IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 21, 2021, accepted November 8, 2021, date of publication November 18, 2021,
date of current version December 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3129187

Efficient Sentiment-Aware Web Crawling Methods for Constructing Sentiment Dictionary

BYUNG-WON ON[1], JUN-YOUNG JO[1], HYUNKWANG SHIN[2], JANGWON GIM[1], GYU SANG CHOI[2], (Member, IEEE), AND SOO-MOK JUNG[3]

[1]Department of Software Convergence Engineering, Kunsan National University, Gunsan-si, Jeollabuk-do 54150, South Korea
[2]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan-si, Gyeongsangbuk-do 38541, South Korea
[3]Division of Computer Science and Engineering, Sahmyook University, Seoul 01795, South Korea

Corresponding authors: Jangwon Gim (jwgim@kunsan.ac.kr) and Gyu Sang Choi (castchoi@ynu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through MSIT under Grant NRF-2019R1F1A1060752 and Grant NRF-2021R1A6A1A03039493.

ABSTRACT In traditional web crawling, all web pages crawled are first stored to databases. As a result, this approach can store unnecessary web pages and requires additional running time for the construction of a sentiment dictionary in a particular domain because sentiment words should be identified by scanning all web pages in the database. To address these problems, we first define the sentiment-aware web crawling problem and then propose two hash-based methods for the implementation. One is based on hash join and the other is bucket-sorted hash join. In particular, we propose a novel bucket-sorted hash join for the efficient sentiment-aware web crawling method. Our experimental results show that the proposed web crawling method using bucket-sorted hash join outperforms existing web crawling methods by significantly reducing the running time and storage space. In the proposed method, the time taken to execute the sentiment-aware task per web page is 0.016 seconds and the database space can be saved by 59% compared to the existing web crawling methods.

INDEX TERMS Hash join, sentiment analysis, sentiment lexicon, web crawling.

I. INTRODUCTION

In the past, most data mining techniques that exploit useful information hidden in *objective* facts have been widely used, but recent studies on analyzing and aggregating *subjective* information of people by the development of smart devices and social network services have been treated to be important. In this sense, sentiment analysis or opinion mining [1] [2] [3] has been actively studied from the past to the present. Recently, public opinion and market research are no longer surveyed in the traditional way, but rather relevant data are automatically collected from the web and then pros and cons of the questionnaire are summarized through sentiment analysis.

To conduct market research in a certain domain, one of challenging research problems is to automatically identify domain-dependent sentiment words [4]. Furthermore, Sentiment Analysis using deep learning requires a large-scale training set of labeled sentiments. To solve this problem, an existing sentiment lexicon was used for automatic generation of a large-scale training set labeled with sentiments [5]. Opinion Summarization is one of the most important problems in sentiment analysis and market research areas recently [6]. Given a large collection of review text documents about a particular product in automobiles or smartphones as input, state–of–the–art opinion summarization methods usually provide (1) a few main aspects, (2) pros and cons per aspect, and (3) extractive/abstractive summaries in favor of/against each aspect.

As such, the sentiment analysis techniques can be widely applied in many diverse applications, making it an important technology in data mining. In particular, a sentiment lexicon is one of the key components in sentiment analysis. If there are abundant sentiment vocabularies in the sentiment lexicon, it will be possible to obtain high accuracy even if any sentiment analysis method is used. Recently, however, social network services and chat applications have become widespread and new sentiment words, abbreviations, and even emoticons are increasing rapidly. Even if we build an almost perfect sentiment lexicon with time and cost, it is necessary to add new sentiment vocabularies in many different domains over time. To add new sentiment words and phrases to the existing

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

sentiment lexicon, web crawlers periodically crawl and collect new text documents from the web [7].

