prior experiment. There is a clear hierarchy of the bounds, with the MSV bound being the loosest one. The results further suggest that each stage of our bound cascade effectively reduces the number of cells for which the true similarity value must be computed. Therefore,



(a) Avg. bound tightness (Bible). (b) Runtime shares (Bible).

(c) Avg. bound tightness (PAN). (d) Runtime shares (PAN).

(e) Avg. bound tightness (Wiki). (f) Runtime shares (Wiki).

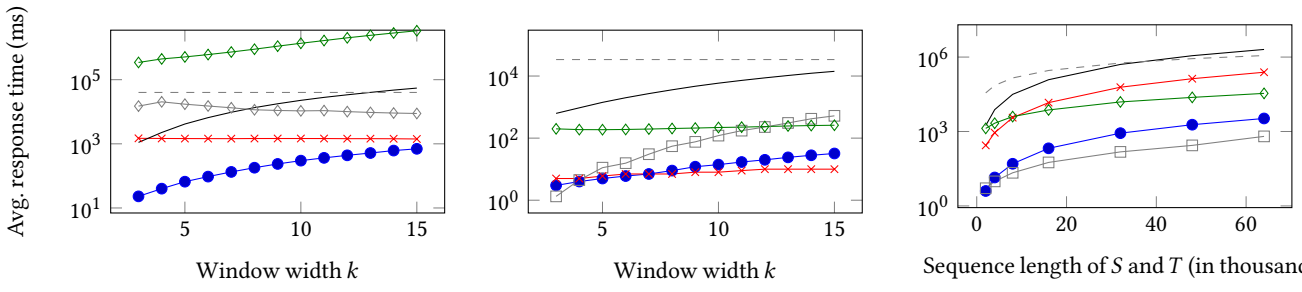
Figure 6: Bound analysis for varying sliding window width k.

independent of the dataset, we compute the true similarity for less than one percent of the alignment matrix cells.

6.2.2 Runtime Experiments. This section compares the runtime performance of SeDA and the baseline solution to the competitors. Our primary interest is to quantify the response-time improvement compared to SBERT, i.e., the gold standard for result quality. Thankfully, we received the implementations of TxtAlign and OPH from the original authors. Except for SBERT bge/m3, all approaches are implemented in C++ using the same compiler and hardware settings. Note that we use the original Python implementation of SBERT to serve as an upper boundary for a reasonable runtime and thus, the high runtime differences do not invalidate the comparison. The runtime of all semantic competitors, i.e., SeDA, FastText, and SBERT, includes the time for mapping text to vectors and to compute the corresponding similarity. As we use pre-trained models, the runtime does not include training the word or sentence embedding.

Runtime Comparisons. In Figure 7, we depict the response-time results for all three corpora. For θ, we use the values that show the best F1-Score in Table 7[4]. Our results reveal that SeDA outperforms

[4]On the Bible corpus, the original implementation of TxtAlign is much slower than a naive implementation of the Jaccard similarity. This issue also persists after seeking help



(a) Long similar texts: Bible corpus. **(b) Short texts with plagiats: PAN11 corpus.** **(c) Very long dissimilar texts: Wikipedia, $k = 10$.**

Figure 7: Runtime results per document pair: SeDA, TxtAlign (Jaccard), OPH, OPH Original, FastText, Baseline, SBERT (bge-m3).

Table 8: Remaining fraction of cells of A after applying the corresponding bound for $k = 10$.

	Remaining	$\theta = 0.5$	$\theta = 0.6$	$\theta = 0.7$	$\theta = 0.8$	$\theta = 0.9$
Bible	MSV	0.619	0.420	0.320	0.284	0.279
	$O(1)$	0.428	0.231	0.143	0.104	0.090
	$O(k)$	0.210	0.120	0.077	0.058	0.051
	$O(k^2)$	0.016	0.002	0.001	< 0.001	< 0.001
	True pos	0.007	0.001	< 0.001	< 0.001	< 0.001
PAN11	MSV	0.310	0.148	0.084	0.064	0.056
	$O(1)$	0.148	0.054	0.025	0.014	0.011
	$O(k)$	0.021	0.012	0.007	0.006	0.005
	$O(k^2)$	0.002	< 0.001	< 0.001	< 0.001	< 0.001
	True pos	0.001	< 0.001	< 0.001	< 0.001	< 0.001
Wiki	MSV	0.231	0.107	0.065	0.055	0.050
	$O(1)$	0.104	0.037	0.019	0.012	0.010
	$O(k)$	0.016	0.009	0.006	0.005	0.004
	$O(k^2)$	0.001	< 0.001	< 0.001	< 0.001	< 0.001
	True pos	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001

