Cognitive Computation (2020) 12:811–833
https://doi.org/10.1007/s12559-019-09706-3

TeKET: a Tree-Based Unsupervised Keyphrase Extraction Technique

Gollam Rabby[1,2] · **Saiful Azad**[1,3] · **Mufti Mahmud**[4] · **Kamal Z. Zamli**[1,3] · **Mohammed Mostafizur Rahman**[5]

Received: 5 June 2019 / Accepted: 28 November 2019 / Published online: 5 March 2020

Abstract

Automatic keyphrase extraction techniques aim to extract quality keyphrases for higher level summarization of a document. Majority of the existing techniques are mainly domain-specific, which require application domain knowledge and employ higher order statistical methods, and computationally expensive and require large train data, which is rare for many applications. Overcoming these issues, this paper proposes a new unsupervised keyphrase extraction technique. The proposed unsupervised keyphrase extraction technique, named *TeKET* or *Tree-based Keyphrase Extraction Technique*, is a domain-independent technique that employs limited statistical knowledge and requires no train data. This technique also introduces a new variant of a binary tree, called *KeyPhrase Extraction* (*KePhEx*) tree, to extract final keyphrases from candidate keyphrases. In addition, a measure, called *Cohesiveness Index* or *CI*, is derived which denotes a given node's degree of cohesiveness with respect to the root. The CI is used in flexibly extracting final keyphrases from the KePhEx tree and is co-utilized in the ranking process. The effectiveness of the proposed technique and its domain and language independence are experimentally evaluated using available benchmark corpora, namely SemEval-2010 (a scientific articles dataset), Theses100 (a thesis dataset), and a *German Research Article* dataset, respectively. The acquired results are compared with other relevant unsupervised techniques belonging to both statistical and graph-based techniques. The obtained results demonstrate the improved performance of the proposed technique over other compared techniques in terms of precision, recall, and F1 scores.

Keywords Candidate keyphrase · Unsupervised machine learning · Automatic keyphrase extraction · Document processing · Recommender system · Binary tree

Introduction

Automatic keyphrase extraction techniques endeavor to extract quality keyphrases automatically from documents. Generally, these keyphrases provide a high-level summarization of the considered document. Therefore, they are utilized in many digital information processing applications, such as information retrieval [2, 78], digital content management [9, 68], natural language processing [28, 53], contextual advertisement [61, 77], recommender system [49, 54], and so on; which are portrayed in Fig. 1. Herein, the concept of information retrieval has been developed to extract desired information from a large collection of textual data. It has been implemented in many practical applications, such as search engines [66], media search [73], digital libraries [39], geographic information retrieval [24], legal information retrieval [12], and many more. It is inane explaining the necessity of these systems, since what data can we retrieve without these systems!

Again, keyphrases play an important role in content management [57]. They are utilized for document indexing [55] to describe or classify the semantic similarity among various documents (a.k.a., document clustering [59, 76] or document classification [41, 74]), and thereby, can be utilized as recommender systems to improve the browsing experience

✉ Saiful Azad
saifulazad@ump.edu.my

✉ Mufti Mahmud
mufti.mahmud@ntu.ac.uk; muftimahmud@gmail.com

1 Faculty of Computing, University Malaysia Pahang, 26300 Gambang, Kuantan, Malaysia

2 Present address: Department of Information and Knowledge Engineering, University of Economics, W. Churchill Sq. 4, 130 67 Prague 3, Czech Republic

3 IBM CoE, UMP, Gambang, Kuantan, Malaysia

4 Department of Computing & Technology, Nottingham Trent University, Clifton Lane, Nottingham, NG 11 8NS, UK

5 Department of Mathematics, American International University – Bangladesh, Dhaka, Bangladesh

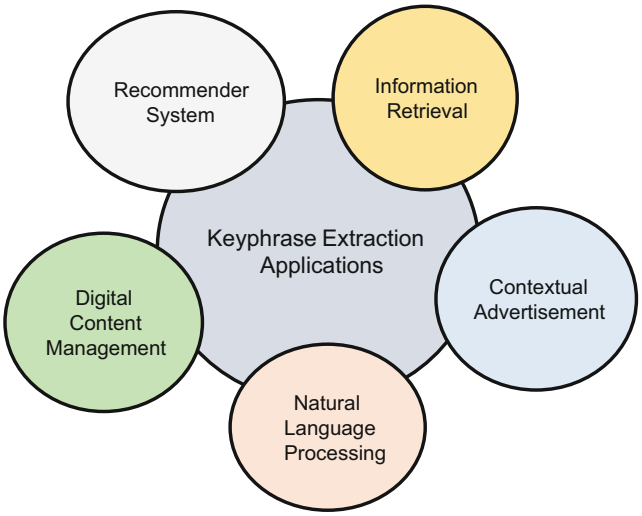

Fig. 1 Prominent applications of keyphrase extraction

of digital libraries. Furthermore, document classification and similar concepts are widely used in machine learning (ML), data mining, database discovery, and so on. Some notable applications using these techniques are newsgroup filtering, target marketing, document organization, health status tracking, and so on [1, 14, 20, 35, 37, 42]. In addition, for any contextual advertising to display advertisements based on user identity and browsing history, keyphrase extraction is a core technique.

To support these aforementioned applications, several keyphrase extraction techniques have been proposed [15–17, 22, 26, 29, 39, 40, 58, 63, 67, 79]. Among them, domain-specific approaches [21] require knowledge of the application domain, and linguistic approaches [65] require expertise of the language, thus are inapplicable in problems from other domains and/or languages. Among the ML-based techniques [25, 36], supervised ML techniques demand a considerable amount of rare train data to extract quality keyphrases. Again, statistical unsupervised techniques [3, 11, 17] are computationally expensive due to their large amount of complex operations, and graph-based unsupervised techniques [6, 7, 19, 60, 70] perform poorly due to their incapability in identifying cohesiveness among various words that form a keyphrase [25]. In light of the aforementioned discussion, the automatic keyphrase extraction remains an important research area to explore.

Hence, this paper proposes a new automatic keyphrase extraction technique with the following notable contributions:

- A domain- and language-independent unsupervised keyphrase extraction technique, named tree-based keyphrase extraction technique (TeKET) that employs limited statistical knowledge and requires no train data.
- A variant of the binary tree, called *Keyphrase Extraction* (*KePhEx*) tree, which extracts final keyphrases from candidate keyphrases.
- A new keyphrase ranking approach employing *Cohesiveness Index* (*CI* or μ) value and *Term Frequency (TF)* as calculating factors.
- Determine effective values for various parameters, which have a direct influence on the performance of the proposed technique in different application domains.

The other sections of this paper are organized as follows. The "Related Works" section lists various prominent keyphrase extraction techniques with their advantages and limitations, and thus, demonstrates the necessity of proposing a new technique. Afterwards, the "Preliminaries" section describes the preliminaries, which includes problem formulation and conceptual framework of the proposed technique. The proposed technique is elaborated in detail in the "Methods" section. The "Experimental Setup" section elaborates on the setup of the experiments, which includes corpus details, evaluation metrics, and implementation details. All the acquired results are plotted and analyzed in the "Results and Discussion" section and are concluded in the "Conclusions" section.

Related Works

Since our proposed technique is an unsupervised keyphrase extraction technique, therefore, this section only discusses similar approaches. Again, as seen in Fig. 2, most of the unsupervised keyphrase extraction techniques could be broadly classified into two groups, namely graph-based and statistical techniques. Prominent approaches of both these groups are scrutinized below.

Graph-Based Techniques

Here, the core idea is to build a graph from an input document and to rank its nodes according to their importance [8]. For instance, KeyGraph [45] is a similar technique which is content sensitive and domain-independent and utilizes co-occurrence of various terms for indexing vertices of the graph. However, it fails to detect the relationships among the low-frequency items inside clusters and also ignores direct relationships between the clusters [71]. On the other hand, PageRank [46] is based on the concept of random walks and is related to eigenvector centrality that tends to favor nodes with many important connections regardless of cohesiveness considerations. This technique is well suited for raking pages on the web and social networks, but not suitable for keyphrase extraction due to lack of consideration of cohesiveness [44, 72].

An extension of PageRank is PositionRank [19], which incorporates all the positions of a word along with its frequency to score the word, and thus, decides the rank

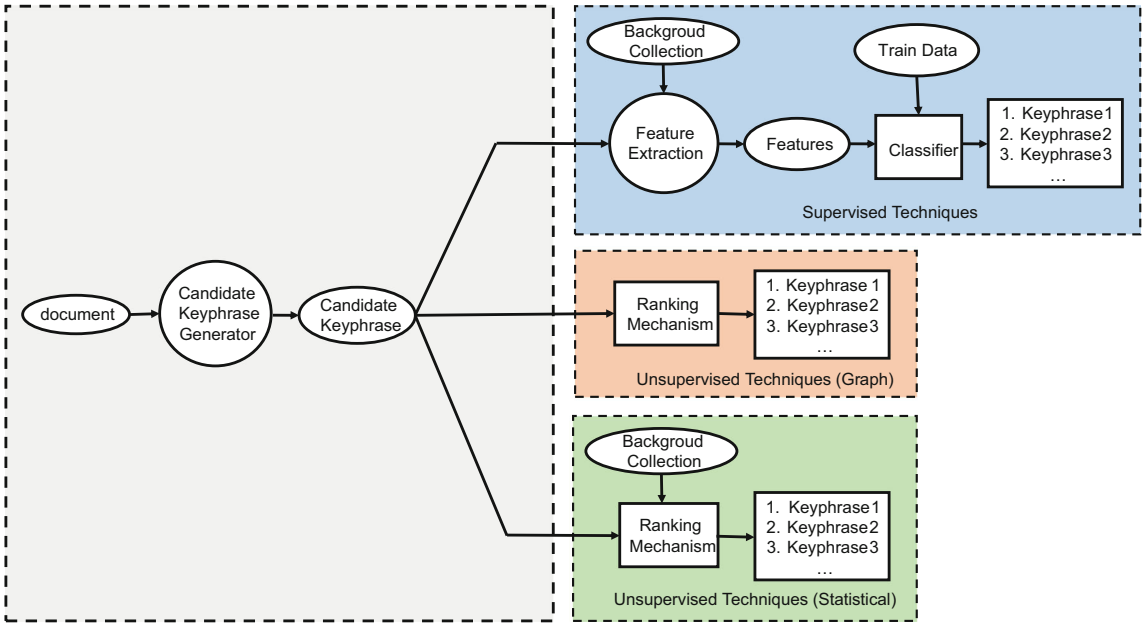

Fig. 2 Functional details of various machine learning–based technique for keyphrase extraction

of that particular word. This way, it outperforms all the techniques that consider only the first position information in the ranking. However, due to ignoring topical coverage and diversity which is not naturally handled by this kind of graphs [25], this technique suffers from considerably limited performance.

TextRank [44] is one of the most well-known graph-based approaches for keyphrase extraction. Here, the scientific documents are modeled as undirected or directed and weighted co-occurrence networks using a co-occurrence window of variable sizes [44]. It experiences several limitations, such as its incapability to capture cohesiveness. Again, retaining only the main core is suboptimal since sometimes it is impractical to discover all the gold standard keyphrases within a unique subgraph, whereas many valuable keyphases may place in the lower levels of the hierarchy [64]. Moreover, selecting or discarding a large group of words at a time reduces the flexibility of the extraction process and negatively impacts the performance. An extension of TextRank is SingleRank [70], which weights an edge equal to the number of times the two corresponding words co-occur. Unlike its predecessor, it does not extract keyphrases by assembling ranked words, instead, only noun phrases are extracted from a document. However, sometimes it assigns higher scores to long but non-significant keyphrases which entices the ranking procedure.

Another enhancement of TextRank is TopicRank [7]. Here, the vertices of a graph are topics, not words. It extracts the noun phrases that represent the main topics of a document and clustered them into topics. A notable advantage of this technique is that it considers topical coverage and diversity. However, it equally weighs all candidates belonging to a single topic, which is impractical. In addition, it suffers from the error propagation problem which may occur during topics formation. To resolve the error propagation problem of TopicRank, the MultipartiteRank technique [6] utilizes a multipartite graph. Here, a complete directed multipartite graph is built that is connected only if they belong to different topics. Since this technique makes good use of relation reinforcement between topics and candidates, it performs better than other graph-based techniques. However, due to clustering error (where candidate keyphrases could be wrongly assigned to a similar topic), it struggles in selecting the most representative candidates.

