International Journal of Computational Linguistics and Applications, Vol. 6, No. 1, 2015, pp. 149–172
Received 22/02/2015, Accepted 18/05/2015, Final 14/06/2015.
ISSN 0976-0962, http://ijcla.bahripublications.com

Computing Efficiently the Closeness of Word Sets in Natural Language Texts

DOMENICO CANTONE
SALVATORE CRISTOFARO
GIUSEPPE PAPPALARDO
Universita di Catania, Italy

ABSTRACT

We consider some search problems which have applications in statistical text analysis and natural language processing. Given two sets of words A *and* B, *we propose a statistical, corpus-based measure of the "closeness" between* A *and* B *in texts. Our proposed measure involves the search, throughout a text corpus, of the words in* A *and* B, *under the restriction that these words should co-occur within a given maximum distance n. We address the problem of efficiently computing this closeness measure and present algorithms for it.*

Keywords: Natural language processing, text search algorithms, closeness measure

1. INTRODUCTION

Discovering and retrieving relations between words is a central topic in computational linguistics, text analysis, and information retrieval. This is particularly true, for instance, in the case of the (semantic) similarity relation between words – *word similarity* – which has several important applications in natural language processing (see [1]). Various notions of word similarity have been proposed and studied over the years. Usually, word similarity relations are supplied with a measure – *a similarity measure* – that provides a statistical estimate of the degree of similarity among words, i.e., how much a word is related or similar to another word.

A conceptually simple case of similarity relation on words is provided by the following notion of *word closeness*: a word x is close to another word y if ("most of the times") x co-occurs with y within a given maximum distance n in a given collection of texts. This notion of word closeness is at the basis of various statistical measures of similarity that use the so-called corpus-based approach [2-5]. In the corpus-based model, similarity measures are computed using co-occurrence statistics of the words from a large corpus of collected texts in which the words occur. Note that this is different from the knowledge-based approach, which includes graph-based algorithms operating on lexical databases such as the WordNet [6-9].

A corpus-based measure for the above relation of word closeness can be easily provided by dividing the number of times that x co-occurs with y in the corpus within a distance of at most n, by the number of times that x occurs in the corpus. The higher is this ratio, the closer is x to y.

We generalize this relation of word closeness to sets of words (relative to a corpus); accordingly, a set A of words is close to another set B of words if the words of A co-occur with the words of B within a given maximum distance n in the corpus. We can define a statistical measure for this relation as follows. We assume that the corpus is provided as a finite ordered sequence (i.e., a string) of words. A *window* is a sequence of at most n consecutive words of the corpus. A window containing the set A is minimal if it cannot be shortened without skipping any word in A. Then, we count the number of minimal windows containing A that can be extended to windows containing also B, and divide this number by the number of minimal windows containing A. The resulting ratio is the *closeness factor of A to B* (for a formal statement, see *Problem* 1 in Section 3.) Again, the higher is the closeness factor, the closer is A to B.

The closeness factor of word sets has applications in the field of *text simplification* [10], a subfield of natural language processing concerned with the problem of reducing the syntactic complexity of a text, while retaining its information content and meaning. The basic idea here is that a text can be simplified by repeatedly removing from it one or more "semantically superfluous" words, i.e., words that do not affect its whole meaning. Note that simplifications of this kind can be used, for instance, to make it easier to analyze the meaning of texts in such contexts as *sentiment analysis* [11], concerned with the study of opinions expressed in text documents. Then, a possible approach to text simplification based on the closeness factor of word sets, could be the following one. Let us first introduce a useful notation to ease our presentation: for a set of words S, we denote by Sym (S) the set of words which are similar to the words in S, under a given relation of word similarity. Next, let T be the set of the words occurring in the text we intend to simplify. After choosing the first candidate word $w_1 \in T$ to be removed, we form the set $\mathrm{Sym}(T \setminus \{w_1\})$ to $\mathrm{Sym}(T)$; and compute the closeness factor of $\mathrm{Sym}(T \setminus \{w_1\})$ to $\mathrm{Sym}(T)$; if this is greater than or equal to some fixed threshold $t \leq 1$, we "safely" remove the word w_1 from T. We then select a second candidate word $w_2 \in T \setminus \{w_1\}$ to be removed from $T \setminus \{w_1\}$ and proceed as above. We keep removing words form T as long as the closeness factor remains above the threshold t. Let $w_1, w_2, \ldots, w_r$ be the list of the words that are removed during this process. Then, the text is simplified by eliminating from it the words of the set $\mathrm{Sym}(\{w_1, w_2, \ldots, w_r\})$. (See Example 1 at the end of Section 3.1.) The "simplification level", i.e., the number of words that are actually removed, depends on the particular candidates selected at each step. The goal is to obtain as high a simplification level as possible. Needless to say, trying all possible choices of the candidate words to be removed is far too expensive, since their number is exponential. Thus, it is essential to develop good selection heuristics, and make the choice of the candidates on the basis of them. In any case, since each step involves the calculation of a closeness factor, it is of prime importance to first design fast algorithms for performing such calculations efficiently.

In this paper we address the latter problem and present some efficient algorithms for computing the closeness factor of two sets of words.

The scenario sketched above for the simplification of a text reduces to the following optimization problem: given a set S of words and a threshold $t \leq 1$, determine a subsets S' of S, of minimum size, such that the closeness factor of S' to S is greater than or equal to t. However, here we will not be concerned with this problem, as it will be the topic of future investigations.

The paper is organized as follows. In Section 2 we introduce basic notations and terminology. Then, in Section 3, we formally define the main problem we are interested in, namely the computation of the closeness factor of two sets of words. In particular, we first develop efficient algorithms for two simpler problems related to it and then combine them into an efficient solution to our main problem (cf. Section 3.1). Finally, in Section 4, we draw our conclusions.

2. BASIC NOTATIONS

We will deal with strings of *words*, i.e., finite ordered sequences (possibly empty) of word occurrences. The *empty string* is denoted with ε and the length of a string σ is denoted with $|\sigma|$. Given a string σ and a word w, we write $\sigma.\mathrm{w}$ for the string obtained by adding the word w to the end of σ. Note that, for each word w, $\varepsilon.\mathrm{w}$ coincides with the string of length 1 consisting in a single occurrence of w. It is also convenient to assume the existence of a special word, the *null-word*, denoted with Λ.

Let σ be a string (of words). Given an index $0 \leq i < |\sigma|$, we denote with σ_i the $(i + 1)$st word of σ (from left to right); we extend this notation by also putting $\sigma_i =_{\mathrm{Def}} \Lambda$ for any $i < 0$ or $i \geq |\sigma|$. Given two integers h and k, we denote with $[h, k]$ the set (interval) of integers ℓ such that $h \leq l \leq k$. The *substring of σ between positions h and k*, where h and k are integers, is the string $\sigma_{[h,\, k]}$ defined recursively by

$$\sigma_{[h,k]} =_{Def} \begin{cases} \varepsilon, & \text{if } h > k \\ \sigma_{[h,k-1]} \cdot \sigma_k, & \text{otherwise.} \end{cases}$$

The set $\{\sigma_i : 0 \leq i < |\sigma|\}$ of the words occurring in σ is briefly denoted by $\langle\sigma\rangle$. Given two integers i and n, with $n \geq 1$, we call $\sigma_{[i,\ i+n-1]}$ the *window of σ of length n (starting) at position i.* The window $\sigma_{[i,i+n-1]}$ is a *minimal window containing a set S of words*, if $S \subseteq \langle\sigma_{[i,i+n-1]}\rangle$ but $S \nsubseteq \langle\sigma_{[h,k]}\rangle$, for all positions $i \leq h$, $k \leq i + n - 1$ such that the interval $[h, k]$ is strictly contained in the interval $[i,\ i + n - 1]$. A *text* is a nonempty string containing no occurrence of the null-word Λ, i.e., $\Lambda \notin \langle\tau\rangle$. The cardinality of a (finite) set S of words is denoted with $|S|$. In the sequel, by a set of words we shall always mean a nonempty set of words not containing the null-word Λ.