the baseline solution by 1.5–3 orders of magnitude over all corpora. For $k = 10$ (the smallest reasonable value from the result quality experiments), we observe a speedup of 2–3 orders of magnitude compared to SBERT. Compared to the other approaches that perform very differently depending on the setting, SeDA is always among the top-2 fastest approaches.

End-to-end Performance Gain. For the envisioned application of SeDA to serve as candidate generation mechanism, for slower AI-based tools, we can approximate the end-to-end performance gain as follows. Take the number of cells exceeding θ, as shown in Table 8 (Line: True positives). These are the candidates examined by the AI-based tools. Multiply the SBERT runtime from this experiment with the true positive rate and add the runtime of SeDA. This is valid, because SBERT runtime increases linear with number of windows to process. Using $\theta = 0.7$ this results in the following numbers: For the Bible use case 0.34 s vs. 40 seconds, for the PAN11 corpus 0.05 s vs. 34 s, and 0.50 s vs. 288, for the Wikipedia use case with sequence length 16,000 tokens, with the correctness as shown in Figures 4 and 5.

from the original authors. We hypothesize that this due to the high average similarity of the Bible text: TxtAlign finds similar passages redundantly, and the response time is dominated by the exact Jaccard computations on the detected passages. We omit TxtAlign in Figure 7a.

Table 9: SeDA variants: ✓ means component is included.

Name	Filters		$O(1)$ & $O(k)$	Zero-Copy Hungarian	$O(k^2)$
	Corpus	Cand.			
Naive					
k^2					✓
zc-k^2				✓	✓
SeDA-1			✓	✓	✓
SeDA		✓	✓	✓	✓
c-SeDA	✓	✓	✓	✓	✓

6.2.3 Runtime Analysis on Large-scale Corpus. Table 9 lists the SeDA variants and their corresponding components used to investigate on the improvement of our proposed optimizations. We start with the naive approach and subsequently add additional components finally resulting in a variant for large-scale corpora named c-SeDA.

c-SeDA. Our corpus deployment strategy is an extension of the candidate filter: Q is the query sequence and $\mathcal{D} = \{S_1, \ldots, S_m\}$ is a corpus of documents (i.e., sequences) based on a common alphabet $\mathcal{T} = \{t_1, \ldots, t_n\}$ (i.e., the set of unique tokens in the corpus). Given a corpus document S_i with the corresponding set of all k-width windows W_S, we can safely exclude a window W_{S_i} if no token pair $(q \in Q, s \in W_{S_i})$ with $sim(s,q) \geq \theta$ exists. The result is a new upstream filter in our bound cascade that conceptually removes lines from A. Residual lines are called *candidate lines*. The new filter consists of two phases: (1) An *offline indexing* phase executed once per corpus, and (2) an *online document-line filtering* phase executed once per query Q.

The *offline indexing* phase creates an inverted index I that maps each token $t_i \in \mathcal{T}$ to its locations in the corpus. In addition, we precompute a data structure N that stores the similarity of all token pairs $(t_i, t_j) \in \mathcal{T} \times \mathcal{T}$ that exceed some low similarity threshold θ', for example, for $\theta' = 0.6$. The *online* phase first creates N_t from N, having the same meaning as in Algorithm 2 line 3 (cf. revised paper): For each

unique token in Q, it contains all tokens from $\mathcal{T}$ that create candidates. Next, we use N_t and the inverted index I from the offline phase to mark every k-width window in the corpus with ≥ 1 token from N_t as candidate line. We inject this result into Algorithm 3 (line 3) by looping only over the candidate lines, instead of all windows.