Statistical Techniques

Although graph-based techniques show acceptable performance on many occasions, they are considerably difficult to implement in comparison with statistical unsupervised keyphrase extraction techniques. Three such prominent techniques are scrutinized below. The most prominent and state-of-the-art statistical technique is Term Frequency - Inverse Document Frequency (TF-IDF) [56], which reflects the importance of a keyphrase to a document in a corpus. Among the two terms, *TF* provides aboutness and *IDF* provides informativeness. In other words, the *IDF* discriminates between informative and non-informative keyphrases across the documents, whereas the *TF* discriminates between popular and non-popular keyphrases in a document. This technique is computationally expensive as

IDF is calculated across different documents [48]. Again, many studies report that this technique is biased towards single terms over compound terms [18].

To resolve the problem of favoring single terms, KP-Miner [18] is proposed. It utilizes some heuristics based on *TF* and positions to identify potential keyphrases which are weighted with *TF-IDF* score [30]. Although it outperforms *TF-IDF*, it experiences several limitations such as a drop in global ranking performance with increasing length or number of documents [43]. In addition, it is computationally expensive due to its dependence on *TF-IDF*.

Another lightweight technique is YAKE [10], which resolves the *IDF* problem. It takes five features into consideration, namely casing, word position, word frequency, word relatedness to context, and word in the different sentences to calculate the weight of a keyphrase. Again, due to generating candidate keyphrases employing N-grams technique, its computational complexity increases linearly with respect to N-grams [75]. Again, due to the same reason, a large number of keyprhases are generated, which entices the ranking procedure.

From the above discussions, it is evident that graph-based techniques and statistical techniques have several adverse characteristics, which restrict them from achieving better performance. To overcome the identified shortcomings, this paper proposes a tree-based technique to extract quality keyphrases from documents.

Preliminaries

This section formulates the problems of keyphrase extractions followed by explaining the conceptual framework that are taken into account while developing the proposed technique.

Problem Formulation

Consider a document, δ, which has been passed to a keyphrase extraction technique to extract the final keyphrases, φ. For this, at first candidate keyphrases, χ are extracted from δ, which will be processed later to extract φ. Any candidate keyphrase χ_i in χ (i.e., $\chi_i \in \chi$) is composed of n number of ordered sequence of words, $\{w_1, w_2, ..., w_m, ..., w_{n-1}, w_n\}$, where n is a positive integer number, i.e., $n \in \mathbb{Z}+$. Since any keyphrase is a coherently connected sequence of words that appear contiguously, χ_i could be represented as an ordered set and its segments also could be represented as ordered subsets. Again, when $n = 1$, χ_i contains only one word, otherwise multiple words. Note that χ_i cannot be empty, and therefore, $|\chi_i| = n \neq 0$.

For extracting a final keyphrase, φ_j (where $\varphi_j \in \varphi$) from a χ_i, the latter is necessary to be processed. For this, the following probable cases need to be considered:

Case 1 : χ_i is φ_j, i.e., $\chi_i = \varphi_j$ or $\chi_i \subseteq \varphi_j$ and $\varphi_j \subseteq \chi_i$.
Case 2 : φ_j is a part of χ_i, i.e., $\varphi_j \subset \chi_i$.
Case 3 : Again, χ_i is a part of φ_j, i.e., $\chi_i \subset \varphi_j$.
Case 4 : χ_i is not a final keyphrase.

Although four probable cases are identified, it is difficult to determine an exact case for a certain candidate keyphrase. To identify that, in the subsequent section, we discuss some hypotheses and observations.

Conceptual Framework

The concept of extracting final keyphrases from candidate keyphrases relies on the following hypotheses and observations:

Hypothesis 1 : For any χ_i, *case 1* and *case 4* can be determined by its popularity. In other words, this decision can be taken based on the frequency of χ_i in a document and applying a binary decision strategy.
Hypothesis 2 : For *case 2*, since a part of χ_i—denoted as χ_i'—is a final keyphrase (i.e., $\chi_i' = \varphi_j$), the popularity and the cohesiveness of χ_i' must be higher than that of χ_i. In this case, χ_i need to be appropriately reduced to χ_i'.
Hypothesis 3 : For *case 3*, since χ_i is a part of φ_j, χ_i need to be expanded to χ_i' such that $\chi_i' = \varphi_j$. Again in this case, the popularity and the cohesiveness of χ_i' must be higher than that of χ_i.

Hypothesis 1 is quite straightforward. A simple binary decision strategy could be applied to determine this. For instance, assuming that the frequency of χ_i is α in δ. Now, it is compared with λ, which is a constant value, and also known as *least seen allowable frequency* (*lsaf*) factor [17]. It separates non-popular keyphrases from popular keyphrases, which are unlikely to become final keyphrases. For instance, when $\alpha < \lambda$, it is most likely not a final keyphrase; otherwise, it is likely to be a final keyphrase. Note that the value of λ varies from one language to another and also is subjected to the length of a document [17]. Hence, an experiment has been conducted to find a suitable *lsaf* value (see the "Parameter Value Selection" section). Again, for *hypothesis 2* and *hypothesis 3*, the proposed rooted binary tree expands or shrinks based on the candidate keyphrases and keeps track of the

cohesiveness of various words in a keyphrase with respect to the root. In the end, the final keyphrases are extracted from the tree as detailed in the subsequent section.

Methods

The entire process of keyphrase extraction using our proposed technique can be parted into three main phases: (i) candidate keyphrase selection or pre-processing, (ii) candidate keyphrase processing or simply processing, and (iii) ranking and selecting final keyphrases or post-processing (see Fig. 3).

Candidate Keyphrase Selection

The proposed technique employs the *Part-Of-Speech* (POS) *Tagging* (POST) approach to extract candidate keyphrases from δ. Since keyphrases are generally noun phrases [13], the proposed technique limits the extraction to only *noun phrases* [13]. For this, the following *POS* pattern is utilized, which has been demonstrated in [52] as one of the most suitable patterns for extracting candidate keyphrases.

$$(< NN.* > + < JJ.* >?)|(< JJ.* >? < NN.* > +)$$

Note that it is a regular expression that is written in a simplified format using NLTK's RegexpParser, where nouns are tagged with NN and adjectives are tagged with JJ. More details could be found in [23].

Once the candidate keyphrases are extracted, they are passed through a cleaning process to filter out those keyphrases that are less likely to be final keyphrases. For that, following conditions are applied: (i) any candidate keyphrase that contains non-alphabetic characters, (ii) any candidate keyphrase that contains single alphabetic word(s), and (iii) if the frequency of any candidate keyphrase fails to satisfy *lsaf* factor (see the "Conceptual Framework" section). The first two conditions filter out candidate keyphrases that make no sense to the human reader in general; and the latter one filters out all non-popular candidate keyphrases from the list.

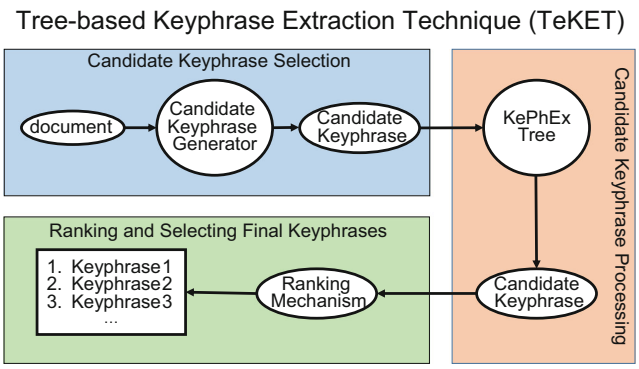

Fig. 3 Functional details of the proposed technique

Candidate Keyphrase Processing Using KeyPhrase Extraction (KePhEx) Tree

In conventional unsupervised keyphrase extraction techniques, candidate keyprhases are not processed; instead, they are sent to the ranking phase immediately after the selection. On the contrary, an intermediate phase between candidate keyphase selection and ranking could release the burden of ranking unnecessary keyphrases, and thus, lead to finding more appropriate keyphrases. The proposed *KePhEx* tree takes all the formerly mentioned hypotheses (see the "Preliminaries" section) into account for extracting final keyphrases. The *KePhEx* tree expands (*hypothesis 3*) or shrinks (*hypothesis 2*) or remains in the same state (*hypothesis 1*) based on the candidate keyphrases. The advantages of employing *KePhEx* tree in keyphrase extraction are threefold: (i) extracts quality keyphases from candidate keyphrases, (ii) provides flexibility during keyphrase extraction, and (iii) contributes in ranking by providing a value that represents cohesiveness of a word in a keyphrase with respect to a root.

Among different classes of tree data structure, the *KePhEx* tree falls under a binary tree. Again, although there exist several variants of a binary tree, it is different from others since the position of every node in the tree and its level are fixed. Again, all the predecessors of a node at the upper-levels (including root) are also fixed unlike other variants. It is so because a good keyphrase must be a coherently connected sequence of words that appear continuously in the text. Every node in a *KePhEx* tree holds a 2-tuple data along with other information, namely a word and its CI or μ value. The CI provides two advantages: (i) assists in finding the cohesiveness of various words with respect to the root of the tree, which is employed as a factor in ranking keyphrases and (ii) provides flexibility during keyphrase extraction as the value of μ increases or decreases based on the existence of that word in candidate keyphrases.

Root Selection

It is important to select a qualified root since a poorly selected root may lead to a poor keyphrase. In this technique, only nouns are designated as roots, which are selected from the candidate keyphase list, χ, and are saved

in another list, η. As noun phrases are the most likely candidate for final keyphrases, selecting them (i.e., nouns) as roots increases the chances of extracting quality final keyphrases.

After selecting the roots, the trees are formed taking these roots into consideration. The entire process from tree formation to final keyphrase extraction is segmented into three main steps, namely (i) tree formation, (ii) tree processing, and (iii) keyphrase extraction.

Tree Formation

For forming a *KePhEx* tree, a root, γ, is selected from η. Afterwards, the proposed system selects candidate keyphrases that contains γ. Let us denote them as similar candidate keyphrases, which could be defined as follows:

Definition 1 Similar candidate keyphrases, σ, are those candidate keyphrases that contain γ in them—irrespective of its position, and $\sigma \subseteq \chi$.

A partial sample of σ for γ = *servic* could be: σ = {*scalabl grid servic discoveri base, grid servic, servic discoveri mechan, scalabl web servic permiss, distribut grid servic discoveri architectur, servic discoveri architectur, grid discoveri servic, servic discoveri, grid inform servic, servic discoveri grid comput, servic technolog, servic discoveri function, grid servic call registri, web servic version, discoveri servic, servic properti, thi servic, index servic, servic discoveri, web servic commun, ...*}. Among them, the first encountered similar candidate keyphrase, σ_1 (e.g., *scalabl grid servic discoveri base*), is employed in forming the *KePhEx* tree and the rest are utilized in processing the tree (see the "Tree Processing" section).

Here, the process of tree formation starts by selecting the position of γ in σ_1; but the tree starts forming once the γ is assigned as the root of the tree and μ value is initialized to 1. For any other word (w_i), its position, w_i^p, is determined at first to decide in which subtree it would be placed. If position of γ, γ^p, is more than w_i^p (i.e., $\gamma^p > w_i^p$), it would be placed in the left subtree, otherwise (i.e., $\gamma^p < w_i^p$), the right subtree. Again, the depth of w_i, w_i^d, in a phrase with respect to γ is also necessary to calculate for determining the level of the tree where w_i would be added, which could be defined as follows:

Definition 2 Depth of w_i, w_i^d, in a keyphrase is the distance of that word from γ irrespective of its direction, which is calculated as, $w_i^d = |\gamma^p - w_i^p|$.

Note that w_i^d in a candidate keyphrase of w_i and the level of w_i in the *KePhEx* tree, w_i^l, are identical, and hence, they are used interchangeably in this paper. Once the subtree of w_i is determined using w_i^p, w_i^d is calculated. The next condition to be satisfied is that all the predecessors must be in their respective places. This can be tracked by traversing the tree from level 0 to $l-1$ and by comparing the word in each level with that of in σ_1 at that depth. Once these constraints are satisfied, w_i is qualified for adding in the tree at level l. For that, a node is created by incorporating w_i in it and initializing μ to 1.