Algorithm 1 Traditional Crawling Algorithm as Baseline

```
1  Input: SEED_URL;
2  enqueue(URL_queue, SEED_URL);
3  if not length(URL_queue) > 99 then
4      while not empty(URL_queue) do
5          URL = dequeue(URL_queue);
6          page = crawl_page(URL);
7          enqueue(crawled_pages, (URL, page));
8          URLlist = extractURLs(page);
9          for u in URLlist do
10             enqueue(links, (URL, u));
11             if u not visit URL_queue and (u,-) not in crawled_pages then
12                 enqueue(URL_queue, u);
13         ReorderQueue(URL_queue);
```

The basic process of web crawling is shown in Algorithm 1. Web crawlers first download initial web pages by a list of seed URLs and then URLs in each web page downloaded are added to the queue. The URLs in the queue are rearranged according to their importance. The URL with the highest priority is deleted from the queue and the corresponding web page is downloaded from the web. The same process is repeated until the queue is empty. In this manner, traditional web crawling methods are likely to store the downloaded web pages in a file system in which all web pages are scanned when a sentiment dictionary for a particular domain is constructed. In this approach, two major problems occur. Because some web pages contain abundant sentimental vocabularies, while other web pages may have few sentimental vocabularies, one problem is that the storage space is excessively wasted by storing all downloaded web pages that contain little sentimental vocabulary. The other problem is that it takes a lot of time to build a sentiment dictionary while all the web pages stored in the file system are scanned [8].

To address these problems, in this work, we propose a new sentiment–aware web crawling approach that filters unnecessary web pages during web crawling. When our proposed web crawler tries to download a web page, it identifies whether or not the web page contains a lot of sentiment vocabularies. As the baseline method, all words on the web page need to be scanned to check if each word exists in the sentiment dictionary. Here, we assume that a sentiment lexicon is given in advance. If a web page contains a lot of sentiment words even though some sentiment words do not exist in the given sentiment lexicon, it may be more valuable than other web pages including little sentiment words. Since the baseline method matches all words with all sentiment words in the sentiment dictionary, it will take a long time. For example, in case that a web page W has n words and the number of the vocabularies in the sentiment dictionary is m, the time

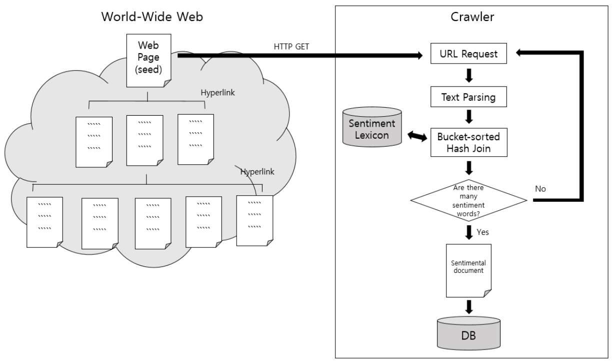


FIGURE 1. Sentiment–aware web crawler.

complexity is $O(knm)$, where k stands for the number of web pages downloaded by web crawlers. This means that the traditional web crawling method needs the intelligent process to *quickly* find out if a web page has many sentiment vocabularies. The goal of the proposed web crawling algorithm is to minimize the execution time of finding web pages with many sentiment words.

For this, we first propose a sentiment-aware web crawling method. Figure 1 illustrates the overall flow chart of the proposed approach. Then, we present two hash-based methods for implementation. One is based on hash join and the other is bucket-sorted hash join. In particular, we propose a new bucket-sorted hash join method that is faster than the existing hash join method. Our experimental results show that the proposed web crawling method outperforms the traditional web crawling method by significantly reducing the running time and storage space. Please refer to the Experimental Results section for details.