Results. We evaluate all these variants for all combinations of $k \in \{5,10,15\}$ and $\theta \in \{0.6,0.7,0.8\}$. The choice of these values is motivated by the insights of the result quality experiments suggesting that $k \leq 4$ does not yield good result quality. Furthermore, increasing k beyond 15 does not improve result quality. The analogous argumentation holds for selection of θ. As corpus C, we use the extended Wikipedia corpus, containing more than 2000 Wikipedia articles with more than 1 million words. We randomly select 20 articles as query documents. For each query document q, we measure the response time for comparing q to all documents in $C-\{q\}$, i.e., against more than 2,000 other documents. We report the average over all 20 queries in Table 10. We argue that this is the most realistic scenario one expects in real-world setting where one compares one article against a corpus of documents.

Table 10: Improvement of individual optimizations (in seconds). Avg. for $(Q,\mathcal{D})$ over 20 queries.

(k,θ)	Naive	k^2	zc-k^2	SeDA-1	SeDA	c-SeDA
(5,0.6)	250	16.9	13.9	6.4	3.0	0.5
(5,0.7)	250	16.9	13.9	5.9	2.8	0.3
(5,0.8)	250	16.9	13.7	5.8	2.4	0.3
(10,0.6)	514	49.6	37.9	10.5	4.8	2.1
(10,0.7)	514	49.4	37.9	9.4	3.9	1.4
(10,0.8)	514	49.3	37.6	8.6	3.6	1.0
(15,0.6)	1138	83.6	60.4	12.7	7.4	4.6
(15,0.7)	1138	82.1	60.3	11.9	5.5	2.9
(15,0.8)	1138	83.5	60.3	10.6	4.8	2.2

The study results are shown in Table 10. The results suggest two primary insights: First, we observe a clear improvement when subsequently adding components to SeDA. The improvement is robust towards changes of k and θ and significant, as SeDA and c-SeDA finishes on average below 10 seconds. Secondly, as expected due to the cubic complexity of the Hungarian algorithm, the window width k appears to have a larger impact on the response time than θ. The offline indexing phase takes about 5 minutes, mainly for computing N.

7 RELATED WORK

Finding similar passages of documents and sequences is a well-studied problem with numerous applications, including plagiarism detection [14, 33], DNA sequence analysis [6], or music copyright protection [40]. Most related to our problem definition are approaches that detect near-duplicate text passages (i.e., plagiarism), which is a postprocessing step of our alignment matrix A. Typically, existing approaches first detect suspicious candidate passages that are then semantically analyzed [15, 17].

Syntactic Candidate Determination. Many early approaches for candidate passage detection reduce the computational effort by relying on very small text passages creating a signature per document. Then, candidates are documents with similar signatures w.r.t. string edit distance, Jaccard, or Hamming distance [26]. To reduce the cost of candidate determination, various approaches approximate the similarity of signatures. Following the introduction of min-hashes. i.e., a locality sensitive hashing scheme [3], various approaches improve the original approach by combining multiple min-hashes into one (smaller) sketch [7, 13, 23, 30, 31, 37]. Independent of the signature scheme, all mentioned approaches miss semantically similar documents that are syntactically dissimilar.

Semantic Analysis. Existing works from natural language processing often compute passage similarity using sentence embeddings, e.g., variants and successors of BERT [10]. For large document corpora, this implies millions of inferences in a large neural network, which incurs high computation cost [27] as shown in our experiments. Therefore, efficient candidate filtering is applied to circumvent this problem. A common approach is concatenating or averaging the word vectors [19]. Similar to our problem definition, each token is associated with semantic information (i.e., a word vector [21]). Hence, one computes $sim(s_i,t_j)$ for every token pair in $S \times T$ and normalizes the result by the size of S and T. Compared to our reduction to the assignment problem, this approach does not yield the one-to-one matching property of tokens.