3. Formalizing the Problem: Some Preliminary Algorithms

In this section and in the subsequent one we address in details the problem of computing the closeness factor introduced in Section 1.

We begin with the following formal definitions.

Definition 1. *Let τ be a text, S a set of words, and n a window-length. For each integer i we put*:

$\mathcal{M}_{\tau,S,n}(i) =_{Def} \min(\{k \in [i-1,\ i+n-1] : S \cap \langle\tau_{[i,\,k]}\rangle = S \cap \langle\tau_{[i,\ i+n-1]}\rangle\})$
$K_{\tau,S,n}(i) =_{Def} |S \cap \langle\tau_{[i,\ i+n-1]}\rangle|$,
$\mathcal{E}_{\tau,S,n}(i) =_{Def} \max(\{k \in [i-n, +1, i]: S \subseteq \langle\tau_{[k,\,k+n-1]}\rangle\} \cup \{i-n\})$.

(*Thus if $S \subseteq \langle\tau_{[i,\ i+n-1]}\rangle$ and $\tau_i \in S \setminus \langle\tau_{[i,i+1,\ \mathcal{M}_{\tau,S,n(i)}]}\rangle$, the window $\tau_{[i,\,\mathcal{M}_{\tau,S,n(i)}]}$ is a minimal window containing S; moreover, $S \subseteq \langle\tau_{[i,\ i+n-1]}\rangle$ iff $K_{\tau,S,n(i)} = |S|$. In addition, for $j \in [i,\ i + n - 1]$, the window $\tau_{[i,\ j]}$ can be extended to a window of length n containing S, iff $j - \mathcal{E}_{\tau,S,n}(i) < n$.*)

Note that, for each $i \leq -n$, we have that $\mathcal{M}_{\tau,S,n}(i) = i - 1$, $\mathcal{K}_{\tau,S,n}(i) = 0$, and $\mathcal{E}_{\tau,S,n}(i) = i - n$; moreover, for $i > -n$, we have

$$\mathcal{E}_{\tau,S,n}(i) = \begin{cases} i, & \text{if } \mathcal{K}_{\tau,B,n}(i) = |B| \\ \max(\{n - i, \mathcal{E}_{\tau,B,n}(i-1)\}), & \text{otherwise.} \end{cases} \tag{1}$$

Also, observe that, for each integer i, we have that $\mathcal{M}_{\tau,S,n}(i) = \mathcal{M}_{\tau_{[0,i+n-1]},S,n}(i)$, $\mathcal{K}_{\tau,S,n}(i) = \mathcal{K}_{\tau_{[0,i+n-1]},S,n}(i)$, and $\mathcal{E}_{\tau,S,n}(i) = \mathcal{E}_{\tau_{[0,i+n-1]},S,n}(i)$.

Next, we state formally the main problem we are interested in.

Problem 1. Given a text τ,[1] two sets A and B of words, and a window-length n, compute the ratio

$$(A \ltimes B)_{\tau,n} =_{Def} \frac{\Omega^{+}_{\tau,A,n}(B)}{\Omega^{+}_{\tau,A,n}(B) + \Omega^{-}_{\tau,A,n}(B)},$$

where

$$\Omega^{+}_{\tau,A,n}(B) =_{Def} \Big| \{ i \in [0, |\tau| - 1] : \mathcal{K}_{\tau,A,n}(i) = |A|, \\ \tau_i \in A \setminus \langle \tau_{[i+1, \mathcal{M}_{\tau,A,n}(i)]} \rangle \\ \text{AND } (\mathcal{M}_{\tau,A,n}(i) - \mathcal{E}_{\tau,B,n}(i) < n) \} \Big|$$

and

$$\Omega^{-}_{\tau,A,n}(B) =_{Def} \Big| \{ i \in [0, |\tau| - 1] : \mathcal{K}_{\tau,A,n}(i) = |A|, \\ \tau_i \in A \setminus \langle \tau_{[i+1, \mathcal{M}_{\tau,A,n}(i)]} \rangle \\ \text{AND } (\mathcal{M}_{\tau,A,n}(i) - \mathcal{E}_{\tau,B,n}(i) \geq n) \} \Big| .$$

[1] Here τ stands for the corpus where the (co-)occurrences of the words of A and B are searched for (see the discussion in Section 1).

We call $(A \ltimes B)_{\tau,n}$ the *closeness factor of* A *to* B *relative to* τ *and* n.

Remark 1. Note that $\Omega^{+}_{\tau,A,n}(B)$ is equal to the number of the windows $\tau_{[i,i+m-1]}$ of τ, with $0 \leq i < |\tau|$ and $m \leq n$, such that (a) $\tau_{[i,i+m-1]}$ is a minimal window containing A, and (b) there exists $h \in [i+m-n, i]$ such that $B \subseteq \langle \tau_{[h,h+n-1]} \rangle$ (i.e., $\tau_{[i,i+m-1]}$ can be extended to a window of length n containing B, namely the window $\tau_{[h,h+n-1]}$). Moreover, $\Omega^{+}_{\tau,A,n}(B) + \Omega^{-}_{\tau,A,n}(B)$ is equal to the number of the minimal windows of τ containing A.

We will solve Problem 1 by considering separately the following two problems whose solutions can be easily combined to produce a solution to our main problem, as shown in Section 3.1.

Problem 2. Given a text τ, a set S of words, and a window-length n, determine the value $\mathcal{M}_{\tau,S,n}(i)$, for each position i of τ.

Problem 3. Given a text τ, a set S of words, and a window-length n, determine the value $\mathcal{K}_{\tau,S,n}(i)$, for each position i of τ.

Let us first consider Problem 2. We shall first present a basic algorithm for it, namely the algorithm Algo1 reported in Fig. 2; this will be subsequently refined into a more efficient variant named Algo2, reported in Fig. 3. We begin with the following observations.

Firstly, note that the value $\mathcal{M}_{\tau,S,n}(i)$ does not depend on the word τ_{i+n-1}, i.e., the last word of the window $\tau_{[i,i+n-1]}$, provided that such word is not contained in the set S. Thus, in particular, if $\tau_{i+n-1} \notin S$, we have that $\mathcal{M}_{\tau,S,n}(i) = \mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$.[2] Secondly, if $\tau_{i+n-1} \in S$, we can easily compute $\mathcal{M}_{\tau,S,n}(i)$ by using the value $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$ and the number $\mathcal{C}$ of occurrences of the word τ_{i+n-1} in the window $\tau_{[i,i+n-2]}$; indeed, if $\mathcal{C} = 0$ then $\tau_{[i,i+n-2]}$ contains no occurrences of τ_{i+n-1} and, in this case, we

[2] Since $\Lambda \notin S$, we have $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i) = \mathcal{M}_{\widetilde{\tau},S,n}(i)$, for $0 \leq i < |\tau| - n + 1$, where $\widetilde{\tau}$ is the string obtained by replacing the word in τ at position $i+n-1$ by the null-word Λ.

plainly have that $\mathcal{M}_{\tau,S,n}(i) = i + n - 1$; otherwise, if $\mathcal{C} \neq 0$, we plainly have that $\mathcal{M}_{\tau,S,n}(i) = \mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$. The number $\mathcal{C}$ can be easily expressed in terms of the number $\mathcal{C}'$ of occurrences of τ_{i+n-1} in the window $\tau_{[i-1,i+n-2]}$ by means of the following simple formula:

$$\mathcal{C} = \begin{cases} \mathcal{C}', & \text{if } \tau_{i-1} \neq \tau_{i+n-1} \\ \mathcal{C}' - 1, & \text{otherwise.} \end{cases}$$

Thus, the value of $\mathcal{M}_{\tau,S,n}(i)$ can be easily computed from the values $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$ and $\mathcal{C}'$.