The contributions of our work are as follows:

- In this work, we first raise the necessity of the sentiment-aware web crawling method rather than the existing web crawling method to build a sentiment dictionary by periodically collecting new words of sentiment and defining the problem formally. Moreover, the proposed novel algorithm that overcomes major drawbacks such as applying insufficient sentiment words in web crawler methods for sentimental analysis or not considering various sentiment classifications expressed in vocabulary. To the best of our knowledge, this is the first study in web crawling.
- Our proposed method, that is, the sentiment-aware web crawling method, can save the space of the database and the time to build the sentiment dictionary compared to the existing web crawling method. However, it takes additional time to determine whether a web document contains many sentiment words. To minimize this additional time, we propose hash join based methods.
- For efficient sentiment-aware task, we propose a solution that fits our problem by borrowing the existing hash join algorithm. We call this approach sentiment-aware web crawling based on hash join. Besides, we propose

a new bucket-sorted hash join scheme that is faster than the existing hash join scheme, and apply it to our problem.
- Our experimental results show that the proposed web crawling method outperforms traditional web crawling method by significantly reducing the running time and storage space. In addition, the proposed bucket-sorted hash join method is faster than the hash join based method in the sentiment-aware task in web crawling.

The remainder of this article is organized as follows: First, we introduce previous studies related to this work in Section II. Then, we formally define our problem in web crawling in Section III. Next, we described the detailed algorithms for the proposed sentiment-aware web crawling based on both hash join and bucket-sorted hash join in Section IV. In the experimental set-up and result sections, we validate the proposed hash join based methods, compared to the traditional web crawling method. In Section VI, we deal with the scope, assumption, and limitation of our proposed method in detail. Finally, we summarize our work followed by the future research direction in Section VII.

II. RELATED WORK

Web crawlers for information retrieval from web sites are classified into four categories: focused, distributed, incremental and hidden web crawlers. Among these strategies, a focused web crawler approach is used to prioritize crawled pages and collect web pages that contain some target information through the hyperlink navigation process. Therefore, focused web crawler is used in web content collection approach targeting keywords, miscellaneous content filtering approach, ontology-based content collection approach, and opinion mining-based approach considering various content expression methods in web pages [9]. Also, according to the collection purpose of the web crawler, it is classified into URL based web crawler and content-based web crawler [10]. Content based approach is classified into two types: a surface web approach that collects superficially displayed web pages and a deep web approach that collects hidden contents. To collect more valuable information in the recent web environment in which various contents are shared, it is also necessary to classify the collection object of the collector in detail. Therefore, in this paper, we define subcategories of the surface web approach as keyword-based web crawlers, multi-resource-based web crawlers, semantic web crawlers, and sentiment analysis crawlers. Figure 2 shows the classification of web crawler.

Through the representative characteristics of openness and accessibility of the World Wide Web, users are experiencing the benefits of creating and sharing many contents on web pages. However, since web pages containing much malicious content have a detrimental effect on users and systems, methods have been proposed to filter such content and prevent access through the web crawler system. Keyword-based approach is used to retrieve the information

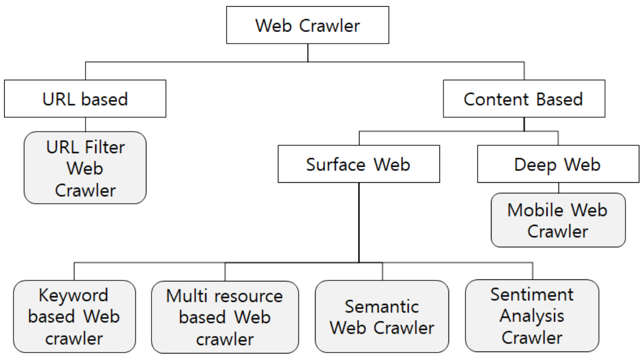


FIGURE 2. Classification of web crawler.

from the web [11]–[13]. [13] proposed a web crawler system that classifies web pages based on keywords contained in web pages. By extracting keywords representing age groups such as "kids," "parenting," and "drowning," it classifies whether the content of a web page is appropriate for a specific age(5-9 years) and blocks access to inappropriate information. At this time, WordNet thesaurus is applied to calculate the similarity between keywords and categories included in the content. Therefore, the system accurately classifies web pages containing malicious advertisements and web pages that do not fit the age classification targeting large-scale web pages. As a result, it is possible to identify and classify harmful web pages through content filtering of web pages, but there is a limitation in that keywords belonging to a specific category (such as age) must be defined in advance.