Once all the words at the left side of γ are added in the left subtree, then the words at the right side of γ are added in the right subtree following the same procedure. The tree formation ends when all the words of σ_1 are added in the tree. This entire process is illustrated in Algorithm 1.

A sample tree is depicted in Fig. 4, which is formed using σ_1 = *scalabl grid servic discoveri base* and γ =*servic*. The tree formation starts by adding *servic* in the tree as root and initializing μ of the node to 1. Afterwards, all the words at the left side (i.e., *grid* and *scalabl*) are added in the left subtree in their respective levels, where levels are calculated based on their respective depths in σ_1. For instance, since $grid^d = 1$, *grid* is added at level 1 in left subtree, whereas, since $scalabl^d = 2$, *scalabl* is added at level 2 in left subtree. Again, when *grid* is added in the tree, it is tracked that its predecessor *servic* is in the tree. Similarly, when *scalabl* is added in the tree, it is tracked that *grid* and *servic* are its predecessors, respectively. Once all the words at the left side of *servic* are added in the tree, the words at the right side (i.e., *discoveri* and *base*) are added in the right subtree employing a similar procedure as the left subtree.

Tree Processing

After forming the tree employing σ_1, the rest of the similar candidate keyphrases, σ', where σ' = {σ_2, σ_3, ..., σ_n} are utilized to process the tree. For that, the cases that are mentioned in the "Preliminaries" section are taken into account, i.e., no tree processing is needed for *case 1*; the tree must be trimmed properly to remove unnecessary parts for *case 2*; and it must be expanded to put on necessary parts from all the similar candidate keyphrases in σ' for *case 3*. This process is described in Algorithm 1.

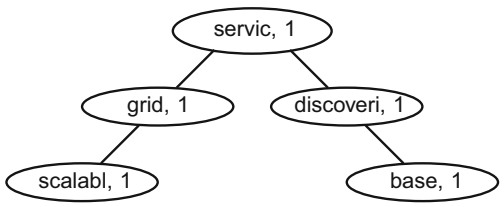

Fig. 4 A newly created tree using the candidate keyphrase, *scalabl grid servic discoveri base*, where $\gamma = servic$

Algorithm 1 AddNode( ): Adding a Node in the KePhEx tree.

Inputs: word (w_i), wordIndex (ι_{w_i}), root (γ), rootIndex (ι_γ), termFreq (τ), phrase (χ_j)
Output: return **TRUE** if a node is added; else return **FALSE**
Variables and others: depth (d), level (ℓ), Node (N), newNode (N_n)

```
1: if w_i = Null then
2:     return
3: if γ = Null then
4:     create N
5:     γ ← N
6:     return TRUE
7: d ← |ι_{w_i} − ι_γ|
8: ℓ ← 0
9: N_n ← γ
10: if ι_{w_i} < ι_γ then
11:     while ℓ < d - 1 || N_n != Null do
12:         if N_n.word = w_i then
13:             return TRUE
14:         else if N_n.prev != Null & N_n.prev.word
            = χ_j[ι_γ - ℓ -1] then
15:             N_n ← N_n.prev
16:             ℓ++ & continue
17:         else if N_n.next!= Null & N_n != γ &
            N_n.next.word = χ_j[N_n - ℓ -1] then
18:             N_n ← N_n.next
19:             ℓ++ & continue
20:         else
21:             if N_n.prev = Null then
22:                 create N
23:                 N_n.prev ← N
24:                 return TRUE
25:             else if N_n.next = Null & N_n != γ then
26:                 create N
27:                 N_n.next ← N
28:                 return TRUE
29:             else if N_n.prev != Null & N_n.prev.word
                != χ_j[ι_γ - ℓ -1] then
30:                 if N_n.prev.τ < τ then
31:                     create N
32:                     N_n.prev ← N
33:                     return TRUE
34:                 else
35:                     return FALSE
36:             else if N_n.next != Null & N_n != γ &
                N_n.next.word != χ_j[ι_γ - ℓ -1] then
37:                 if N_n.next.τ < τ then
38:                     create N
39:                     N_n.prev ← N
40:                     return TRUE
41:                 else
42:                     return FALSE
43:             else
44:                 return FALSE
45:     return FALSE
46: if ι_{w_i} > ι_γ then
47:     while ℓ < d - 1 or N_n != Null do
48:         if N_n.word = w_i then
49:             return TRUE
50:         else if N_n.next != Null & N_n.next.word =
            χ_j[ι_γ + ℓ +1] then
51:             N_n ← N_n.next
52:             ℓ++ & continue
53:         else if N_n.prev != Null & N_n != γ &
            N_n.prev.word = χ_j[ι_γ + ℓ +1] then
54:             N_n ← N_n.prev
55:             ℓ++ & continue
56:         else
57:             if N_n.next = Null then
58:                 create N
59:                 N_n.next ← N
60:                 return TRUE
61:             else if N_n.prev = Null & N_n != γ then
62:                 crete N
63:                 N_n.prev ← N
64:                 return TRUE
65:             else if N_n.next != Null & N_n.next.w_i !=
                χ_j[ι_γ + ℓ +1] then
66:                 if N_n.next.τ < τ then
67:                     create N
68:                     N_n.next ← N
69:                     return TRUE
70:                 else
71:                     return FALSE
72:             else if N_n.prev != Null & N_n != γ &
                N_n.prev.w_i != χ_j[ι_γ + ℓ +1] then
73:                 if N_n.prev.τ < τ then
74:                     create N
75:                     N_n.next ← N
76:                     return TRUE
77:                 else
78:                     return FALSE
79:             else
80:                 return FALSE
81:     return FALSE
```

Let us fetch a similar candidate keyphrase, σ_i', from σ', and utilizes it for processing the *KePhEx* tree. At first, γ^p in σ_i' need to be determined. Like tree formation, the tree processing also starts from γ followed by the words at the left side of γ and then, right side. Afterwards, any word ($w_i \in \sigma_i'$) at position w_i^p is qualified to be added to the left subtree if $w_i^p < \gamma^p$; otherwise, when $w_i^p > \gamma^p$, it is qualified to be added to the right subtree. Again, the depth (w_i^d) is calculated to determine at which level w_i is qualified to be added in the tree and all the predecessors (from 0 to $l-1$) are checked with the ones in σ_i' before their inclusion.

At level l, where w_i is qualified for possible inclusion, three events can occur: (i) there is no node, (ii) there is only one node, and (iii) there are two nodes. In the case of the first event, a node is created for w_i by initializing μ to 1, and then, is added it as a left child for the left subtree or as a right child for the right subtree. For the second event, if the word in the node is the same as w_i, then no node is added. Otherwise, a node is created like before and it is added as a new child at the present level in the subtree. Lastly, if both children already exist at that level, the new node with w_i replaces the node whose word has the lowest *TF*. The reason is that any word with higher *TF* is highly likely to form a quality final keyphrase. For that, if the lower *TF* node is a leaf node, the new node will replace it. Otherwise, if it is a root of a subtree, then the subtree is deleted from the tree and the new node is added in that position. This process is deemed complete when all the words of σ_i' have been considered.

Update μ Values The process of updating μ values starts as soon as the nodes of σ_i' have been added to the tree as demonstrated in Algorithm 2. It starts by determining γ^p in σ_i'. If γ^p is 0, i.e., γ is the leftmost word of σ_i', μ values of all the nodes in the left subtree are decreased. Similarly, if γ^p is $|\sigma_i'|-1$, i.e., γ is the rightmost word of σ_i', μ values of all the nodes in the right subtree are decreased. Afterwards, the μ value of the root is increased and the tree is traversed and compared starting from the left subtree followed by the right subtree using iterative procedures.

At a given level l for any w_i, three events may occur: (i) w_i is absent in l, (ii) w_i is present as a left child, and (iii) w_i is present as a right child. For the first event, μ values of all the nodes in the left and right subtree are decreased. In the second case, μ value of the left child is increased, whereas they are decreased for the nodes in the right subtree, and then, move to the next level. In the case of the last event, μ value of the right child is increased, whereas they are decreased for the nodes in the left subtree, and then, move to the next level. This procedure continues until all the words are taken into account.

An example of tree processing and updating μ values are demonstrated in Fig. 5, where tree in Fig. 4 is utilized as the initial tree. Again, the tree is formed using σ_1 in σ, and the rest (i.e., σ') are utilized to process the tree. As in Fig. 5a, since σ_1' is *grid servic*, and both the words already exist in the tree in sequence, the tree remains in the same state as before. However, μ values of the nodes that contain *grid* and *servic* are increased, and all others are decreased. In Fig. 5b, among the three words, only *mechan* does not exist in the right subtree at level 2; therefore, it is added as the left child. Afterwards, μ values are increased based on σ_2'. Similarly, the tree keeps amending with every encountered σ_i' and μ values are also updated accordingly. This process keeps continuing until all the keyphrases in σ' are processed. Although this example demonstrates only expansion or no change of tree state, the shrinkage occurs in the keyphrase extraction phase.

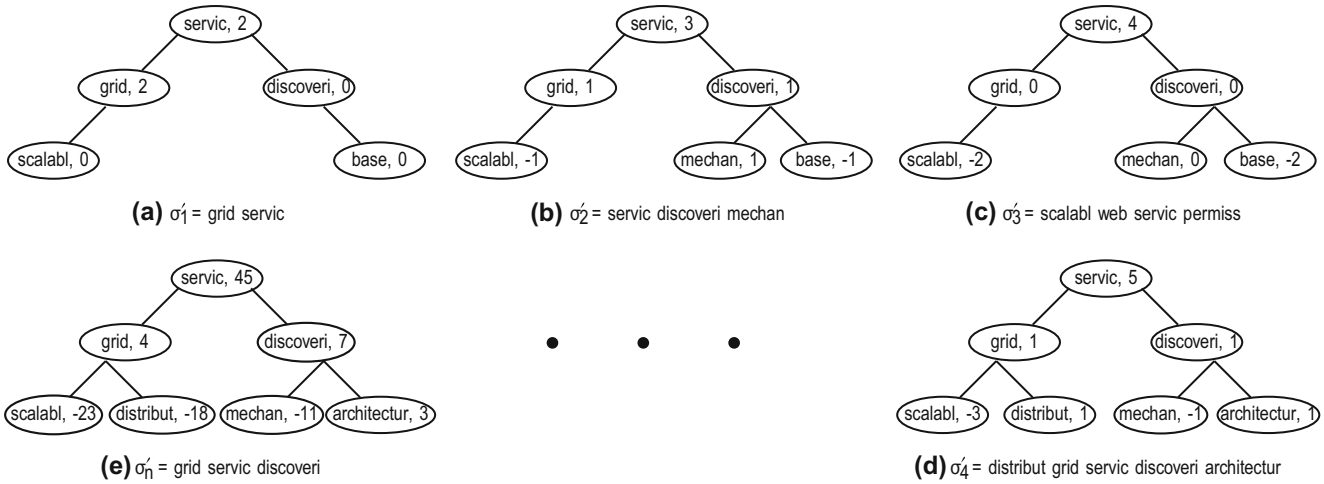

Fig. 5 Several tree processing steps are shown for various similar candidate keyphrases, where $\gamma = servic$

Algorithm 2 Update μ values.

Inputs: root (γ), phrase (χ_j), newNode (N)

find γ^p in χ_j
if γ^p = -1 **then**
    return
else if $\gamma^p = 0$ and $\gamma.prev$!= Null **then**
    decrease μ of all Nodes in left subtree by 1
else if $\gamma^p = len(phrase) - 1$ and $\gamma.next$!= Null **then**
    decrease μ of all Nodes in right subtree by 1
increase μ value of γ by 1
assign $\gamma.prev$ to N
for i from γ^p - 1 to 0 **do**
    if N = Null **then**
        break
    if N.word = χ_j[i] **then**
        increase μ value of N by 1
    if $N.prev$!= Null and $i - 1 > -1$ and *N.prev.word* =χ_j[i - 1] **then**
        if $N.next$!= Null **then**
            decrease μ of all Nodes in right subtree by 1
            assign $N.prev$ to N and continue
        else if $N.next$!= Null and $i - 1 > -1$ and $N.next.word$ = χ_j[i - 1] **then**
            if $N.prev$!= Null **then**
                decrease μ of all Nodes in left subtree by 1
                assign $N.next$ to N and continue
            else
                decrease μ of all Nodes in left subtree by 1
                decrease μ of all Nodes in right subtree by 1
                break
        else
            decrease μ of all the nodes in the subtree where N is the root by 1
assign $\chi_j.next$ to N
for i from γ^p + 1 to $len(phrase) - 1$ **do**
    if N = Null **then**
        break
    if $N.word$ equals χ_j[i] **then**
        increase μ value of N by 1
    if $N.next$!= Null and $i + 1 < len(phrase)$ and $N.next.word$ = χ_j[i + 1] **then**
        if $\chi_j.prev$!= Null **then**
            decrease μ of all Nodes in left subtree by 1
            assign $N.next$ to N and continue
        else if $N.prev$!= Null and $i + 1 < len(phrase)$ and $N.prev.word$ = χ_j[i + 1] **then**
            if $N.next$!= Null **then**
                decrease μ of all Nodes in right subtree by 1
                assign $N.prev$ to N and continue
            else
                decrease μ of all Nodes in left subtree by 1
                decrease μ of all Nodes in right subtree by 1
                break
        else
            decrease μ of all the nodes in the subtree where N is the root by 1

Algorithm 3 Find paths from leaf to root.