Next, we turn our attention to the computation of the value $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$. We show how to compute it from $\mathcal{M}_{\tau,S,n}(i-1)$, using the positions of the occurrences of τ_{i-1} in the window $\tau_{[i-1,i+n-2]}$. Suppose first that $\tau_{i-1} \in S$. Let $\mathcal{L}$ be the set of the positions k of τ such that $\tau_k = \tau_{i-1}$, where $\mathcal{M}_{\tau,S,n}(i-1) < k < i + n - 1$.[3] Moreover, let $\mathcal{C}''$ be the number of occurrences of the word τ_{i-1} in the window $\tau_{[i-1,i+n-2]}$. Then $\mathcal{C}'' - 1 - |\mathcal{L}|$ is equal to the number of occurrences of the word τ_{i-1} which are strictly comprised between positions $i-1$ and $\mathcal{M}_{\tau,S,n}(i-1)$ of τ; i.e.,

$$\mathcal{C}'' - 1 - |\mathcal{L}| = \left|\{k \in [i, \mathcal{M}_{\tau,S,n}(i-1) - 1] : \tau_k = \tau_{i-1}\}\right| .$$

We observe the following facts. (A) If $\mathcal{C}'' - 1 - |\mathcal{L}| \neq 0$, there are occurrences of τ_{i-1} between positions $i-1$ and $\mathcal{M}_{\tau,S,n}(i-1)$ of τ, so that the value $\mathcal{M}_{\tau,S,n}(i-1)$ does not depend by any means on the word τ_{i-1}; hence, in this case, we have that $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i) = \mathcal{M}_{\tau,S,n}(i-1)$. Similarly, (B) if $\mathcal{L} = \emptyset$, then $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i) = \mathcal{M}_{\tau,S,n}(i-1)$ as well. When (C) $\mathcal{C}'' - 1 - |\mathcal{L}| = 0$ and $\mathcal{L} \neq \emptyset$ hold, it can be readily verified that, instead, $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i) = \min(\mathcal{L})$. (See Fig. 1 for a pictorial illustration of cases (A), (B), and (C) above.)

Finally, observe that if $\tau_{i-1} \notin S$, then $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i) = \max(\{i-1, \mathcal{M}_{\tau,S,n}(i-1)\})$.

[3] Plainly, since $\tau_{i-1} \in S$, we have $\mathcal{M}_{\tau,S,n}(i-1) \geq i-1$. Moreover, $\mathcal{M}_{\tau,S,n}(i-1) = i-1$ iff $S = \{\tau_{i-1}\}$, and if $\mathcal{M}_{\tau,S,n}(i-1) > i-1$, then $\tau_{\mathcal{M}_{\tau,S,n}(i-1)} \neq \tau_{i-1}$.

The previous observations can be stated formally as Lemmas 1 and 2 below. These will be expressed in terms of some maps to be introduced in Definition 2 below. In fact, rather than computing directly the values $\mathcal{M}_{\tau,S,n}(i)$, it is more convenient to compute the quantities $\mathcal{M}_{\tau,S,n}(i)$ defined below as this allows one to avoid dealing with the case $\tau_{i-1} \notin S$ examined above, yielding simpler calculations. Note in fact that, for each i, $\mathcal{M}_{\tau,S,n}(i) = \max\{\mathcal{M}_{\tau,S,n}(i), i-1\}$.

Definition 2. *For each text τ, set S of words, and window-length n, word w, we define the following maps:*

$$\mathcal{M}_{\tau,S,n}(i) =_{\mathit{Def}} \begin{cases} \mathcal{M}_{\tau,S,n}(i-1), & \textit{if } i > -n \text{ AND} \\ & \qquad \mathcal{M}_{\tau,S,n}(i) = i-1 \\ \mathcal{M}_{\tau,S,n}(i), & \textit{otherwise} \end{cases}$$

$$\mathcal{C}^{w}_{\tau,n}(i) =_{\mathit{Def}} |\{k \in [i, i+n-1] : \tau_k = w\}|$$

$$\mathcal{L}^{w}_{\tau,S,n}(i) =_{\mathit{Def}} \{k \in [i, i+n-1] : k > \mathcal{M}_{\tau,S,n}(i) \text{ AND } \tau_k = w\}\,.$$

Below we let τ be a fixed text, S be a fixed set of words, and n be a fixed window-length. In addition, to simplify the notations, for each integer i and word w, we shall write $\mathcal{M}(i)$, $\mathcal{C}^w(i)$, and $\mathcal{L}^w(i)$ for the values $\mathcal{M}_{\tau,S,n}(i)$, $\mathcal{C}^w_{\tau,n}(i)$, and $\mathcal{L}^w_{\tau,S,n}(i)$, respectively; also, we shall write $\widetilde{\mathcal{M}}(i)$, $\widetilde{\mathcal{L}}^w(i)$, and $\widetilde{\mathcal{C}}^w(i)$ for the values $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$, $\mathcal{L}^w_{\tau_{[0,i+n-2]},S,n}(i)$, and $\mathcal{C}^w_{\tau_{[0,i+n-2]},n}(i)$, respectively.[4]

We are now ready to state the two lemmas which summarize our previous discussion.

Lemma 1. *For each i, the following properties hold:*

(a) If $\tau_{i-1} \notin S$, then $\widetilde{\mathcal{M}}(i) = \mathcal{M}(i-1)$, $\widetilde{\mathcal{L}}^w(i) = \mathcal{L}^w(i-1)$, and $\widetilde{\mathcal{C}}^w(i) = \mathcal{C}^w(i-1)$, for each $w \in S$.

[4] Observe that, if $w \neq \Lambda$ and $0 \leq i < |\tau| - n + 1$, then $\mathcal{C}^w_{\tau_{[0,i+n-2]},n}(i) = \mathcal{C}^w_{\widetilde{\tau},n}(i)$, where $\widetilde{\tau}$ stands for the string obtained by replacing the word at position $i+n-1$ of τ by the null-word Λ; similarly for $\mathcal{L}^w_{\tau_{[0,i+n-2]},S,n}(i)$. (See also footnote 2.)