In addition, since web pages contain information in various types of resources such as images and videos, and text content, a focused web crawler targeting these various web resources has been proposed. [14] proposed a method for classifying web pages based on text and images included in web pages based on artificial neural network algorithms. The proposed algorithm extracts features including objectionable words, sentences having objectionable words, images having objectionable words as titles through the feature extractor module. Learning the extracted features as input data of the Advanced Quantum-based Neural Network (AQNN) Classifier classifies whether the web page is objectionable or non-objectionable. If an objectionable is found on a website, the website and URL are added to the database and managed. Therefore, when users try to access any objectionable website, the number of objectionable URLs' database increases and the website may be blocked. As a result, it proves that web content filtering is possible through machine learning using text and images collected by a web crawler. However, since the proposed method focuses only on whether the feature is objectionable or not, it is limited to sentiment classification of websites where sentiment words and sentences frequently appear, such as product and movie reviews and news article comments.

Semantic web pages are being created so that humans and machines can understand web resources included in web pages. Web resources are defined using standardized syntax such as Web Ontology Language (OWL) and Resource Description Framework (RDF) so that crawlers can understand them. As a result, the recall and accuracy of the web crawler show much better performance than the general web type. Ontology-based crawlers have been proposed to collect ontology included in these semantic web pages [15] [16] proposed a distributed web crawler system that understands the theme of content included in web pages and supports the decision-making required for crawling page selection. Web resources included in semantic web pages are analyzed through the ontology analyzer module, enabling the crawling of web pages highly related to specific search keywords. Therefore, the robustness and reliability of the web crawler can be improved because web pages, including web resources expressed in RDF/OWL, can be handled more efficiently. However, although the ontology-based crawler can help the understanding and identification of web resources, there is an apparent weakness in that the performance of the web crawler is reduced for the resources included in the traditional web page, which still occupies a large proportion.

From an industry point of view, web pages are actively used for product advertisements and product introductions. In addition, product users provide user opinions and exchange information through web pages based on product information. At this time, users obtain the information they want through questions and answers about the product information and actively express their opinions using various emotional vocabulary. Therefore, to quickly and accurately analyze the question and answer sentences included in the web page, extracting the response sentence for a specific question sentence plays a vital role in the opinion mining field. [17] proposed a method to automatically extract and crawl texts for questions and answers included in websites. Extracts original data from web pages and extracts keywords through pre-processing such as word separation and morphological analysis. The extracted keyword is calculated using the thesaurus to calculate the similarity with the question sentences. Therefore, it is possible to extract the most similar question sentence to the corresponding keyword and extract the answer sentence, including the corresponding keyword as a pair with the question sentence. Therefore, the proposed crawler system can help construct the set of question and answer sentences. However, the proposed method calculates the similarity with the question sentences only for the vocabulary constructed in the thesaurus and collects the answer sentences. Therefore, there is a limit to data extraction and crawling, including sentiment words, which have different expression methods according to changes in trends, and words with various morphological changes.

Sentiment analysis deals with computational treatment of sentiment, opinion and subjectivity in text. However, it is difficult to draw a positive or negative conclusion from the various opinions. Recently, in an environment where opinions among users through huge amounts of web pages are actively exchanged, an efficient analysis and mining method of website data is needed for accurate sentiment analysis. [18] proposed an advanced sentiment classification method in financial news articles and proposed a market price prediction method. In this work, customized crawlers extract financial news articles for the target region(industry) from multiple sources. Syntactic analysis and sentiment classification are performed from all web documents, and market prices are also collected. Each sentence is extracted from all collected web documents, and the sentiment score is calculated and accumulated through parsing the sentences. As a result, based on the correlation between news sentiments and market prices, market price prediction is to be facilitated. However, since it is necessary to collect, store, and parse all sentences included in all the news articles, it takes much time to analyze sentiment analysis for a large-scale website. The duplication of the collected data is a significant drawback. Figure 10 shows the comparison results for methodologies of the content-based web crawler.