Input: root (γ), nodeList, node (η)
Output: return leftPaths, rightPaths

Procedure RootToLeafPaths(root, nodeList, leftPaths, rightPahts)

if γ = Null **then**
    return
append γ in nodeList
RootToLeafPaths($\gamma.prev$, nodeList, leftPaths, rightPahts)
if $\eta.prev$ = Null and $\eta.next$ = Null **then**
    create an empty container, x
    for j from 0 to $len(nodeList) - 1$ **do**
        append $nodeList$[j] in x
    if $len(nodeList) > 1$ **then**
        if $nodeList[0].prev = nodeList[1]$ **then**
            append x in leftPaths
        else
            append x in rightPaths
    else
        append x in leftPaths
        return
RootToLeafPaths($\gamma.next$, nodeList, leftPaths, rightPaths)
pop an element from the nodeList

Algorithm 4 Find Keyphrases from the KePhExTree.

Input: μ, finalPhraseList, root (γ), mamu, newPhrase (φ). morePhase (Ω)
Output: finalPhraseList

```
prune the tree according to the mamu value
if γ = Null then
    return finalPhraseList
else if γ.μ < μ then
    return finalPhraseList
create empty lists, such as nodeList, leftPaths, rightPaths
call the procedure at Algo. 3
create two empty lists, namely leftPhraseList, rightPhraseList
    /* Append from leftPaths */
for i from 0 to len(leftPaths) − 1 do
    create an empty list, named φ
    for l from 0 to len(leftPaths[i] − 1 do
        create an empty list, named morePhrase
        insert leftPaths[i][l] in ϕ at position 0
        copy φ in Ω
        append Ω in leftPhraseList
        if Ω does not exist in finalPhraseList and it is found in candidatePhraseList then
            append Ω to finalPhraseList
    /* Append from rightPaths */
for j from 0 to len(rightPaths) − 1) do
    create an empty list, named φ
    for m from 0 to len(rightPahts[j]) − 1 do
        create an empty list, named Ω
        append rightPaths[j][m] in φ
        copy φ in Ω
        append Ω in rightPhraseList
        if Ω does not exist in finalPhraseList and it is found in candidatePhraseList then
            append Ω to finalPhraseList
    /* Combine leftPhraseList and rightPhraseList */
for i from 1, len(leftPhraseList) − 1 do
    create an empty list, named φ
    if len(leftPhraseList[i]) > 1 then
        for l from 0, len(leftPhraseList[i]) − 1 do
            append leftPhraseList[i][l] to φ
    else
        continue
    for j from 0, len(rightPhraseList) − 1 do
        if len(rightPhraseList[j]) > 1 then
            create an empty list, named Ω
            copy φ in Ω
            for r from 1, len(rightPhraseList[j]) do
                append rightPhraseList[j][r] in Ω
            if Ω does not exist in finalPhraseList and it is found in candidatePhraseList then
                append Ω is finalPhraseList
return finalPhraseList
```

Keyphrase Extraction

This process is initiated by pruning the weak nodes from the tree. Here, weak nodes are selected based on their cohesiveness with respect to γ with an assumption that they may not be the parts of final keyphrases. For that, a constant integer value, named *minimum allowable* μ *(mamu)*, is utilized. A node whose μ value is lower than the *mamu* is pruned from the tree. For instance, in Fig. 5e, it could be observed that several nodes in the tree contain lower μ values, i.e., their cohesiveness with respect to γ is weak, and hence, most likely, they would not be a part of the final keyphrase. Now, *mamu* value determined which nodes to keep in the tree and which to prune from the tree. Such a tree is depicted in Fig. 6, where *mamu* is considered as 2.

Hence, if that node is a root of a subtree than that entire subtree is also erased from the tree with the assumption that a weak root would form a weak subtree. Again, a *mamu* value must be selected with considerable attention because a smaller *mamu* value results in many and/or long keyphrases, whereas a large *mamu* value results in lower and/or abbreviated keyphrases. Therefore, it is essential to find a suitable *mamu* value for improved performance of the system. Hence, this paper conducts an experiment to find a suitable *mamu* value (see the "Parameter Value Selection" section). Again, this *mamu* value also provides flexibility during keyphrase extraction.

Afterwards, all paths from the root to the leaves are extracted to discover final keyphrases. Since this procedure is dissimilar to any conventional tree traversal technique (namely *preorder*, *inorder*, and *postorder*), they are not directly applicable in this case. Hence, *inorder* tree traversal technique is enhanced to perform the task, which is explained in Algorithm 3. This algorithm extracts all the paths from root to leaf and separates them in left paths (paths from left subtree) and right paths (paths from right subtree), which are later processed to generate final keyphrases individually (one final keyphrase from one path) or collectively (by joining a path from the left

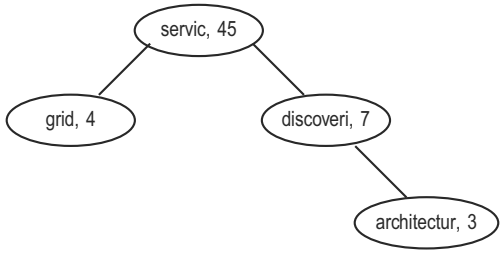

Fig. 6 The resultant tree for *mamu* = 2 for the *KePhEx* tree in Fig. 5

subtree and a path from the right subtree) as demonstrated in Algorithm 4.

Now, in the case of left paths, since they are extracted from root to leaf, they are unlikely to be the final keyphrases as they are aligned in reverse direction, and hence, misses the coherent relationship. Therefore, all left paths are reversed before extracting final keyphrases. Afterwards, all the words are acquired from each path and a keyphrase is formed. Then, its presence (entirely) is checked in χ as a candidate keyphrase or a part of candidate keyphrase. A similar technique is followed to extract keyphrases from right paths with an exception is that the paths are not reversed since they are already satisfying the coherent relationship conditions. After acquiring all the final keyphrases from the left and right paths, they are concatenated to generate more long and meaningful keyphrases. Again, these keyphrases will qualify as final keyphrases if they are entirely found in χ as candidate keyphrases or part of candidate keyphrases.

Flexibility During Keyphrase Extraction The proposed technique offers flexibility in keyphrase extraction via employing the *mamu* values. As an example, Table 1 is generated using the tree in Fig. 6. As expected, for different *mamu* values, different final keyphrases are generated. These keyphrases also differ in length and quantity. For instance, the longest keyphrase generated by *mamu* values from 1 *to* 3 is 4, whereas it is 3 for *mamu* value 4, 2 for *mamu* values from 5 to 7 and so on. On the other hand, for *mamu* values from 1 to 4, 3 final keyphrases are extracted, whereas it is only 1 for *mamu* values from 5 to 45 and 0 afterwards. From here, we can conclude that a greedy approach may choose a lower *mamu* value and hence, would get considerably many and/or lengthy keyphrases; but the quality would be a little bit compromised. On the other hand, a conservative approach may choose a large *mamu* value which will in turn provide considerably lower and/or mostly abbreviated keyphrases. Hence, to receive a desired level of performance, *mamu* value must be set properly. To realize this, an experimental evaluation is performed in the "Results and Discussion" section and the results are analyzed with detail evidences.

Table 1 Final keyphrases from the resultant tree in Fig. 6

SN	*mamu (+ve value)*	Final keyphrase
1	1 to 3	grid servic
2	1 to 3	servic discoveri architectur
3	1 to 3	grid servic discoveri architectur
4	4	grid servic
5	4	servic discoveri
6	4	grid servic discoveri
7	5 to 7	servic discoveri
8	8 to 45	servic
9	≥ 46	—

After extracting all the final keyphrases from the tree for a γ, the next γ is chosen from the list η and the same procedure is repeated again. It continues until all the nouns in η are considered as γ. After finish extracting all the final keyphrases, they are passed for ranking and selecting.

Ranking and Selecting Final Keyphrases

Generally, automatic keyphrase extraction techniques extract a good number of final keyphrases. However, various applications including recommender system and document indexing techniques utilize only a certain number of top keyphrases. Therefore, an automatic keyphrase extraction technique must offer the most relevant top-N keyphrases to these applications. Hence, keyphrase extraction is also accounted for as a ranking problem.

In the proposed ranking technique, the μ value is employed along with the *TF* as follows to calculate weight, ω of a keyphrase p:

$$\omega_p = \sum_{k=1}^{N} tf_k \times \sum_{k=1}^{N} \mu_k \tag{1}$$

Here, N is the number of words in p. The first factor in Eq. 1, i.e., *TF*, is utilized to identify the popularity of that particular keyphrase in a document with an assumption that the non-popular keyphrases are unlikely to become a final keyphrase. For that, *TF* of all the words in p are summed together. It is noteworthy to mention that instead of averaging each factor, summation is performed to eliminate the bias towards the single terms. Again, the second factor is for realizing the cohesiveness of every word in that keyphrase to γ, which can be found by summing the μ values of all the words in p.

After calculating the ω values for all keyphrases, they are sorted to arrange them in rank. Since the quantity of final keyphrases is limited, any sorting algorithm is suitable. In the proposed system, the *quick sort* [27] algorithm is applied to perform the task rapidly. After ranking, these keyphrases

are ready to be rendered. Now, when a user or an application seeks for any N keyphrases, the system will provide top-N keyprhases from the rank 1 to N, respectively.

Experimental Setup

Since the proposed technique is an *unsupervised machine learning based technique*, its performance is compared with other relevant unsupervised techniques. For this, both statistical (*TF-IDF* and *YAKE*) and graph-based (*singleRank* (*SR*), *positionRank* (*PR*), *topicRank* (*TR*), and *multipartiteRank* (*MR*)) keyphrase extraction techniques are considered. All of these techniques are evaluated under a uniform experimental setup taking multiple available benchmark corpora into consideration, which are elaborated in the subsequent section.

Corpora Details

The primary corpus that has been employed for testing the proposed technique along with other similar techniques is the *SemEval-2010* [33]. This dataset is composed of a train and a test dataset along with other datasets that are collected from the *ACM Digital Library*. Since our proposed technique and all the compared techniques are unsupervised techniques, train and test datasets are not utilized as per their literal meaning. Therefore, they are denoted as set 1 and set 2, respectively in this paper, which will also eliminate any further confusions. This corpus has been chosen since it ensures the variability in terms of topics. Here, all the papers are clustered in four groups following four 1998 ACM classifications: C2.4—Distributed Systems, H3.3—Information Search and Retrieval, I2.11—Distributed Artificial Intelligence— Multiagent Systems, and J4—Social and Behavioral Sciences—Economics. The distribution of documents in the corpus is mentioned in Table 2.

All the documents in the corpus are in plain text and the average length of these documents is about 2000 words. Although the *XML* version of this dataset exists, we prefer text dataset since the former one is heavy, verbose, and rare. For comparison, gold standard keyphrases have been employed that come along with the dataset and composed of author-assigned keyphrases and reader-assigned keyphrases. Table 3 exhibits the distribution of author- and reader-assigned keyphrases in the corpus.

Again, for testing the domain independence of the proposed technique, *Theses100* benchmark dataset [69] is employed. This dataset is composed of 100 master and Ph.D. theses from the University of Waikato, New Zeland. All the documents are in plain text, and the average length of these documents is about 7000 words. For comparison, gold standard keyphrases have been taken into account that come along with the dataset.