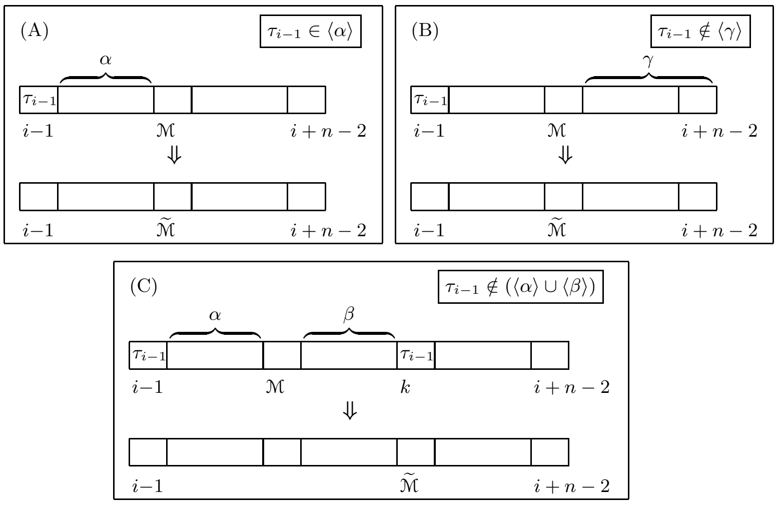
(A)
$\tau_{i-1} \in \langle\alpha\rangle$
α
τ_{i-1}
$i-1$
$\mathcal{M}$
$i+n-2$
$\Downarrow$
$\widetilde{\mathcal{M}}$
(B)
$\tau_{i-1} \notin \langle\gamma\rangle$
γ
(C)
$\tau_{i-1} \notin (\langle\alpha\rangle \cup \langle\beta\rangle)$
β
k

Fig. 1. The computation of $\mathcal{M}_{\tau_{[0,i+n-2]},S,n}(i)$ (denoted as $\widetilde{\mathcal{M}}$) from $\mathcal{M}_{\tau,S,n}(i-1)$ (denoted as $\mathcal{M}$), when $\tau_{i-1} \in S$. In the pictures, α, β and γ denote the windows $\tau_{[i,\mathcal{M}-1]}$, $\tau_{[\mathcal{M}+1,k-1]}$ and $\tau_{[\mathcal{M}+1,i+n-2]}$, respectively, with $k = \min(\mathcal{L})$. Cases (A) $\mathcal{C}'' - 1 - |\mathcal{L}| \neq 0$; (B) $\mathcal{L} = \emptyset$; and (C) $\mathcal{C}'' - 1 - |\mathcal{L}| = 0$, with $\mathcal{L} \neq \emptyset$. In both cases (A) and (B) we have $\widetilde{\mathcal{M}} = \mathcal{M}$, whereas in case (C) we have $\widetilde{\mathcal{M}} = \min(\mathcal{L})$.

(b) If $\tau_{i-1} \in S$, then
- *(b.1) $\widetilde{\mathcal{C}}^w(i) = \mathcal{C}^w(i-1)$, for each $w \in S \setminus \{\tau_{i-1}\}$;*
- *(b.2) $\widetilde{\mathcal{C}}^{\tau_{i-1}}(i) = \mathcal{C}^{\tau_{i-1}}(i-1) - 1$;*
- *(b.3) if $\mathcal{L}^{\tau_{i-1}}(i-1) \neq \emptyset$ and $\mathcal{C}^{\tau_{i-1}}(i-1)-1-|\mathcal{L}^{\tau_{i-1}}(i-1)| = 0$, then $\widetilde{\mathcal{M}}(i) = \min(\mathcal{L}^{\tau_{i-1}}(i-1))$ and $\widetilde{\mathcal{L}}^w(i) = \{j \in \mathcal{L}^{\tau_{i-1}}(i-1) : j > \widetilde{\mathcal{M}}(i)\}$, for each $w \in S$;*
- *(b.4) if $\mathcal{L}^{\tau_{i-1}}(i-1) = \emptyset$ or $\mathcal{C}^{\tau_{i-1}}(i-1)-1-|\mathcal{L}^{\tau_{i-1}}(i-1)| \neq 0$, then $\widetilde{\mathcal{M}}(i) = \mathcal{M}(i-1)$ and $\widetilde{\mathcal{L}}^w(i) = \mathcal{L}^w(i-1)$, for each $w \in S$.*

Lemma 2. *For each i, the following properties hold:*

(a) If $\tau_{i+n-1} \notin S$, then $\mathcal{M}(i) = \widetilde{\mathcal{M}}(i)$, $\mathcal{L}^w(i) = \widetilde{\mathcal{L}}^w(i)$, and $\mathcal{C}^w(i) = \widetilde{\mathcal{C}}^w(i)$, for each $w \in S$.
(b) If $\tau_{i+n-1} \in S$, then

(b.1) $\mathcal{C}^{\tau_{i+n-1}}(i) = \widetilde{\mathcal{C}}^{\tau_{i+n-1}}(i) + 1$*;*

(b.2) $\mathcal{C}^{w}(i) = \widetilde{\mathcal{C}}^{w}(i)$*, for each* $w \in S \setminus \{\tau_{i+n-1}\}$*;*

(b.3) if $\widetilde{\mathcal{C}}^{\tau_{i+n-1}}(i) = 0$*, then* $\mathcal{M}(i) = i+n-1$ *and* $\mathcal{L}^{w}(i) = \emptyset$*, for each* $w \in S$*;*

(b.4) if $\widetilde{\mathcal{C}}^{\tau_{i+n-1}}(i) \neq 0$*, then* $\mathcal{M}(i) = \widetilde{\mathcal{M}}(i)$*,* $\mathcal{L}^{w}(i) = \widetilde{\mathcal{L}}^{w}(i)$ *for each* $w \in S \setminus \{\tau_{i+n-1}\}$*, and also* $\mathcal{L}^{\tau_{i+n-1}}(i) = \widetilde{\mathcal{L}}^{\tau_{i+n-1}}(i) \cup \{i+n-1\}$*.*

Lemmas 1 and 2 readily lead to the basic algorithm Algo1 for Problem 2 reported in Fig. 2. Algorithm Algo1 scans the text τ from left to right. It moves iteratively a window of length n along τ, advancing it one position at a time. During iteration i, the items $\mathcal{M}(i)$, $\mathcal{L}^{w}(i)$, $\mathcal{C}^{w}(i)$, $\widetilde{\mathcal{M}}(i)$, $\widetilde{\mathcal{L}}^{w}(i)$, and $\widetilde{\mathcal{C}}^{w}(i)$, for $w \in S$, are computed by using information gathered during previous iterations, according to the properties in Lemmas 1 and 2. In more details, the algorithm Algo1 works as follows. It uses an array C indexed by the elements of S, such that, for each word $w \in S$, the entry $\mathsf{C}[w]$ maintains successively the counters $\mathcal{C}^{w}(i)$ and $\widetilde{\mathcal{C}}^{w}(i)$, for $-n \leq i < |\tau|$; more precisely, during iteration i, just before the execution of the conditional test of line 13, we have that $\mathsf{C}[w] = \widetilde{\mathcal{C}}^{w}(i)$, whereas, before the execution of the instruction of line 22 we have that $\mathsf{C}[w] = \mathcal{C}^{w}(i)$. The sets $\widetilde{\mathcal{L}}^{w}(i)$ and $\mathcal{L}^{w}(i)$ are maintained by means of an array Q of $|S|$ queues indexed by the elements of S. Specifically, for each word $w \in S$ and each i, during iteration i of Algo1, just before the conditional test in line 13 (resp., before the instruction of line 22), the queue $\mathsf{Q}[w]$ contains the elements of the set $\widetilde{\mathcal{L}}^{w}(i)$ (resp., $\mathcal{L}^{w}(i)$) increasingly ordered, with the smallest element at the head of the queue and the largest one at the tail. We assume that each queue $\mathsf{Q}[w]$ supports the following operations: (a) `dequeue`$(\mathsf{Q}[w])$, which removes the element at the head of $\mathsf{Q}[w]$; (b) `enqueue`$(\mathsf{Q}[w], j)$, which adds the number j at the tail of the queue $\mathsf{Q}[w]$; (c) `head`$(\mathsf{Q}[w])$, which returns (but not remove) the element at the head of $\mathsf{Q}[w]$; (d) `size`$(\mathsf{Q}[w])$, which returns the number of elements currently contained in the queue $\mathsf{Q}[w]$. The algorithm Algo1 also uses the integer variable M to store successively the values $\mathcal{M}(-n), \mathcal{M}(-n+1), \mathcal{M}(-n+2), \ldots$ as they are

```
Algo1($\tau$,$S$,$n$)
 1.  M $:= -n-1$
 2.  for $w \in S$ do
 3.     Q$[w] := \emptyset$
 4.     C$[w] := 0$
 5.  for $i := -n+1$ to $|\tau|-1$ do
 6.     if $\tau_{i-1} \in S$ then
 7.        C$[\tau_{i-1}]$ := C$[\tau_{i-1}] - 1$
 8.        if size(Q$[\tau_{i-1}]$) $> 0$ AND
                    (C$[\tau_{i-1}]$ $-$ size(Q$[\tau_{i-1}]$)) $= 0$ then
 9.           M := head(Q$[\tau_{i-1}]$)
10.           for $w \in S$ do
11.              while (size(Q$[w]$) $> 0$ AND
                                      head(Q$[w]$) $\leq$ M) do
12.                 dequeue(Q$[w]$)
13.     if $\tau_{i+n-1} \in S$ then
14.        C$[\tau_{i+n-1}]$ := C$[\tau_{i+n-1}] + 1$
15.        if C$[\tau_{i+n-1}] = 1$ then
16.           M $:= i+n-1$
17.           for $w \in S$ do
18.              while (size(Q$[w]$) $> 0$) do
19.                 dequeue(Q$[w]$)
20.        else
21.           enqueue(Q$[\tau_{i+n-1}]$, $i+n-1$)
22.     if M $< i-1$ then M $:= i-1$
23.     output(M) $\lhd$ outputs the value $\mathcal{M}_{\tau,S,n}(i)$
```