III. PROBLEM STATEMENT

To build a sentiment dictionary, web crawlers first collect a large amount of web pages and store them in a database. After that, the entire database is scanned and the sentiment vocabulary is identified. The problem of this traditional approach is that web documents that contain little sentiment vocabulary are also stored together, which wastes storage space in the database and increases the unnecessary time to build the sentiment dictionary because the entire database must be scanned.

To solve this problem, we propose a web crawler using hash join during the crawling process to quickly identify and selectively store only documents containing a large number of positive and negative vocabularies before storing the web document to the database. Throughout this article, we will call such a crawler a *sentiment–aware* web crawler.

In this work, for formal problem definition, assume that a web crawler $\mathbb{C}$ visits a web page W and can use a sentiment lexicon D that is available in public. Specifically, W is a set of terms i.e., $W = \{t_1, t_2, \ldots, t_n\}$ and D is a set of sentiment terms i.e., $D = \{s_1, s_2, \ldots, s_m\}$. Unlike the existing web crawlers, the sentiment-aware web crawlers must perform the following task in Algorithm 2.

Please note that this task is based on the hypothesis that W, stored in the database, also includes many additional sentiment vocabularies disappearing in D.

As one major weak point in the sentiment-aware task, it takes additional time unlike the existing web crawlers. In the formulation, when $\mathbb{C}$ visits k web pages, the time complexity is $O(knm)$. Compared to the existing web crawling method, it takes more $O(nm)$. In order to minimize the time to perform the sentiment-aware task, we use a hash join method in Algorithm 3 rather than the pair-wise matching in Algorithm 2 and further propose an efficient method by improving the existing hash join method. In the efficient sentiment-aware task, the time complexity

Algorithm 2 Sentiment-Aware Task

```
matching = 0;
for $t \in W$ do
    for $s \in D$ do
        if $t == s$ then
            matching++;
$\alpha = \frac{\text{mathcing}}{n}$;
if $\alpha \geq$ Threshold $\theta$ then
    Store $W$ to the database;
```

Algorithm 3 Efficient Sentiment-Aware Task

```
for $s \in D$ do
    Hash function
    $h(s) \rightarrow$ one of buckets in Hash Table $H$;
matching = 0;
for $t \in W$ do
    $h(t) \rightarrow$ B[i] in $H$;
    for $t == s \in B[i]$ do
        matching++;
$\alpha = \frac{\text{mathcing}}{n}$;
if $\alpha \geq$ Threshold $\theta$ then
    Store $W$ to the database;
```

is $O(m + n|B|) \approx O(n|B|)$, where $|B|$ is the number of the sentiment terms per bucket and $|B| \ll O(m)$.

The aim of our approach is to reduce the running time in the sentiment-aware task. By collecting only web documents rich in sentiment vocabulary through the sentiment-aware task, database space is saved, and there is no need to scan the entire database, making it easy to build a sentiment lexicon. It is also essential to study in depth the trade-off of the advantages and disadvantages arising from performing the sentiment-aware task.

IV. MAIN PROPOSAL: EFFICIENT SENTIMENT-AWARE TASKS IN WEB CRAWLERS

To develop the efficient sentiment-aware crawling methods, we propose two hash-based approaches. One is based on hash join and the other is bucket-sorted hash join. In the first approach, we merely apply existing hash join scheme to our problem to see if it is working on the sentiment-aware crawling task. On the other hand, in the second approach, we first propose a new bucket-sorted hash join scheme that is more efficient than the existing hash join scheme, and then we apply the proposed bucket-sorted hash join scheme to the sentiment-aware crawling task. For the details, we will describe the sentiment-aware web crawling method using hash join in Section IV-A and the sentiment-aware web crawling method using bucket-sorted hash join in Section IV-B.