8 CONCLUSIONS

We presented the novel semantic document alignment problem with the objective to efficiently find semantically similar passages of documents based on matching tokens of k-width window pairs. The key novelty is to consider semantic information already in the matching phase and not only in the subsequent semantic analysis. Existing matching approaches are limited to syntactic information, missing potential matches. To mitigate the problem, low similarity thresholds must be used, resulting in many false positive matches. By modeling the similarity of k-width window pairs as assignment problem on bipartite graphs, we further introduce the one-to-one matching property of tokens. This is another novelty that has not been used in semantic analysis approaches before. Empirical results, e.g., on a well-established plagiarism detection benchmark reveal a clear superiority over Jaccard-based syntactic alignments with an F1-Score w.r.t. precision and recall of 0.93 and a result quality comparable to the award-winning plagiarism tool SEF. We propose SeDA an efficient algorithms for computing the semantic document alignment. SeDA effectively leverages a candidate generation mechanism, a bound cascade, and exploits the sliding window property. Compared to a syntactic state-of-the-art competitor, SeDA achieves comparable or superior run times on three real-world text corpora.

ACKNOWLEDGMENTS

We thank Aditi Dudeja for fruitful discussion on matching algorithms. This research was funded in whole or in part by the State of Salzburg, Austria: ref. no. 20204-WISS/262/9-2021, by the Austrian Science Fund (FWF) 10.55776/P34962, and by the Database Research Cluster at the University of Salzburg, core facility (CF) 3914 of the Austrian Federal Ministry for Women, Science, and Research (https://forschungsinfrastruktur.bmfwf.gv.at/en/fi/_3914).

REFERENCES

[1] Anonymous. 2023. Wikipedia Plaintext Articles (2023-07-01). https://www.kaggle.com/datasets/jjinho/wikipedia-20230701.

[2] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *International Conference on World Wide Web (WWW)*. ACL, 131–140.

[3] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. 1997. Syntactic Clustering of the Web. *Comput. Networks* 29, 8-13 (1997), 1157–1166.

[4] Dionisio Candido. 2023. *Synopsis of the Book of Esther: Masoretic Text, Old Greek, Alpha Text, Old Latin, Vulgate, Jewish Antiquities.* Peeters Publishers.

[5] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. *CoRR* abs/2402.03216 (2024).

[6] International Human Genome Sequencing Consortium. 2001. Initial sequencing and analysis of the human genome. *Nature* 409 (2001), 860–921.

[7] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. 2017. Fast Similarity Sketching. In *Annual Symposium on Foundations of Computer Science (FOCS)*, Chris Umans (Ed.). IEEE, 663–671.

[8] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SilkMoth: An Efficient Method for Finding Related Sets with Maximum Matching Constraints. *PVLDB* 10, 10 (2017), 1082–1093.

[9] Dong Deng, Yufei Tao, and Guoliang Li. 2018. Overlap Set Similarity Joins with Theoretical Guarantees. In *International Conference on Management of Data (SIGMOD)*. ACM, 905–920.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. ACL, 4171–4186.

[11] Jack R. Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* 19, 2 (1972), 248–264.

[12] ERF Medien e.V. 2025. Bibleserver. https://www.bibleserver.com/.

[13] Weiqi Feng and Dong Deng. 2021. Allign: Aligning All-Pair Near-Duplicate Passages in Long Texts. In *International Conference on Management of Data (SIGMOD)*. ACM, 541–553.

[14] Jérémy Ferrero, Frédéric Agnès, Laurent Besacier, and Didier Schwab. 2016. A Multilingual, Multi-style and Multi-granularity Dataset for Cross-language Textual Similarity Detection. In *International Conference on Language Resources and Evaluation (LREC)*. ELRA, 4162–4169.

[15] Tomáš Foltýnek, Norman Meuschke, and Bela Gipp. 2019. Academic Plagiarism Detection: A Systematic Literature Review. *ACM Comput. Surv.* 52, 6 (2019).

[16] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning Word Vectors for 157 Languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[17] Matthias Hagen, Martin Potthast, and Benno Stein. 2015. Source Retrieval for Plagiarism Detection from Large Web Corpora: Recent Approaches. In *Working Notes Papers of the CLEF 2015 Evaluation Labs, CEUR Workshop Proceedings*, Vol. 1391. CEUR-WS.org, 1531–1543.