Fig. 2. The algorithm Algo1 for Problem 2

computed during the execution. Note that at iteration i of Algo1, just after the execution of instruction at line 9, the variable M stores the intermediate value $\widetilde{\mathcal{M}}(i)$, whereas, after the execution of instruction at line 16, M holds the value $\mathcal{M}(i)$.

Note that each queue of the array Q can be efficiently implemented as a circular array of size n in such a way that the supported operations `enqueue`, `dequeue`, `head`, and `size` take constant execution time each. Using such an implementation, the space complexity of the algorithm $\mathsf{Algo1}$ is readily seen to be $\mathcal{O}(|S| \cdot n)$. Concerning the time complexity of the algorithm $\mathsf{Algo1}$, we must first be explicit on the representation of the input set S. In fact, the particular representation used for S may affect, for instance, the implementation of the conditional tests in lines 6 and 13, and hence their computational costs. We shall assume that the set S is represented as a *map* $\mathbb{S} : S \cup \langle \tau \rangle \longrightarrow [-1, |S| - 1]$ such that

(a) $\mathbb{S}(w) = -1$ if and only if $w \notin S$, for every word $w \in S \cup \langle \tau \rangle$, and
(b) $\mathbb{S}(w') \neq \mathbb{S}(w'')$, for all distinct words $w', w'' \in S \cup \langle \tau \rangle$.

Thus, for each word $w \in S$, the value $\mathbb{S}(w)$ corresponds to the index of w in the array Q. By using this representation, the conditional tests in lines 6 and 13 of $\mathsf{Algo1}$ can just be expressed in the form $\mathbb{S}(\tau_i) \neq -1$ and $\mathbb{S}(\tau_{i+n-1}) \neq -1$, respectively. In the analysis below we shall assume that, for every word $w \in S \cup \langle \tau \rangle$, the value $\mathbb{S}(w)$ can be computed in constant time $\mathcal{O}(1)$,[5] so that the above two tests will take constant execution time. Thus, returning to the running time of $\mathsf{Algo1}$, note that, for each $w \in S$, the **while-loop**s of lines 11 and 18 are executed at most n times

[5] Such a requirement can be achieved as follows. Let $\boldsymbol{W}$ be a fixed set of words (the *vocabulary*), which includes all of the words we are dealing with, so that, in particular, $S \cup \langle \tau \rangle \subseteq \boldsymbol{W}$. We use a *hash function* h which injectively maps each word $w \in \boldsymbol{W}$ to its *hash code* $h(w)$, a nonnegative integer number. Then, we store the map $\mathbb{S}$ in a table T of size $\max\{h(w) : w \in \boldsymbol{W}\}$, in such a way that (a) $T[h(w)] = \mathbb{S}(w)$, for $w \in S$, and (b) $T[h(w)] = -1$, for $w \in \boldsymbol{W} \setminus S$. Thus, the calculation of $\mathbb{S}(w)$ reduces to the extraction of the value $T[h(w)]$, for each word w. Now, the function h can be chosen in such a way that the computation of $h(w)$ can be performed in $\mathcal{O}(|w|)$ time, for each word $w \in \boldsymbol{W}$. Hence, since the maximum length of words in $\boldsymbol{W}$ is fixed (and in fact, in practical cases this quantity is "small enough"), it turns out that the function h can be computed in constant time, yielding a constant execution time for computing $\mathbb{S}(w)$, for each word w. Note also that an additional integer variable could be used to hold the number of entries in T containing values different from -1. This would allow one to retrieve the cardinality of S in constant time.

since each queue $\mathsf{Q}[w]$ has size at most n, and so—under the above assumption concerning the representation of S—the running time of $\mathsf{Algo1}$ is easily seen to be $\mathcal{O}(|\tau| \cdot |S| \cdot n)$.[6] However, this is just a rough estimate. A slightly more accurate analysis leads to an $\mathcal{O}(|\tau| \cdot \max(|S|, n))$ time complexity. Indeed, note that, for each $w \in S$, the queue $\mathsf{Q}[w]$ contains at most all of the occurrences of w in the window $\tau_{[i,i+n-1]}$. Thus, the sum of the sizes of the queues in the array Q, i.e., $\sum_{w \in S} \mathtt{size}(\mathsf{Q}[w])$, is bounded above by n. Hence, the **for-loop**s of lines 10 and 17 are executed $\mathcal{O}(\max(|S|, n))$ times, and therefore the overall time complexity of $\mathsf{Algo1}$ is in fact $\mathcal{O}(|\tau| \cdot \max(|S|, n))$.

The basic algorithm $\mathsf{Algo1}$ presented above can be slightly modified to a $\mathcal{O}(|\tau|)$-time variant, still retaining the same space complexity. Note in fact that there is no need to update simultaneously all of the queues of the array Q during the **for-loop**s of lines 10 and 17; indeed, for each $w \in S$, we can safely postpone the updating of the queue $\mathsf{Q}[w]$ to the next time that w is encountered during the scan of τ, without affecting the correctness of the algorithm. This observation leads to the variant $\mathsf{Algo2}$ of $\mathsf{Algo1}$ reported in Fig. 3. As anticipated, the running time of $\mathsf{Algo2}$ is $\mathcal{O}(|\tau|)$. To see this, consider the behaviour of the queue $\mathsf{Q}[w]$, for an arbitrary word $w \in S$. Let $\mathsf{Occs}(w)$ be the number of occurrences of w in τ. During the execution of $\mathsf{Algo2}$, the number of enqueue operations involving $\mathsf{Q}[w]$ is no larger than the number of positions i such that $\tau_{i+n-1} = w$ and hence no larger than $\mathsf{Occs}(w)$. Thus the number of dequeue operations involving $\mathsf{Q}[w]$ is at most $\mathsf{Occs}(w)$. The total cost of the execution of the two **while-loop**s of lines 7 and 13 is therefore bounded above by the quantity $\sum_{w \in S} \mathsf{Occs}(w)$ which is at most equal to $|\tau|$. Thus, it plainly follows that the running time of $\mathsf{Algo2}$ is in fact $\mathcal{O}(|\tau|)$.