A. SENTIMENT-AWARE TASK USING HASH JOIN

1) MAIN CONCEPT OF HASH JOIN

In this section, we assume that tables R and S will be joined with hash join, to explain them easily. The hash join scheme has two steps: build and probe. In the build step, it builds a hash table for the table with the smaller number of tuples. In the probe step, it probes the constructed hash table using the other table. This can be accomplished with all these operations in memory [19] if the tuples of these tables are small. However, if these tables are too big to fit in the memory, they require expensive disk I/O operations. The early hash join [20] is based on symmetric hash join and uses a single hash table for each input. It also consists of reading and flushing. This join algorithm dynamically customizes the balance between initial performance results and minimum execution time. The grace hash join scheme [21], [22] was proposed to reduce the number of expensive disk I/O operations. The scheme consists of two phases. In the first phase, table R is divided into multiple partitions, and each partition is read from the disk and hashed with a hash table. In this article, a directory (i.e., hash table) consists of multiple directory entries and resides in main memory. Buckets are also allocated in the memory space to hold corresponding records by the hashed value. Each directory entry can have multiple linked buckets, and a hashed record is assigned to each corresponding directory entry. If the allocated memory space for a certain directory entry in the hash table is full, all records of the allocated memory space are written back to the disk and deleted, and the algorithm then continues to process all the tuples in table R. In the second phase, it builds the hash table with a different hash function after reading the tuples in table R from the storage. Then, it probes the built hash table for table R with a tuple of table S, after reading the tuple of table S from the storage. Thus, the grace hash join scheme can significantly reduce the number of disk I/Os because it accesses tables R and S with block I/Os.

The hybrid hash join scheme [23], [24] is a variant of the grace hash join and is intended to utilize memory more efficiently. In hybrid hash join, the first partition of table R is kept in the memory and the hash table is built during the first phase, without writing back to the storage. When hashing the first partition of table S, the algorithm directly probes the hash table of the first partition of table R in the memory. Except for the first partition of tables R and S, other partitions will be processed in exactly the same way as the grace hash join. Thus, the hybrid hash join scheme can eliminate the disk I/Os for the first partition.

2) SENTIMENT-AWARE TASK USING HASH JOIN

In the previous section, we describe the main concept of hash join schemes like grace hash join and hybrid hash join. We borrow the main algorithm of the hash join schemes to apply for our problem. In this work, the input data of the hash join scheme are: One table is a set of the terms in W

and the other is a set of the sentiment terms in D. This is, $W = \{t_1, t_2, \ldots, t_n\}$ and $D = \{s_1, s_2, \ldots, s_m\}$. The scheme consists of two phases. In the first phase, each $s \in D$ is hashed into one of buckets $B^D = \{B^D_1, B^D_2, \ldots, B^D_l\}$ in a hash table H^D through a hash function $h_1()$. For example, $h_1(s_5) \rightarrow B^D_2$. The hash table H^D is a set of directory entries, each of which has multiple buckets. If the first bucket of the directory entry does not have enough space, a new bucket is linked to the first bucket in a separate chaining method. For example, a term needs to be added to the first bucket, but if it is already full, after the second bucket is created, the term is inserted in the bucket. When the overflow bucket is also full, another overflow bucket is created and connected. In the directory entry, all buckets are linked by a linked list. In Figure 3(a), the directory entry B^D_1 in H^D has two buckets. The first bucket has two sentiment terms s_1 and s'_1 and the second bucket has two sentiment terms s_2 and s_{10}. In the figure, using $h_1()$, all terms in D are hashed into H^D. Similarly, as shown in Figure 3(b), a web document W crawled by a web crawler has five terms. Through the same hash function $h_1()$, the all terms are hashed into the hash table H^W. In the figure, three terms t_1, t'_1, and t''_1 are hashed into B^W_1; one term t_2 is hashed into B^W_2; and one term t_3 is hashed into B^W_l.