Furthermore, a *German Research Article* dataset has been created to test the language independence of the proposed technique due to the absence of such benchmark dataset, which is later uploaded in for further reference. All the articles in this dataset are collected from various open score research article database [51]. All the documents in this corpus are in plain text and the average length of these documents is about 2000 words. For comparing the performance of various keyphrase extraction techniques, author-assigned keyphrases are considered as gold standard keyphrases.

Evaluation Metrics

Three prominent and relevant metrics, namely, *precision* (ϱ), *recall* (ς), and *F1-score* (ϕ) have been used for comparing the proposed technique's performance with other considered techniques. Here, ϱ is the ratio of correctly predicted positive values with respect to the total predicted values. It can be calculated using the following formula:

$$\varrho = \frac{\kappa_{\text{correct}}}{\kappa_{\text{extract}}} \tag{2}$$

where, κ_{correct} is the number of correctly matched keyphrases with gold standard keyphrases and κ_{extract} is the number of extracted keyphrases from a document, i.e., value of N in case of extracting top-N keyphrases.

On the other hand, ς is the ratio of correctly predicted positive values with respect to the actual positive values and can be calculated as follows:

$$\varsigma = \frac{\kappa_{\text{correct}}}{\kappa_{\text{standard}}} \tag{3}$$

where, κ_{standard} is the number of keyprhases in gold standard keyphrase list for that particular document.

Table 2 Number of documents per topic in the four ACM document classifications

Dataset	Total	Document topic			
		C	H	I	J
Set 1	144	34	39	35	36
Set 2	100	25	25	25	25

Table 3 Keyphrase distribution of gold standard in different datasets

Dataset	Author assigned	Reader assigned	Combined
Set 1	559	1824	2223
Set 2	387	1217	1482

Again, ϕ is the weighted average of ϱ and ς, which can be calculated using the following formula:

$$\phi = \frac{2 \times \varrho \times \varsigma}{\varrho + \varsigma} \tag{4}$$

This metric is much more sophisticated than conventional accuracy metric since it takes both false positives and false negatives into consideration.

Implementation Details

The proposed technique is implemented using Python3 employing several necessary packages, such as PorterStemmer [31, 32, 47], Sent_tokenize, Word_tokenize of Natural Language Tool Kit [4, 62], Regular Expression [13, 38], and so on. Note that all the words are stemmed initially before passing them to the processing phase employing porterStemmer. Again, for gold standard keyphrases, no such processing is required since they are already stemmed.

For other compared techniques, *Python Keyphrase Extraction* (*PKE*) toolkit [5]—which is an open-source python-based keyphrase extraction toolkit—is utilized. Here, we would like to mention that for all the experiments, whatsoever, a uniform experimental environment is offered to ensure a level playing ground for all the techniques. For the compared techniques, top-N keyprhases are acquired from the *PKE* using respective *Application Programming Interfaces* (*APIs*). Afterwards, these acquired keyphrases are compared with the gold standard keyphrases, and then, metrics are calculated accordingly. All experimental codes and corpus are currently available in [50] for access upon request.

Results and Discussion

This section includes the results that are acquired from the experiments along with their detail analyses. It starts with selecting suitable parameter values which have direct influence on the performance of the proposed technique. For other compared techniques, standard parameter values are selected as suggested in [5].

Parameter Value Selection

Among various parameters of the proposed technique, two parameters have definite impacts on the performance, which are *lsaf* (discussed in the "Conceptual Framework" section) and *mamu* (discussed in the "Keyphrase Extraction" section). Here, the former parameter is utilized to filter out all non-popular candidate keyphrases from the list and the latter plays an important role in extracting keyphrases from the resultant tree. As mentioned earlier, a lower *mamu* value would result in many but low-quality keyphrases, whereas a high *mamu* value would result in few but abbreviated keyphrases. Therefore, it is necessary to determine, which *mamu* value would give the superlative performance.

For determining the suitable *lsaf* value, an experiment has been performed varying it from 0 to 5 for two arbitrarily selected *mamu* values. The experiments are performed on set 2 dataset to acquire top-N keyphrases, where $N = 5$, 10, and 15, which are then utilized to calculate precision, recall, and F1-score. The acquired results are demonstrated in Table 4. The highest performance shown for *F1* value is 15.6 for top-15 keyphrases by $lsaf$ values 3 and 4, whereas the lowest performance shown is 10.5 for the top-5 keyphrases by $lsaf$ value 1. It is because a lower value of

Table 4 Performance of proposed technique for various *lsaf* values for two arbitrarily selected μ values on set 2 dataset

lsaf	μ	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
1	0	20.3	7.1	10.5	16.2	11.5	13.3	13.5	14.1	13.7
1	2	20.5	7.2	10.6	16.8	11.9	13.8	14.0	14.6	14.2
2	0	21.3	7.6	11.2	17.2	12.3	14.2	14.3	15.2	14.6
2	2	21.9	7.8	11.5	17.0	12.1	14.1	14.4	15.3	14.7
3	0	21.3	7.6	11.1	17.7	12.6	14.6	14.9	15.8	15.3
3	2	21.3	7.6	11.1	17.8	12.6	14.6	15.3	16.1	15.6
4	0	21.6	7.7	11.28	17.9	12.71	14.74	15.2	16.1	15.5
4	2	21.6	7.7	11.28	17.9	12.71	14.75	15.3	16.2	15.6
5	0	21.6	7.7	11.28	17.9	12.71	14.74	15.1	16	15.4
5	2	21.6	7.7	11.28	17.9	12.71	14.75	15.2	16.1	15.5

$lsaf$ incorporates non-popular keyphrases during ranking, and thus, entice ranking approach. From the results, it is evident that with increasing *lsaf* value, *F1* value increases for any *mamu* value until $lsaf = 3$; afterwards, it becomes almost steady. Hence, 3 could be considered as the threshold value of *lsaf* and is utilized in other experiments.

Again, to select a suitable *mamu* value, we also conduct another set of experiments varying *mamu* values from 0 to 5, fixing *lsaf* to 3, and taking set 1 and set 2 datasets of the corpus into consideration. For both datasets, results are acquired for top-N keyphrases, where $N = 5$, 10, and 15. All the acquired results are stated in Tables 5 and 6 for set 1 and set 2 datasets, respectively. They are also plotted using contour graphs in Figs. 7 and 8 for more depictions.

As could be observed from the tables as well as from the figures is that performance differences in several *mamu* values are not as evident as *lsaf* values since we have already filtered out non-popular keyphrases by selecting $lsaf = 3$. The highest *F1* achieved is 15.6 for set 2 dataset and 13.2 from set 1 dataset, and both cases, it is achieved by $mamu = 2$. Again, for most of the cases with increasing *mamu* values, performance increases to a certain point, and afterwards, it decreases. In our case, $mamu = 2$ is the threshold for both the datasets. The reason is that it maintains the trade-off between the keyphrase length and quantity. On the other hand, smaller *mamu* values produce considerably many and/or lengthy keyphrases; but the quality is a little bit compromised, whereas higher *mamu* values attain considerably lower and/or mostly abbreviate keyphrases. In the latter case, since lengthy keyphrases are ignored, the performance is also a little bit compromised. Hence, $mamu = 2$ is locked for the rest of the experiments.

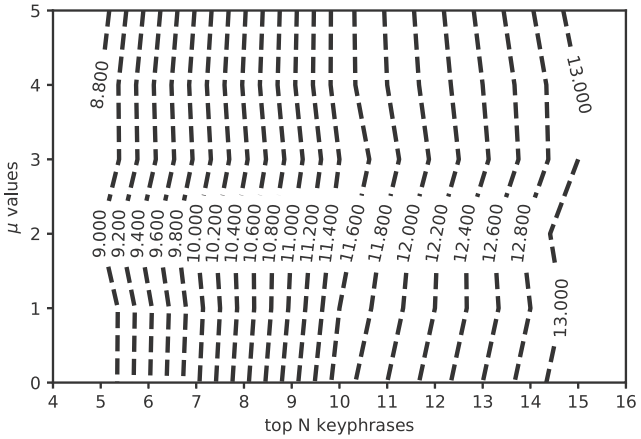

Fig. 7 Performance of the proposed technique for various μ values on set 1 dataset

Results Analyses

Here, we would like to note that the performance of all the technique would have improved if 15% of the reader-assigned keyphrases that are absent would have appeared in the text, and if 19% of the author-assigned keyphrases that are absent would have appeared in the text. Hence, all the results in this paper are based on 85% and 81% for the reader- and author-assigned keyphrases, respectively.

For all the techniques, three experiments are performed for each dataset with a target of extracting top-N keyphrases, where $N = 15$ is preferred in many literatures [33, 34], and hence, is our choice. Again, once we have top-15 keyphrases, we can derive the top 5 and top-10 keyphrases from there. These experiments are performed for (i) reader-assigned keyphrases, (ii) author-assigned keyphrases, and (iii) combined keyphrases (combines reader- and author-assigned keyphrases). The acquired results for set 2 dataset are shown in Tables 7, 8, and 9 for reader-assigned, author-assigned, and combined keyphrases, respectively.

From the tables, it could be observed that generally, statistical-based techniques performed better than graph-based techniques. It is because graph-based techniques are not good in capturing the cohesiveness of words in a keyphrase, experience clustering errors, suffer from error propagation problem and so on, which are mentioned the "Related Works" section.

Table 5 Performance of proposed technique for various μ values on set 1 dataset

μ	Top 5			Top 10			Top 15		
	P	R	F1	P	R	F1	P	R	F1
0	17.6	6.0	8.8	14.8	9.9	11.7	13.3	13.6	13.2
1	17.6	6.0	8.8	14.7	9.9	11.6	13.1	13.5	13.1
2	17.9	6.1	9.0	14.5	9.8	11.5	13.2	13.6	13.2
3	17.6	5.9	8.8	14.4	9.7	11.4	13.0	13.3	13.0
4	17.6	5.9	8.8	14.5	9.8	11.5	13.1	13.4	13.0
5	17.4	5.9	8.7	14.4	9.7	11.5	13.1	13.5	13.1

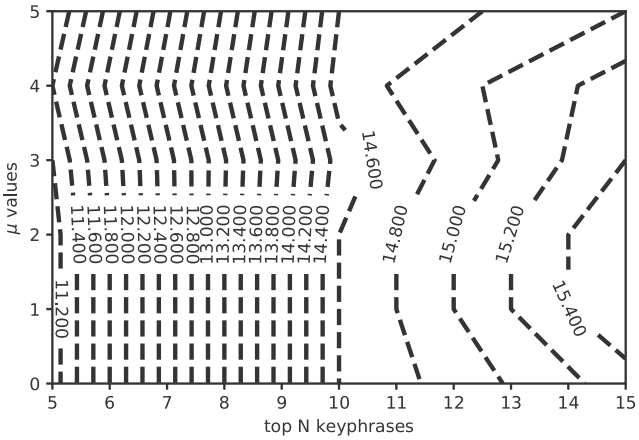

Fig. 8 Performance of the proposed technique for various μ values on set 2 dataset

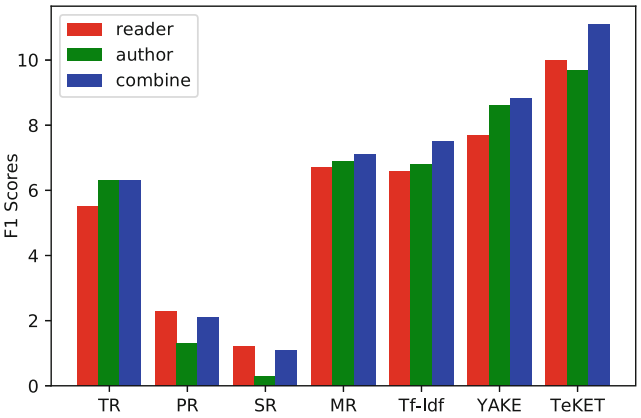

Fig. 9 F1 scores of various unsupervised keyphrase extraction techniques for top-5 keyphrases employed on set 2 dataset

On the other hand, statistical-based techniques are simple to implement and utilize basic features, like term frequency, inverse document frequency, word positions, and word relatedness to a context to extract the most descriptive terms in a document. Despite that, they demonstrate better performance over the graph-based techniques because statistical characteristics of the aforementioned basic features repeat over and over in most of the documents for the top keyphrases.