[18] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759* (2016).

[19] Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Trans. Assoc. Comput. Linguistics* 3 (2015), 211–225.

[20] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. 2016. An empirical evaluation of set similarity join techniques. *Proc. VLDB Endow.* 9, 9 (2016), 636–647.

[21] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems (NIPS)*. 3111–3119.

[22] James Munkres. 1957. Algorithms for the Assignment and Transportation Problems. *J. Soc. Indust. Appl. Math.* 5, 1 (1957), 32–38.

[23] Zhencan Peng, Yuheng Zhang, and Dong Deng. 2024. Near-Duplicate Text Alignment with One Permutation Hashing. *Proc. ACM Manag. Data* 2, 4 (2024).

[24] Martin Potthast, Andreas Eiselt, Luis Alberto Barrón Cedeño, Benno Stein, and Paolo Rosso. 2011. Overview of the 3rd international competition on plagiarism detection. In *CEUR workshop proceedings*, Vol. 1177. CEUR Workshop Proceedings.

[25] Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. 2010. An Evaluation Framework for Plagiarism Detection. In *Coling 2010*. Coling 2010 Organizing Committee, 997–1005.

[26] Jianbin Qin, Chuan Xiao, Yaoshu Wang, Wei Wang, Xuemin Lin, Yoshiharu Ishikawa, and Guoren Wang. 2021. Generalizing the Pigeonhole Principle for Similarity Search in Hamming Space. *IEEE Trans. on Knowl. and Data Eng.* 33, 2 (2021), 489–505.

[27] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *CoRR* abs/1908.10084 (2019).

[28] Miguel A. Sanchez-Perez, Alexander Gelbukh, and Grigori Sidorov. 2015. Adaptive Algorithm for Plagiarism Detection: The Best-Performing Approach at PAN 2014 Text Alignment Competition. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Springer, 402–413.

[29] Daniel Schmitt, Daniel Kocher, Nikolaus Augsten, Willi Mann, and Alexander Miller. 2023. A Two-Level Signature Scheme for Stable Set Similarity Joins. *Proc. VLDB Endow.* 16, 11 (2023), 2686–2698.

[30] Anshumali Shrivastava. 2017. Optimal Densification for Fast and Accurate Minwise Hashing. In *International Conference on Machine Learning, (ICML)*, Vol. 70. PMLR, 3154–3163.

[31] Anshumali Shrivastava and Ping Li. 2014. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. In *International Conference on Machine Learning (ICML) (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, 557–565.

[32] T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197.

[33] Efstathios Stamatatos, Martin Potthast, Francisco Rangel, Paolo Rosso, and Benno Stein. 2015. Overview of the PAN/CLEF 2015 Evaluation Lab. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Springer, 518–538.

[34] Jiannan Wang, Guoliang Li, and Jianhua Fe. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *International Conference on Data Engineering (ICDE)*. 458–469.

[35] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending string similarity join to tolerant fuzzy token matching. *ACM Trans. Database Syst.* 39, 1 (2014), 7:1–7:45.

[36] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2017. Leveraging set relations in exact set similarity join. *Proc. VLDB Endow.* 10, 9 (2017), 925–936.

[37] Zhizhi Wang, Chaoji Zuo, and Dong Deng. 2022. TxtAlign: Efficient Near-Duplicate Text Alignment Search via Bottom-k Sketches for Plagiarism Detection. In *International Conference on Management of Data (SIGMOD)*. 1146–1159.

[38] Manuel Widmoser, Daniel Kocher, Nikolaus Augsten, and Willi Mann. 2023. MetricJoin: Leveraging Metric Properties for Robust Exact Set Similarity Joins. In *International Conference on Data Engineering (ICDE)*. 1045–1058.

[39] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* 36, 3 (2011).

[40] Yuchen Yuan, Charles Cronin, Daniel Müllensiefen, Shinya Fujii, and Patrick E. Savage. 2023. Perceptual and automated estimates of infringement in 40 music copyright cases. *Trans. Int. Soc. Music. Inf. Retr.* 6, 1 (2023).