Next we turn to Problem 3. We can solve this problem by means of the following simple iterative procedure. As in the case of algorithms $\mathsf{Algo1}$ and $\mathsf{Algo2}$, a window of length n is iteratively

[6] All algorithms in the rest of the paper will implicitly use for the input set S the same representation by a map $\mathbb{S}$ as above.

```
Algo2($\tau$,$S$,$n$)
 1.  $\mathsf{M} := -n - 1$
 2.  for $w \in S$ do
 3.     $\mathsf{Q}[w] := \emptyset$
 4.     $\mathsf{C}[w] := 0$
 5.  for $i := -n + 1$ to $|\tau| - 1$ do
 6.     if $\tau_{i-1} \in S$ then
 7.        while ($\mathtt{size}(\mathsf{Q}[\tau_{i-1}]) > 0$ AND
                                    $\mathtt{head}(\mathsf{Q}[\tau_{i-1}]) \leq \mathsf{M}$) do
 8.           $\mathtt{dequeue}(\mathsf{Q}[\tau_{i-1}])$
 9.        $\mathsf{C}[\tau_{i-1}] := \mathsf{C}[\tau_{i-1}] - 1$
10.        if $\mathtt{size}(\mathsf{Q}[\tau_{i-1}]) > 0$ AND
                      $(\mathsf{C}[\tau_{i-1}] - \mathtt{size}(\mathsf{Q}[\tau_{i-1}])) = 0$ then
11.           $\mathsf{M} := \mathtt{head}(\mathsf{Q}[\tau_{i-1}])$
12.     if $\tau_{i+n-1} \in S$ then
13.        while ($\mathtt{size}(\mathsf{Q}[\tau_{i+n-1}]) > 0$ AND
                                    $\mathtt{head}(\mathsf{Q}[\tau_{i+n-1}]) \leq \mathsf{M}$) do
14.           $\mathtt{dequeue}(\mathsf{Q}[\tau_{i+n-1}])$
15.        $\mathsf{C}[\tau_{i+n-1}] := \mathsf{C}[\tau_{i+n-1}] + 1$
16.        if $\mathsf{C}[\tau_{i+n-1}] = 1$ then
17.           $\mathsf{M} := i + n - 1$
18.        else
19.           $\mathtt{enqueue}(\mathsf{Q}[\tau_{i+n-1}], i + n - 1)$
20.     if $\mathsf{M} < i - 1$ then $\mathsf{M} := i - 1$
21.     output($\mathsf{M}$) $\triangleleft$ outputs the value $\mathcal{M}_{\tau,S,n}(i)$
```

Fig. 3. The variant Algo2 of Algo1 for Problem 2

advanced along the text τ, moving it to the right one position at a time, starting from the initial position $i = -n$. We use an array R of length $|S|$, indexed by the elements of S, whose entry $\mathsf{R}[w]$ maintains successively the number of occurrences of the word w in the window starting at the current position i, for each word $w \in S$. An integer variable K is used to count the number of words of S that are contained in the window starting at position i. More formally, for each $i \geq -n$, when the iteration relative to position i is completed, we have:

(a) the entry $\mathsf{R}[w]$ of the array R contains the quantity $\mathcal{C}^{w}_{\tau,n}(i)$, for each word $w \in S$;
(b) the variable K holds the value $\mathcal{K}_{\tau,S,n}(i)$.

Note that initially, i.e., when $i = -n$, we have that: (i) $\mathsf{R}[w] = 0$, for each word $w \in S$; and (ii) $\mathsf{K} = 0$. Next we describe how the two items R and K are updated when moving from position $i-1$ to position i. Let R_{i-1} and K_{i-1} be the contents of R and K, respectively, just after position $i-1$ has been processed. The following four cases arise.

Case 1) $\tau_{i-1}, \tau_{i+n-1} \notin S$. Plainly, in this case no entry of the array R needs to be updated, nor the variable K needs to be changed.

Case 2) $\tau_{i-1} \in S$ and $\tau_{i+n-1} \notin S$. Since $\tau_{i-1} \in S$ and $\tau_{i-1} \neq \tau_{i+n-1}$, the number of occurrences of τ_{i-1} in the current window $\tau_{[i,i+n-1]}$ is equal to the number $\mathsf{R}_{i-1}[\tau_{i-1}]$ of occurrences of τ_{i-1} in the previous window $\tau_{[i-1,i+n-2]}$ decreased by 1. Thus the entry $\mathsf{R}[\tau_{i-1}]$ is updated to the value $\mathsf{R}_{i-1}[\tau_{i-1}] - 1$. Moreover, if $\mathsf{R}_{i-1}[\tau_{i-1}] = 1$, then the current window $\tau_{[i,i+n-1]}$ does not contain any occurrence of the word τ_{i-1}, whereas the previous window $\tau_{[i-1,i+n-2]}$ contains exactly one occurrence of τ_{i-1}, and therefore the variable K is updated to the value $\mathsf{K}_{i-1} - 1$. When $\mathsf{R}_{i-1}[\tau_{i-1}] \neq 1$, then, as in **Case 1)**, we do nothing.

Case 3) $\tau_{i-1} \notin S$ and $\tau_{i+n-1} \in S$. Since $\tau_{i+n-1} \in S$ and $\tau_{i-1} \neq \tau_{i+n-1}$, the number of occurrences of τ_{i+n-1} in the current window $\tau_{[i,i+n-1]}$ is equal to the number $\mathsf{R}_{i+n-1}[\tau_{i+n-1}]$ of occurrences of τ_{i+n-1} in the previous window $\tau_{[i-1,i+n-2]}$ increased by 1. Thus, the entry $\mathsf{R}[\tau_{i+n-1}]$ is updated to the value $\mathsf{R}_{i-1}[\tau_{i+n-1}]+1$. Moreover, if $\mathsf{R}_{i-1}[\tau_{i+n-1}] = 0$, then the current window $\tau_{[i,i+n-1]}$ contains exactly one occurrence of the word τ_{i+n-1}, whereas the previous window $\tau_{[i-1,i+n-2]}$ does

```
Algo3($\tau$,$S$,$n$)
 1.  K := 0
 2.  for $w \in S$ do
 3.     R[$w$] := 0
 4.  for $i := -n+1$ to $|\tau|-1$ do
 5.     if $\tau_{i-1} \in S$ then
 6.        R[$\tau_{i-1}$] := R[$\tau_{i-1}$] $- 1$
 7.        if R[$\tau_{i-1}$] $= 0$ then K := K $- 1$
 8.     if $\tau_{i+n-1} \in S$ then
 9.        R[$\tau_{i+n-1}$] := R[$\tau_{i+n-1}$] $+ 1$
10.        if R[$\tau_{i+n-1}$] $= 1$ then K := K $+ 1$
11.  output(K) $\lhd$ outputs the value $\mathcal{K}_{\tau,S,n}(i)$
```

Fig. 4. The algorithm Algo3 for Problem 3

not contain any occurrence of τ_{i+n-1}, and therefore the variable K is updated to the value $\mathsf{K}_{i-1}+1$. When $\mathsf{R}_{i-1}[\tau_{i+n-1}] \neq 0$, we do nothing, as in **Case 1)**.

Case 4) $\tau_{i-1}, \tau_{i+n-1} \in S$. In this case we simply perform the operations involved in **Case 2)**, followed by the operations involved in **Case 3)**.