Again, among all the graph-based techniques, *SR* performs the worst in terms of all the considered metrics. The highest *F1* achieves by this technique is only 1.9 for top-15 in the case of reader-assigned gold standard keyphrases, whereas the lowest is 0.3 for top-5 keyphrases in the case of author-assigned gold standard keyphrases. It is because *SR* assigns higher scores to long but non-significant keyphrases. In detail, *SR* assigns the weights of the edges based on the number of co-occurrences. Afterwards, keyphrases are extracted in the form of noun phrases and then ranked based on the sum of the significance of the words they contain. Therefore, non-significant long keyphrases receive higher scores than abbreviated keyphrases.

With respect to *SR*, *PR* outperforms the former in terms of all the metrics and for all top-*N* keyphrases. This happens because it incorporates the position information of a word and its occurrences to score words. It receives an average *F1* score of 3.57 for reader-assigned keyphrases, 1.9 for author-assigned keyphrases, and 3.63 for combined keyphrases for all the top-*N* cases that we considered in this paper. However, it fails to ensure topical coverage and diversity that are not naturally handled by this kind of graphs.

On the other hand, due to taking the topical coverage into account, *TR* overpowers *PR* technique for any metric or any parameter, which was absent in the latter technique. Here, topic relations are accounted to find the semantic relatedness between the candidate keyphrases they instantiate. It demonstrates an average performance improvement of 93.55% over *PR* for reader-assigned keyphrases, 216.82% for author-assigned keyphrases, and 119.24% for combined keyphrases. Again, *F1* value of top-5 keyphrases contributes more in these performance differences—around 140% for reader-assigned, 385% for author-assigned, and 200% for combined keyphrases. Although it maximizes the topical coverage, it suffers from several limitations. For instance, all candidates under a single topic are considered equally, and therefore, post-ranking heuristics are necessary to select the most representative keyphrases from each topic. Again, if any error occurs while forming topics, it will propagate throughout the model and thus negatively impacts its performance.

Since *MR* resolves the issue of error propagation, it performs superiorly over *TR*, and thus over *SR* and *PR*. To resolve this issue, *MR* utilizes the multipartite graph, hence the name, which connects sets of topic related candidates tightly. The average *F1* receives for reader-assigned keyphrases is 8; whereas, it is for author-assigned keyphrases is 5.47, and combined keyphrases is 7.33. However, it struggles with selecting the most representative candidates due to clustering errors, where candidate keyphrases could be wrongly assigned to the same topic.

Among the statistical-based approaches, *TF-IDF* performs comparably to *MR* for all the metrics and attributes. For instance, it receives an average *F1* of 7 for reader-assigned keyphrases, 6.8 for author-assigned keyphrases, and 8.57 for combined keyphrases. In *TF-IDF*, IDF provides informativeness and TF provides aboutness. Here, *TF* discriminates the non-popular keyphrases from the popular keyphrases in a document, whereas *IDF* discriminates between informative and non-informative keyphrases across the documents. A keyphrase receives high *IDF* when it is

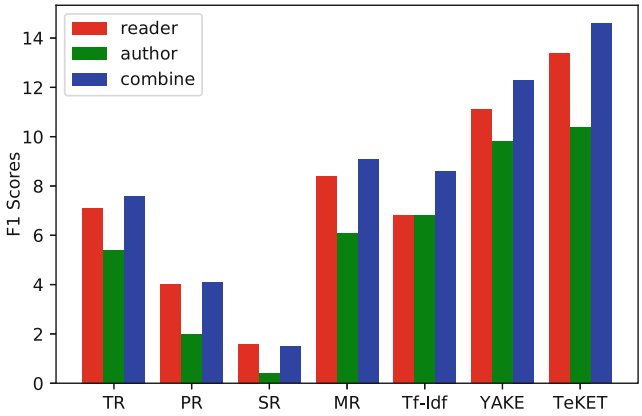

Fig. 10 F1 scores of various unsupervised keyphrase extraction techniques for top-10 keyphrases employed on set 2 dataset

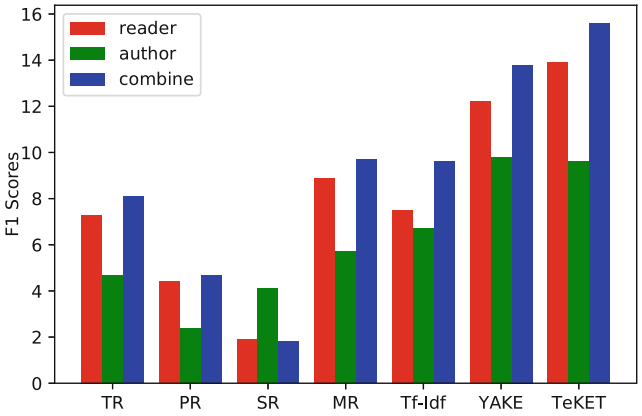

Fig. 11 F1 scores of various unsupervised keyphrase extraction techniques for top-15 keyphrases employed on set 2 dataset

rare along the collections. However, it favors single terms or bias towards single terms over compound terms, and hence, demonstrates considerably lower performance over *YAKE* on set 2 dataset.

In the case of *YAKE*, it takes five features into account, namely casing, word position, word frequency, word relatedness to context, and word in the different sentences, to rank keyphrases. Since many quality keyphrases pursue these statistical features unconsciously, it shows better performance over *TF-IDF* technique. It receives an average performance enhancement of 47.53% for reader-assigned keyphrases, 38.95% for author-assigned keyphrases, and 34.83% for combined keyphrases. However, since candidate keyprhases are generated using N-grams technique, where N is 1-, 2-, and 3-grams, a considerably large number of keyphrases are generated, which entices ranking procedure.

In terms of any metric and any attribute, *TeKET* outperforms the other techniques that are considered in this evaluation significantly. For instance, it outperforms *YAKE* by 21.51% for *F1* measure on an average in case of reader-assigned keyphrases, 5.61% in case of author-assigned keyphrases, and 20.49% in case of combined keyphrases. Again, our proposed technique receives the highest *F1* value among all the techniques, i.e., 15.6, for top-15 keyphrases in case of combined gold standard keyphrase list. One of the reasons for its excellent performance is that it extracts final keyphrases from candidate keyphrases using the *KePhEx* tree, and hence, considers most likely keyphrases during ranking. In addition, it utilizes two factors (*TF* and μ) in ranking, where the preceding factor is utilized to discriminate non-popular keyphrases from popular keyphrases and the latter factor is utilized to find the cohesiveness of various words in a keyphrase with respect to the root. Again, in the calculation, summation is preferred over average to facilitate longer keyphrases.

In Figs. 9, 10, and 11, *F1* scores of various techniques for top-5, 10, and 15 keyphrases are shown in the case of reader-assigned, author-assigned, and combined gold standard keyphrases. Like the table, *SR* demonstrates the substandard performance. Although, *PR* outperforms *SR*, but it falls short in front of *TR* for a considerably larger margin. Again, *MR* and *TF-IDF* demonstrate comparable performance in case of all three top-N values. Although, *YAKE* performs better over other considered keyphrase extraction techniques, but our proposed technique overpowers all

Table 6 Performance of proposed technique for various μ values on set 2 dataset

μ	Top 5			Top 10			Top 15		
	P	R	F1	P	R	F1	P	R	F1
0	21.3	7.6	11.1	17.7	12.6	14.6	14.9	15.8	15.3
1	21.3	7.6	11.1	17.8	12.6	14.6	15.3	16.2	15.6
2	21.3	7.6	11.1	17.8	12.6	14.6	15.3	16.1	15.6
3	21.5	7.7	11.2	17.7	12.5	14.5	15.1	15.9	15.4
4	21.7	7.8	11.4	17.9	12.7	14.7	15.0	15.9	15.3
5	21.3	7.6	11.2	17.8	12.6	14.6	14.7	15.6	15.0

Table 7 Performance of different unsupervised machine learning–based keyphrase extraction techniques for reader-assigned keyphrases on set 2 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	9.4	4.0	5.5	7.8	6.6	7.1	6.6	8.4	7.3
	PositionRank	3.8	1.6	2.3	4.4	3.8	4.0	3.9	5.1	4.4
	SingleRank	1.9	0.8	1.2	1.8	1.5	1.6	1.7	2.2	1.9
	MultipartiteRank	11.3	4.8	6.7	9.2	7.8	8.4	8.0	10.3	8.9
Statistical-based	TF-IDF	11.1	4.7	6.6	7.4	6.4	6.8	6.9	8.9	7.5
	YAKE	12.7	5.5	7.7	12.0	10.4	11.1	10.9	14.1	12.2
Tree-based (proposed)	TeKET	16.5	7.2	10.0	14.5	12.6	13.4	12.5	16.1	13.9

Table 8 Performance of different unsupervised machine learning–based keyphrase extraction techniques for author-assigned keyphrases on set 2 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	6	7.2	6.3	3.9	9.5	5.4	3.1	11.2	4.7
	PositionRank	1.2	1.8	1.3	1.5	3.9	2.0	1.5	6.8	2.4
	SingleRank	0.4	0.4	0.3	0.3	0.7	0.4	0.2	1.2	0.41
	MultipartiteRank	6.4	8.1	6.9	4.4	11.0	6.1	3.7	13.7	5.7
Statistical-based	TF-IDF	6.4	7.9	6.8	4.9	11.9	6.8	4.3	16.4	6.7
	YAKE	7.8	10.3	8.6	6.8	18.6	9.8	6.2	2.4	9.8
Tree-based (proposed)	TeKET	8.8	11.6	9.7	7.3	19.8	10.4	6.1	24.2	9.6

Table 9 Performance of different unsupervised machine learning–based keyphrase extraction techniques for combined keyphrases on set 2 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	12.3	4.2	6.3	9.4	6.5	7.6	8	8.3	8.1
	PositionRank	4.2	1.4	2.1	5.1	3.6	4.1	4.6	4.9	4.7
	SingleRank	2.2	0.7	1.1	1.9	1.2	1.5	1.8	1.8	1.8
	MultipartiteRank	13.9	4.8	7.1	11.1	7.8	9.1	9.5	10.1	9.7
Statistical-based	TF-IDF	14.3	5.1	7.5	10.3	7.4	8.6	9.4	10.1	9.6
	YAKE	16.9	6.0	8.83	14.9	10.6	12.3	13.5	14.3	13.8
Tree-based (proposed)	TeKET	21.3	7.6	11.1	17.8	12.6	14.6	15.3	16.1	15.6

Table 10 Performance of different unsupervised machine learning–based keyphrase extraction techniques for reader assigned keyphrases on set 1 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	8.0	3.1	4.5	6.3	5.0	5.5	5.4	6.5	5.8
	PositionRank	3.6	1.4	2.0	3.2	2.6	2.9	2.9	3.6	3.2
	SingleRank	1.5	0.58	0.84	0.97	0.77	0.85	1.2	1.4	1.3
	MultipartiteRank	8.8	3.5	5.0	7.5	6.0	6.6	6.3	7.7	6.8
Statistical-based	TF-IDF	7.5	3.1	4.4	5.9	4.9	5.3	4.7	5.8	5.2
	YAKE	7.4	3.0	4.2	7.0	5.6	6.1	6.7	8.1	7.2
Tree-based (proposed)	TeKET	14.0	5.7	8.0	11.0	9.0	9.8	10.1	12.7	11.1

Table 11 Performance of different unsupervised machine learning–based keyphrase extraction techniques for author assigned keyphrases on set 1 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	3.4	4.7	3.9	2.7	7.6	3.8	2.2	9.4	3.5
	PositionRank	1.8	2.4	2.0	1.7	5.0	2.5	1.3	5.8	2.2
	SingleRank	0.4	0.6	0.4	0.6	1.8	0.9	0.6	2.6	1.0
	MultipartiteRank	4.4	6.2	5.0	3.4	9.8	5.0	3.0	12.7	4.7
Statistical-based	TF-IDF	4.2	5.9	4.6	3.1	8.0	4.3	2.6	10.0	4.0
	YAKE	5.4	7.2	6.0	4.8	12.8	6.8	4.5	17.9	7.1
Tree-based (proposed)	TeKET	8.4	11.3	9.4	6.7	18.1	9.6	6.1	24.4	9.6