The procedure described above translates into the algorithm Algo3 reported in Fig. 4.

Plainly, the space complexity of Algo3 is $\mathcal{O}(|S|)$; moreover, its running time can be readily seen to be $\mathcal{O}(|\tau|)$.

3.1 *Efficiently Solving the Main Problem*

Algorithms Algo2 and Algo3 can be combined into a single iterative algorithm that solves our main Problem 1 in time $\mathcal{O}(|\tau|)$ and space $\mathcal{O}(|A| \cdot n + |B|)$, where we recall that τ is the text, n is the window-length, and A and B are the sets of words to be searched for in τ. The complete pseudo-code of the resulting algorithm, named AlgoMain, is reported in Fig. 5–6.

```
AlgoMain($\tau$,$A$,$B$,$n$)
 1.  M $:= -n - 1$
 2.  for $w \in A$ do
 3.     Q$[w] := \emptyset$
 4.     C$[w] := 0$
 5.  A $:= 0$
 6.  K $:= 0$
 7.  for $w \in B$ do
 8.     R$[w] := 0$
 9.  E $:= -2n$
10.  $\Omega^+ := 0$
11.  $\Omega^- := 0$
12.  for $i := -n + 1$ to $|\tau|$ do
13.     if $\tau_{i-1} \in A$ then
14.        while (size(Q$[\tau_{i-1}]$) $> 0$ AND
                                  head(Q$[\tau_{i-1}]$) $\leq$ M) do
15.           dequeue(Q$[\tau_{i-1}]$)
16.        C$[\tau_{i-1}]$ := C$[\tau_{i-1}] - 1$
17.        if (C$[\tau_{i-1}]$ − size(Q$[\tau_{i-1}]$) $= 0$) AND
                                              A $= |A|$ then
18.           if (M − E $< n$) then $\Omega^+ := \Omega^+ + 1$
19.           else $\Omega^- := \Omega^- + 1$
20.        if C$[\tau_{i-1}] = 0$ then A := A − 1
21.        if size(Q$[\tau_{i-1}]$) $> 0$ AND
                        (C$[\tau_{i-1}]$ − size(Q$[\tau_{i-1}]$)) $= 0$ then
22.           M := head(Q$[\tau_{i-1}]$)
```

Fig. 5. The algorithm AlgoMain for Problem 1, part 1

Note that the block of instructions from line 1 to line 4 (resp., from line 6 to line 8) of algorithm AlgoMain corresponds to the initialization of algorithm Algo2 (resp., algorithm Algo3), whereas the instructions from line 13 to line 32, excluding lines 17, 18, 19, 20 and 28, correspond to the body of the main **for-loop** of algorithm Algo2. Also, observe that the instructions from line 33 to

```
23.     if $\tau_{i+n-1} \in A$ then
24.        while (size(Q[$\tau_{i+n-1}$]) $> 0$ AND
                              head(Q[$\tau_{i+n-1}$]) $\leq$ M) do
25.           dequeue(Q[$\tau_{i+n-1}$])
26.        C[$\tau_{i+n-1}$] := C[$\tau_{i+n-1}$] $+ 1$
27.        if C[$\tau_{i+n-1}$] $= 1$ then
28.           A := A $+ 1$
29.           M := $i + n - 1$
30.        else
31.           enqueue(Q[$\tau_{i+n-1}$], $i + n - 1$)
32.     if M $< i - 1$ then M := $i - 1$
33.     if $\tau_{i-1} \in B$ then
34.        R[$\tau_{i-1}$] := R[$\tau_{i-1}$] $- 1$
35.        if R[$\tau_{i-1}$] $= 0$ then K := K $- 1$
36.     if $\tau_{i+n-1} \in B$ then
37.        R[$\tau_{i+n-1}$] := R[$\tau_{i+n-1}$] $+ 1$
38.        if R[$\tau_{i+n-1}$] $= 1$ then K := K $+ 1$
39.     if K $= |B|$ then E := $i$
40.     else
41.        if (E $< n - i$) then E := $n - i$
42.  output($\frac{\Omega^+}{\Omega^+ + \Omega^-}$) $\lhd$ outputs the closeness factor
                                                  of $A$ to $B$
```

Fig. 6. The algorithm AlgoMain for Problem 1, part 2

line 38 of algorithm AlgoMain correspond to the body of the main **for-loop** of algorithm Algo3. The additional integer variables A and E are used to maintain successively the values $\mathcal{K}_{\tau,A,n}(i)$ and $\mathcal{E}_{\tau,B,n}(i)$, for $i \geq -n$, as they are computed during the execution of the algorithm; thus, at the end of iteration i of AlgoMain we have A $= \mathcal{K}_{\tau,A,n}(i)$ and E $= \mathcal{E}_{\tau,B,n}(i)$. Note that the variable E is successively updated according to relation (1) in Section 3. Finally, the two variables Ω^+ and Ω^- are used to hold the values $\Omega^+_{\tau,A,n}(B)$ and $\Omega^-_{\tau,A,n}(B)$; more precisely, at the end of the execution of the main **for-loop** of line 12 of AlgoMain, we have that $\Omega^+ = \Omega^+_{\tau,A,n}(B)$ and $\Omega^- = \Omega^-_{\tau,A,n}(B)$. In particular, concerning the calculation of the values $\Omega^+_{\tau,A,n}(B)$ and $\Omega^-_{\tau,A,n}(B)$, AlgoMain determines whether $\tau_{[i-1,\mathcal{M}_{\tau,A,n}(i-1)]}$ is a minimal window containing A, for $i = 1, 2, \ldots, |\tau|$, in the following way. At iteration i, initially AlgoMain checks whether $\tau_{i-1} \in A$ (see the instruction at line 13). If $\tau_{i-1} \notin A$, then, plainly, the window $\tau_{[i-1,\mathcal{M}_{\tau,A,n}(i-1)]}$ cannot be a minimal window containing A, and in fact, in this case, no update of the variables Ω^+ and Ω^- takes place. On the other hand, if $\tau_{i-1} \in A$ holds, AlgoMain successively updates the queue $\mathsf{Q}[\tau_{i-1}]$ and the entry $\mathsf{C}[\tau_{i-1}]$ by executing the **while-loop** at lines 14–15 and the instruction at line 16, respectively. In fact, immediately after the execution of the instruction at line 16, we have that $\mathsf{C}[\tau_{i-1}] - \mathtt{size}(\mathsf{Q}[\tau_{i-1}])$ is equal to the number of occurrences of the word τ_{i-1} in the window $\tau_{[i,\mathcal{M}_{\tau,A,n}(i-1)]}$, as can be readily verified. In addition, the variable A contains the number of words in A that occur in the window $\tau_{[i-1,i+n-2]}$. Therefore, $\tau_{[i-1,\mathcal{M}_{\tau,A,n}(i-1)]}$ is a minimal window containing A iff $\mathsf{C}[\tau_{i-1}] - \mathtt{size}(\mathsf{Q}[\tau_{i-1}])$ is equal to 0 and A is equal to $|A|$, which corresponds in fact to the conditional test at line 17.

By inspection, it is not difficult to verify that AlgoMain has an $\mathcal{O}(|A| \cdot n + |B|)$ space complexity; moreover, by using considerations similar to those made for algorithms Algo2 and Algo3, and assuming that the quantities $|A|$ and $|B|$ can be computed in constant time,[7] the running time of AlgoMain can be readily seen to be $\mathcal{O}(|\tau|)$.

We conclude the section with a simple example illustrating the idea, sketched in Section 1, for simplifying a text by means of the closeness factor of word sets.

Example 1. Let the words we are dealing with be represented by the following symbols:

$$\mathrm{A} \quad \mathrm{B} \quad \mathrm{C} \quad \mathrm{D} \quad \mathrm{E} \quad \mathrm{F}$$

[7] See footnote 5.

and let

$$\tau = \mathrm{DCCDAADEFFEABBCFD} \qquad \mathrm{EEFABCDCBDDCEAAEBCBC} \tag{2}$$

be the corpus and $\sigma = \mathrm{DACBDCB}$ be the text we intend to simplify. Moreover, let us assume that the maximum word distance n is 6 and that the threshold t is 0.7. Let $T = \langle\sigma\rangle = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$ be the set of the words occurring in σ. We start by selecting the first candidate word $w_1 = \mathrm{A}$ and try to remove it from T. This involves computing the closeness factor $(T \setminus \{w_1\} \ltimes T)_{\tau,n}$ and then comparing it with the threshold t. By executing the algorithm AlgoMain, we get $(T \setminus \{w_1\} \ltimes T)_{\tau,n} = 4/5$.[8] Since $4/5 > t$ we can safely remove the word w_1 from T thus obtaining the word set $T_1 = T \setminus \{w_1\} = \{\mathrm{B}, \mathrm{C}, \mathrm{D}\}$. Subsequently, we select the second candidate word $w_2 = \mathrm{C}$ to be removed from T_1 and compute the closeness factor $(T_1 \setminus \{w_2\} \ltimes T_1)_{\tau,n}$, obtaining $(T_1 \setminus \{w_2\} \ltimes T_1)_{\tau,n} = 4/5$.[9] As before, since this value is greater than the threshold t, we remove w_2 from T_1 and form the word set $T_2 = T_1 \setminus \{w_2\} = \{\mathrm{B}, \mathrm{D}\}$. At this point, it can be verified that, for every word $w \in T_2$, the closeness factor $(T_2 \setminus \{w\} \ltimes T_2)_{\tau,n}$ is less than t,[10] so that no further word can be removed and the process terminates. Hence, the words that have been removed are $w_1 = \mathrm{A}$ and $w_2 = \mathrm{C}$ which we then eliminate from σ, obtaining the simplified text DBDB. Note that, if at the second step we had selected the word $w_2 = \mathrm{B}$, rather than $w_2 = \mathrm{C}$, we would have obtained $(T_1 \setminus \{w_2\} \ltimes T_1)_{\tau,n} = 5/7$ which is less than the threshold t, and so we could not have removed this word from T_1. Similarly, the

[8] In fact, in the corpus τ there are exactly five minimal windows containing $T \setminus \{w_1\} = \{\mathrm{B}, \mathrm{C}, \mathrm{D}\}$, namely the windows starting at positions 13, 21, 23, 24 and 25; of these windows, only the one starting at position 24 cannot be extended to a window containing T. Thus $\Omega^{+}_{\tau,T\setminus\{w_1\},n}(T) = 4$ and $\Omega^{-}_{\tau,T\setminus\{w_1\},n}(T) = 1$ which yield $(T \setminus \{w_1\} \ltimes T)_{\tau,n} = 4/5$.

[9] In fact, it can be verified that $\Omega^{+}_{\tau,T_1\setminus\{w_2\},n}(T_1) = 4$ and $\Omega^{-}_{\tau,T_1\setminus\{w_2\},n}(T_1) = 1$.

[10] More precisely, we have that $(\{\mathrm{B}\} \ltimes \{\mathrm{B}, \mathrm{D}\})_{\tau,n} = (\{\mathrm{D}\} \ltimes \{\mathrm{B}, \mathrm{D}\})_{\tau,n} = 4/6$.

choice $w_2 = \mathrm{D}$ would have not led to any removal as well, since, in this case, we would have had $(T_1 \setminus \{w_2\} \ltimes T_1)_{\tau,n} = 3/7 < t$, as can easily be verified.

4 Conclusions

We have introduced a statistical, corpus-based measure of the closeness of two sets A and B of words in texts—the *closeness factor*—and we have presented an efficient algorithm for computing it. Our proposed algorithm involves a window-based search of the words of the sets A and B through a corpus τ, under the restriction that the words of the sets should co-occur within a given maximum distance n.

The algorithm runs in $\mathcal{O}(|\tau|)$-time, where $|\tau|$ is the length of τ, and uses $\mathcal{O}(|A| \cdot n + |B|)$-space, where $|A|$ and $|B|$ are the cardinalities of A and B, respectively. The closeness factor has applications in various subfields of natural language processing, among which text simplification and sentiment analysis.

We plan to investigate further problems involving the closeness factor, such as the optimization problem mentioned in Section 1.

References

1. Indurkhya, N., Damerau, F.J. (eds.) 2010. *Handbook of Natural Language Processing*. 2nd ed. CRC, Boca Raton, FL.
2. Radinsky, K., Agichtein, E., Gabrilovich, E., Markovitch, S. 2011. A word at a time: Computing word relatedness using temporal semantic analysis. In proceedings of the *20th International Conference on World Wide Web* (pp. 337-346), 11, New York, NY, USA, ACM.
3. Reisinger, J., Mooney, R. J. 2010. Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 109-117).
4. Lowe, W. 2001. Towards a theory of semantic space. In J. T. Moore, & K. Stenning (Eds.), *Proceedings of the 23rd Conference of the Cognitive Science Society* (pp. 576-581).

5. Lee, L. 1999. Measures of distributional similarity. In proceedings of the *37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, Morristown, NJ, USA, Association for Computational Linguistics (pp. 25-32).
6. Fellbaum, C. (ed) 1998. *WordNet: An Electronic Lexical Database*. MIT Press.
7. Budanitsky, A. & Hirst, G. 2006. Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32, 13-47.
8. Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Pasca, M., Soroa, A. 2009. A study on similarity and relatedness using distributional and WordNet-based approaches. In proceedings of *NAACL-HLT 09*.
9. Borz`ı, V., Faro, S. & Pavone, A. 2014. Automatic extraction of semantic relations by using web statistical information. In N. Hernandez, R. J¨aschke & M. Croitoru (Eds.), *Graph-Based Representation and Reasoning*. *Lecture Notes in Computer Science*. Springer International Publishing (pp. 174-187).
10. Siddharthan, A. 2006. Syntactic simplification and text cohesion. *Research on Language and Computation*, 4/1, 77-109
11. Pang, B. & Lee, L. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2/1-2, 1-135.

ACKNOWLEDGMENTS

We would like to thank the anonymous referee for his/her helpful comments. This work has been partially supported by the project PRISMA PON04a2 A (funded by the Italian Ministry of University and Research within the PON 2007-2013 "Smart cities and communities" framework) and by the FIR-2014 project "COMPACT: COMPuting reliably on AuthentiCated Texts" (funded by the University of Catania).

DOMENICO CANTONE
DIPARTIMENTO DI MATEMATICA E INFORMATICA
UNIVERSITA DI CATANIA, VIALE ANDREA DORIA N. 6,
I-95125 CATANIA, ITALY.

SALVATORE CRISTOFARO
DIPARTIMENTO DI MATEMATICA E INFORMATICA
UNIVERSITA DI CATANIA, VIALE ANDREA DORIA N. 6,
I-95125 CATANIA, ITALY.

GIUSEPPE PAPPALARDO
DIPARTIMENTO DI MATEMATICA E INFORMATICA
UNIVERSITA DI CATANIA, VIALE ANDREA DORIA N. 6,
I-95125 CATANIA, ITALY.