Table 12 Performance of different unsupervised machine learning–based keyphrase extraction techniques for combined keyphrases on set 1 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	9.5	3.0	4.5	7.5	4.8	5.8	6.5	6.3	6.3
	PositionRank	4.8	1.5	2.3	4.4	2.9	3.4	3.8	3.8	3.8
	SingleRank	1.6	0.5	0.7	1.3	0.9	1.0	1.7	1.6	1.6
	MultipartiteRank	11.2	3.6	5.4	9.3	6.1	7.3	8.1	7.9	7.9
Statistical-based	TF-IDF	10.1	3.4	5.0	8.1	5.4	6.4	6.4	6.4	6.3
	YAKE	10.8	3.5	5.3	10.0	6.6	7.9	9.3	9.2	9.1
Tree-based (proposed)	TeKET	17.9	6.1	9.0	14.5	9.8	11.5	13.2	13.6	13.2

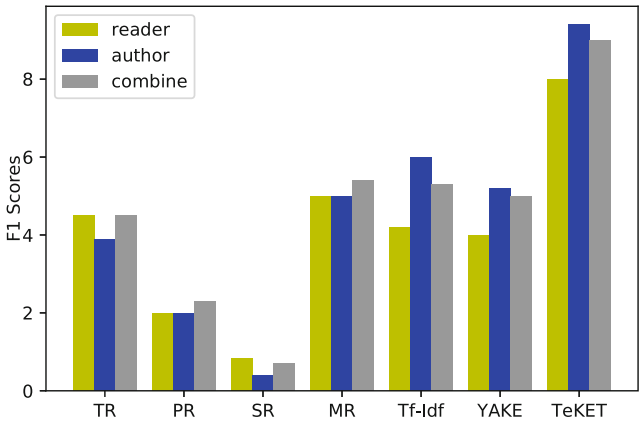

Fig. 12 F1-Scores of various unsupervised keyphrase extraction techniques for top-5 keyphrases employed on set 1 dataset

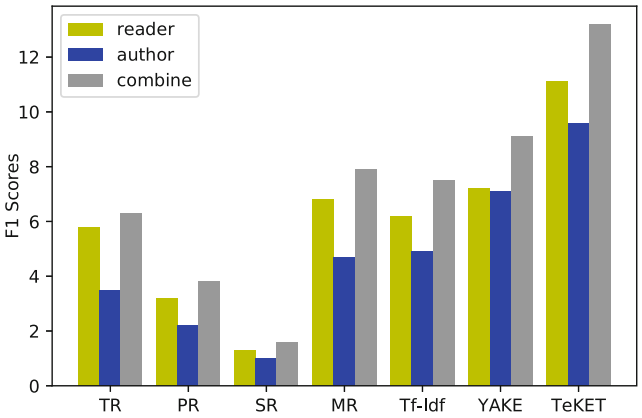

Fig. 14 F1 scores of various unsupervised keyphrase extraction techniques for top-15 keyphrases employed on set 1 dataset

others. The reasons of their performance differences are same as before.

The acquired results for set 1 data are plotted in Tables 10, 11, and 12 for reader-assigned, author-assigned, and combined keyphrases respectively. Likewise for set 2, all the results are acquired for three metrics and compared with top-N keyphrases, where $N = 5$, 10, and 15. The average *F1* scored by the *SR* technique for all cases is 1.05, which is the lowest among all. On the other hand, it is 2.7, 4.84, and 6.03 for *PR*, *TR*, and *MR*, respectively. Due to utilizing multipartite graph, it extracts more gold standard keyphrases than others. Again, the average *F1* scores for *TF-IDF* and *YAKE* are 5.06 and 6.63, respectively. Unlike set 2 data, *TF-IDF* performs worse than *MR* for set 1 data; however, the latter almost catches *YAKE* in terms of *F1* score. Conversely, our proposed technique overpowers all the considered techniques with an average *F1* score of 10.13 for the reasons that are stated before.

The *F1* scores of various gold standard keyphrase classes (reader, author, and combined) for set 1 data are shown in Figs. 12, 13, and 14 for top-5, 10, and 15 keyphrases, respectively. Like the previous cases, performances of *SR*, *PR*, and *TR* remain in the same increasing order. However, *F1* scores of *MR*, *TF-IDF*, and *YAKE* are comparable for all top-N keyphrases, unlike test data where *YAKE* outperforms the other two. In any case, our proposed technique overpowers the rest of the compared techniques.

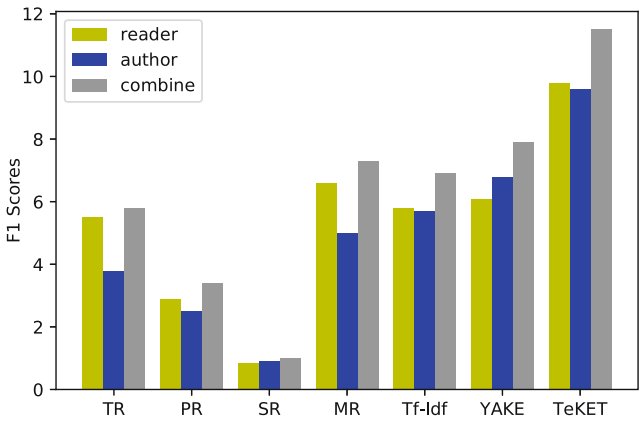

Fig. 13 F1 scores of various unsupervised keyphrase extraction techniques for top-10 keyphrases employed on set 1 dataset

TeKET Is Domain Independent

To demonstrate the domain independence property of the proposed technique, we have conducted an experiment on the *Theses100* dataset (see the "Corpora Details" section for a detailed description) following a similar experimental environment discussed in the "Evaluation Metrics" section. The justification for selecting such a dataset is to highlight that the proposed technique also works satisfactorily with a large amount of words. The average length of the documents in *Theses100* dataset is $\sim$ 7,000 words with respect to $\sim$ 2,000 words in research articles. The obtained results of the experiments are reported in Table 13.

It can be seen from the obtained results that the performance of almost all the comparable techniques deteriorates in terms of the considered metrics. One of the reasons for this low performance is that, when a document contains many words (as in a thesis), keyphrase extraction technique produces a large number of keyphrases. This, in turn, makes it very challenging to select top-N keyphrases from there for the ranking procedure.

Now, while comparing the performance of *TeKET* to other relevant techniques considered in this paper, *TEKET* outperforms the other techniques significantly in any metric and attribute. For example, *TeKET* outperforms its closest competitor, *YAKE*, by 5.2% on *F1* measure for all top-N keyphrases. The reason behind this is that *TeKET* employs an intermediate phase to extract final keyphrases from the candidate keyphrases through the KePhEx tree,

Table 13 Performance of different unsupervised machine learning–based keyphrase extraction techniques on Thesis100 dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	5.8	4.1	4.7	3.9	5.6	4.5	2.8	6.1	3.8
	PositionRank	0.6	0.3	0.4	0.5	0.6	0.5	0.5	1.0	0.7
	SingleRank	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.3	0.2
	MultipartiteRank	6.2	4.6	5.2	4.2	6.1	4.9	3.0	6.6	4.0
Statistical-based	TF-IDF	1.4	1.3	1.2	0.9	1.6	1.1	0.7	2.0	1.0
	YAKE	7.4	5.2	6.0	4.9	7.3	5.8	3.8	8.2	5.1
Tree-based (proposed)	TeKET	9.9	7.6	8.3	7.0	10.5	8.1	5.5	12.6	7.3

and therefore, ranks only most probable keyphrases. On the other hand, in the case of *YAKE*, several inferior keyphrases exhibit similar statistical behavior like top keyphrases, and therefore, perform poorly. Again, although being a statistical-based approach, *TF-IDF* fails to exhibit comparative performance to *YAKE*. It is because the former one employs only two factors, namely *TF* and *IDF*, whereas, the latter consider multiple relevant features including casing, word position, frequency, and relatedness to context for ranking the keyphrases.

In general, all the graph-based techniques exhibit inferior performance than *YAKE* and *TeKET* since they have a number of shortcomings including not being good in capturing the cohesiveness of words, subject to clustering errors, experiences error propagation problem, etc. Among them, *MR* performs the best in terms of all the considered metrics. The highest *F1* achieved by *MR* is only 5.2 for top-5 keyphrases and the lowest is 4.0 for top-15 keyphrases. It is mainly due to resolving the error propagation problem that exists in *TR*. Now, *TR* is the closest competitor to *MR* with the highest *F1* score of 4.7 for top-5 keyphrases. Among the rest, the performance of *PR* and *SR* is negligible.

TeKET Is Language Independent

To demonstrate the language independence of the proposed technique, a *German Research Article* dataset [51] has been employed (see the "Corpora Details" section for details). Necessary adaptations have been made to all relevant techniques including the proposed one to ensure the experiment's successful run. The obtained results have been reported in Table 14.

We can see in the reported results that the performance of *PR*, *SR*, and *TF-IDF* are negligible (i.e., almost zero). In the case of *SR*, since it has a tendency of assigning higher scores to long but non-significant keyphrases, it fails to find gold standard keyphrases from the top ranked ones. On the other hand, in the case of *PR*, due to its inaccurate weight assignments to various keyphrases belonging to a single topic, it also fails to score better. Again, in the case of *TF-IDF*,

Table 14 Performance of different unsupervised machine learning–based keyphrase extraction techniques on German Research Article dataset

Approach	Technique	Top 5			Top 10			Top 15		
		P	R	F1	P	R	F1	P	R	F1
Graph-based	TopicRank	6.0	6.5	6.2	5.0	9.5	6.4	3.9	11.5	5.8
	PositionRank	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	SingleRank	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	MultipartiteRank	8.0	9.0	8.5	6.0	12.0	7.8	6.0	18.0	8.9
Statistical-based	TF-IDF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	YAKE	8.0	9.0	8.4	4.0	9.0	5.5	2.6	9.0	4.1
Tree-based (proposed)	TeKET	8.8	9.4	9.1	5.5	11.6	7.8	4.4	13.8	6.7

it fails due to receiving higher ranks by the inferior keyphrases. In the case of *TR*, it exhibits relatively better performance as it takes topical coverage into account. The highest *F1* score it receives is 6.4 for top-10 keyphrases, which is higher than that of *YAKE*. The latter only receives higher *F1* score for top-5 keyphrases.

For this corpus, *TeKET* and *MR* perform comparably. For instance, although *TeKET* outperforms *MR* for top-5 keyphrases, it suffers a defeat for top-15 keyphrases. For top-10 keyphrases, both exhibit identical performance. The reason for *MR*'s better performance is due to solving the error propagation problem of the *TR* technique. Based on the aforementioned discussions and the reported results in the table, we can conclude that the proposed technique is also language independent.

Conclusions

In this paper, a new unsupervised automatic keyphrase extraction technique, named *Tree-based Keyphrase Extraction Technique (TeKET)* is proposed, which is domain and language independent, employs limited statistical knowledge, but no train data are required. It introduces a new variant of binary tree, called *KeyPhrase Extraction (KePhEx) tree*), for extracting final keyphrases from candidate keyphrases. The proposed tree is formed using a candidate keyphrase and processed with other similar candidate keyphrases of a certain root. In the end, the tree is pruned before extracting final keyphrases employing the *mamu* value, which also provides flexibility during keyphrase extraction process from the tree. Afterwards, all the final keyphrases are extracted from the resultant tree and they are ranked taking *TF* and μ factors into account, and then, sorted. At last, top-*N* keyphrases are selected from the sorted list and returned.

Our proposed technique is compared with other prominent unsupervised keyphrase extraction techniques on a uniform experimental setup. The results are acquired for three datasets, namely *SemEval-2010*, *Theses100*, and German Research Article to evaluate their performance. According to the acquired results, TeKET outperforms the rest of the compared techniques in terms of F1 scores for all considered parameters. They also establish the claim of domain and language independence of the proposed technique.

Acknowledgements The authors would like to thank Prof. Min-Yen Kan from the National University of Singapore for providing the dataset.

Author Contributions This work was carried out in close collaboration among all authors. S.A. conceived the method and experiments. G.R. and S.A. implemented and conducted the experiments. M.M. contributed to the experiments and in analyzing the results. K.Z.Z. and M.M.R. analyzed the results. All authors have contributed to writing the paper.

Funding Information This work has been financially supported in part by the RDU project under Grant No. RDU180359 and RDU182201-3.

Compliance with Ethical Standards

Conflict of Interests The authors declare that they have no conflict of interest.

Ethical Approval This article does not contain any studies with human participants or animals.

Informed Consent As this article does not contain any studies with human participants or animals, the informed consent is not applicable.

References

1. Adeniyi D, Wei Z, Yongquan Y. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. Appl Comput Inform. 2016;12(1):90–108.
2. Arampatzis A, Tsoris T, Koster CHA, Weide TPVD. Phrase-based information retrieval. Inf Process Manag. 1998;34(6):693–707.
3. Bennani-Smires K, Musat C, Hossmann A, Baeriswyl M, Jaggi M. Simple unsupervised keyphrase extraction using sentence embeddings. arXiv:180104470. 2018.
4. Bird S, Loper E. NLTK: the natural language toolkit. In: Proc ACL; 2004. p. 214–217.
5. Boudin F. pke: an open source python-based keyphrase extraction toolkit. In: Proc COLING; 2016. p. 69–73.
6. Boudin F. Unsupervised keyphrase extraction with multipartite graphs. In: Proc NAACL: Human language technologies; 2018. p. 667–672.
7. Bougouin A, Boudin F, Daille B. Topicrank: Graph-based topic ranking for keyphrase extraction. In: Proc IJCNLP; 2013. p. 543–551.
8. Brin S, Page L. The anatomy of a large-scale hypertextual web search engine. Comput Netw ISDN Syst. 1998;30(1-7):107–117.
9. Brown JS, Duguid P. Organizing knowledge. California Management Review. 1998;40(3):90–111.
10. Campos R, Mangaravite V, Pasquali A, Jorge AM, Nunes C, Jatowt A. A text feature based automatic keyword extraction method for single documents. In: Proc ECIR; 2018. p. 684–691.
11. Campos R, Mangaravite V, Pasquali A, Jorge AM, Nunes C, Jatowt A. Yake! collection-independent automatic keyword extractor. In: Proc ECIR; 2018. p. 806–810.
12. Chor B, Gilboa N, Naor M. Private information retrieval by keywords. Princeton: Citeseer; 1997.
13. Chowdhury GG. Natural language processing. Wiley Online Library. 2003;37:51–89.

14. Das AS, Datar M, Garg A, Rajaram S. Google news personalization: scalable online collaborative filtering. In: Proc WWW; 2007. p. 271–280.
15. Dashtipour K, Poria S, Hussain A, Cambria E, Hawalah AY, Gelbukh A, Zhou Q. Multilingual sentiment analysis: state of the art and independent comparison of techniques. Cogn Comput. 2016;8(4):757–771.
16. DeWilde B. Intro to automatic keyphrase extraction. http://bdewilde.github.io/blog/2014/09/23/intro-to-automatic-keyphrase-extraction/. 2014.
17. El-Beltagy SR, Rafea A. Kp-miner: a keyphrase extraction system for english and arabic documents. Inf Syst. 2009;34(1):132–144.
18. El-Beltagy SR, Rafea A. Kp-miner: Participation in semeval-2. In: Proc SemEval; 2010. p. 190–193.
19. Florescu C, Caragea C. Positionrank: an unsupervised approach to keyphrase extraction from scholarly documents. In: Proc. ACL; 2017. p. 1105–1115.
20. Franceschini F, Maisano D, Mastrogiacomo L. Empirical analysis and classification of database errors in scopus and web of science. J Informetr. 2016;10(4):933–953.
21. Frank E, Paynter GW, Witten IH, Gutwin C, Nevill-Manning CG. Domain-specific keyphrase extraction. In: Proc. IJCAI; 1999. p. 668–673.
22. Freitag D. Machine learning for information extraction in informal domains. Mach learn. 2000;39(2-3):169–202.
23. Hardeniya N, Perkins J, Chopra D, Joshi N, Mathur I. Natural language processing: python and NLTK. Birmingham: Packt Publishing Ltd; 2016.
24. Hariharan R, Hore B, Li C, Mehrotra S. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In: Proc. SSBDM; 2007. p. 16–16.
25. Hasan KS, Ng V. Automatic keyphrase extraction: a survey of the state of the art. In: Proc. ACL; 2014. p. 1262–1273.
26. Herrera JP, Pury PA. Statistical keyword detection in literary corpora. Eur Phys J B. 2008;63(1):135–146.
27. Hoare CAR. Quicksort. The Computer Journal. 1962;5(1):10–16. https://doi.org/10.1093/comjnl/5.1.10.
28. Huang F, Zhang Y, Vogel S. Mining key phrase translations from web corpora. In: Proc. HLT; 2005. p. 483–490.
29. Hulth A. Improved automatic keyword extraction given more linguistic knowledge. In: Proc. EMNLP; 2003. p. 216–223.
30. Jean-Louis L, Zouaq A, Gagnon M, Ensan F. An assessment of online semantic annotators for the keyword extraction task. In: Proc. PRICAI; 2014. p. 548–560.
31. Kantrowitz M, Mohit B, Mittal V. Stemming and its effects on tfidf ranking. In: Proc. SIGIR; 2000. p. 357–359.
32. Karaa WBA, Gribâa N. Information retrieval with porter stemmer: a new version for english. In: Advances in computational science, engineering and information technology. Springer; 2013. p. 243–254.
33. Kim SN, Medelyan O, Kan MY, Baldwin T. Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In: Proc. SemEval; 2010. p. 21–26.
34. Kim SN, Medelyan O, Kan MY, Baldwin T. Automatic keyphrase extraction from scientific articles. Lang Resour Eval. 2013;47(3):723–742.
35. Kononenko I. Machine learning for medical diagnosis: history, state of the art and perspective. Artif Intell Med. 2001;23:89–109.
36. Kosala R, Blockeel H. Web mining research: a survey. ACM SIGKDD Explor Newsl. 2000;2(1):1–15.
37. Kotler P, Roberto EL. Social marketing. Strategies for changing public behavior. New York: Free Press; 1989.
38. Kuchling A. Regular expression howto. https://docs.python.org/3/howto/regex.html. 2018.
39. Lawrence S, Giles CL, Bollacker K. Digital libraries and autonomous citation indexing. Computer. 1999;32(6):67–71.
40. Litvak M, Last M. Graph-based keyword extraction for single-document summarization. In: Proc. MMIES; 2008. p. 17–24.
41. Manevitz LM, Yousef M. One-class svms for document classification. J Mach Learn Res. 2001;2(Dec):139–154.
42. McCallum A, Nigam K, et al. A comparison of event models for naive bayes text classification. In: AAAI-98 Workshop learn. text categ.; 1998. p. 41-48.
43. Merrouni ZA, Frikh B, Ouhbi B. Automatic keyphrase extraction: an overview of the state of the art. In: Proc. CiST; 2016. p. 306–313.
44. Mihalcea R, Tarau P. Textrank: Bringing order into text. In: Proceedings of the 2004 conference on empirical methods in natural language processing; 2004. p. 404–411.
45. Ohsawa Y, Benson NE, Yachida M. Keygraph: automatic indexing by co-occurrence graph based on building construction metaphor. In: Proc. ADL; 1998. p. 12–18.
46. Page L, Brin S, Motwani R, Winograd T. The pagerank citation ranking: bringing order to the web. Stanford InfoLab, Tech rep. 1999.
47. Paik JH, Pal D, Parui SK. A novel corpus-based stemming algorithm using co-occurrence statistics. In: Proc SIGIR; 2011. p. 863–872.
48. Pandarachalil R, Sendhilkumar S, Mahalakshmi G. Twitter sentiment analysis for large-scale data: an unsupervised approach. Cognitive Computation. 2015;7(2):254–262.
49. Pudota N, Dattolo A, Baruzzo A, Ferrara F, Tasso C. Automatic keyphrase extraction and ontology mining for content-based tag recommendation. Int J Intell Syst. 2010;25(12):1158–1186.
50. Rabby G, Azad S. Automatic keyphrase extraction. https://drive.google.com/drive/folders/1e2UrDtYqRAjAE5hso4oXobX_Djuo_VUW. 2019.
51. Rabby G, Azad S. Datasets - german papers. https://github.com/corei5/TeKET/tree/master/Data%20set/German%20Papers. 2019.
52. Rabby G, Azad S, Mahmud M, Zamli KZ, Rahman MM. A flexible keyphrase extraction technique for academic literature. In: Procedia Comput Sci; 2018. p. 653–663.
53. Reilly RG, Sharkey N. Connectionist approaches to natural language processing. Abingdon: Routledge; 2016.
54. Ricci F, Rokach L, Shapira B. Introduction to recommender systems handbook. In: Recommender systems handbook. Springer; 2011. p. 1–35.
55. Rowley J, Hartley R. Organizing knowledge: an introduction to managing access to information. Abingdon: Routledge; 2017.
56. Salton G, Buckley C. Term-weighting approaches in automatic text retrieval. Inf Process Manag. 1988;24(5):513–523.
57. Seuring S, Gold S. Conducting content-analysis based literature reviews in supply chain management. Supply Chain Manag: Int J. 2012;17(5):544–555.
58. Siddiqi S, Sharan A. Keyword and keyphrase extraction techniques: a literature review. International Journal of Computer Applications. 2015;109(2):18–23.
59. Steinbach M, Karypis G, Kumar V, et al. A comparison of document clustering techniques. In: KDD Workshop on text mining, boston; 2000. p. 525–526.
60. Sterckx L, Demeester T, Deleu J, Develder C. Topical word importance for fast keyphrase extraction. In: Proc WWW; 2015. p. 121–122.
61. Sterckx L, Demeester T, Deleu J, Develder C. Creation and evaluation of large keyphrase extraction collections with multiple opinions. Lang Resour Eval. 2018;52:503–532.
62. Sugiyama K, Kan MY. Scholarly paper recommendation datasets. http://www.comp.nus.edu.sg/~sugiyama/SchPaperRecData.html. 2018.
63. Thomas JR, Bharti SK, Babu KS. Automatic keyword extraction for text summarization in e-newspapers. In: Proc ICIA; 2016. p. 86–92.

64. Tixier A, Malliaros F, Vazirgiannis M. A graph degeneracy-based approach to keyword extraction. In: Proc EMNL; 2016. p. 1860–1870.
65. Tomokiyo T, Hurst M. A language model approach to keyphrase extraction. In: Proc ACL; 2003. p. 33–40.
66. Tümer D, Shah MA, Bitirim Y. An empirical evaluation on semantic search performance of keyword-based and semantic search engines: Google, yahoo, msn and hakia. In: Proc ICIMP; 2009. p. 51–55.
67. Vencovsky F, Lucas B, Mahr D, Lemmink J. Comparison of text mining techniques for service aspect extraction. In: Proc ECSM; 2017. p. 297–307.
68. Vállez M, Pedraza-Jiménez R, Codina L, Blanco S, Rovira C. A semi-automatic indexing system based on embedded information in html documents. Libr Hi Tech. 2015;33(2):195–210.
69. University of Waikato NZ. Datasets of automatic keyphrase extraction. https://github.com/LIAAD/KeywordExtractor-Datasets#theses. 2019.
70. Wan X, Xiao J. Collabrank: towards a collaborative approach to single-document keyphrase extraction. In: Proc COLING; 2008. p. 969–976.
71. Wang H, Xu F, Hu X, Ohsawa Y. Ideagraph: a graph-based algorithm of mining latent information for human cognition. In: Proc SMC; 2013. p. 952–957.
72. Wang J, Liu J, Wang C. Keyword extraction based on pagerank. In: Proc PAKDD; 2007. p. 857–864.
73. Wang QF, Xu M, Hussain A. Large-scale ensemble model for customer churn prediction in search ads. Cognitive Computation. 2019;11(2):262–270.
74. Wu Z, Zhu H, Li G, Cui Z, Huang H, Li J, Chen E, Xu G. An efficient wikipedia semantic matching approach to text document classification. Inf Sci. 2017;393:15–28.
75. Xu C, Wu Y, Liu Z. Multimodal fusion with global and local features for text classification. In: Proc ICONIP; 2017. p. 124–134.
76. Xu W, Liu X, Gong Y. Document clustering based on non-negative matrix factorization. In: Proc SIGIR; 2003. p. 267–273.
77. Yoo SC, Eastin MS. Contextual advertising in games: impacts of game context on a player's memory and evaluation of brands in video games. J Mark Commun. 2017;23(6):614–631.
78. Zhai C, Lafferty J. A study of smoothing methods for language models applied to ad hoc information retrieval. In: ACM SIGIR Forum; 2017. p. 268-276.
79. Zhang K, Xu H, Tang J, Li J. Keyword extraction using support vector machine. In: Proc WAIM; 2006. p. 85–96.