Eq 1 is the hash function used in the first phase. We call it the first hash function. Suppose that a term x is a string of length n. $x[i]$ means the byte value for the i–th character of x. The directory entry to which x should be stored in H^D is determined by $h_1(x)\%|H^D|$, where $|H^D|$ is the total number of the directory entries in H^D.

$$h_1(x) = x[0] \times 31^{(n-1)} + x[1] \times 31^{(n-2)} + \ldots + x[n-1] \tag{1}$$

The purpose of the first phase is to group similar terms from each table. For instance, in Figure 3(a), assume that two terms s_1 and s'_1 are identical in D. Likewise, three terms t_1, t'_1, and t''_1 are identical in W. In each table, through $h_1()$, similar terms are hashed into the same directory. For example, s_1, s'_1, s_2, and s_{10} are in B^D_1 as the same directory in H^D. Consequently, since the same hash function $h_1()$ is used, the words in B^D_1 and B^W_1 are likely to match each other. Therefore, in the second phase, B^D_1 does not need to compare all buckets of W, and only performs a join operation on the bucket B^W_1.

The output of the first phase is $\{(B^D_1, B^W_1), (B^D_2, B^W_2), \ldots, (B^D_l, B^W_l)\}$. For each pair, e.g., $(B^D_{i=1}, B^W_{i=1})$, $s \in B^D_{i=1}$ is first hashed into one of buckets in the hash table H^S, through the second hash function $h_2()$. As depicted in Figure 3(c), four terms s_1, s'_1, s_2, and s_{10} in B^D_1 are hashed into one of buckets in H^S. In the figure, s_1 and s'_1 are in B^S_1; s_2 is in B^S_2; and s_{10} is in B^S_l. Through this process, the hash table H^S is constructed for B^D_1. In fact, this is the build step. Next, for $t \in B^W_1$, it is hashed into one of buckets in H^S using $h_2()$. We call this process the probe step. Each term in B^W_1 is assigned to one bucket by $h_2()$. For instance, in Figure 3(d), $t_1 \in B^W_1$ is assigned to B^S_1 in H^S. Finally, the join operation is performed on (t_1, s_1) and (t_1, s'_1).

Algorithm 4 Insert Operation

```
1  Function Insert(Record R, The Bucket B)
2  while (B is full) do
3      B : = B->next;
4      if (B is equal to the last bucket) then
5          Allocate a new bucket;
6          B : = the new bucket;
7          break;
8  if (B ! = empty) then
9      A := The last record in B;
10     R := The new record;
11     while (A.Key > R.Key) do
12         Move right A in the bucket;
13         A := A's preceding record;
14     Add R into A's right side;
15 else
16     Add R into B;
```

Eq 2 is the hash function used in the second phase. We call it the second hash function. Assuming that a term x is a string of length n, $x[i]$ means the byte value for the i–th character of x. The directory entry to which x should be stored in H^S is determined by $h_2(x)\%|H^S|$, where $|H^S|$ is the total number of the directory entries in H^S.

$$h_2(x) = x[0] + x[1] + \ldots + x[n-1] \tag{2}$$

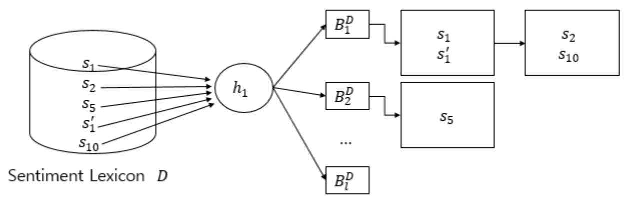
B. SENTIMENT-AWARE TASK USING BUCKET-SORTED HASH JOIN

1) MAIN CONCEPT OF BUCKET-SORTED HASH JOIN

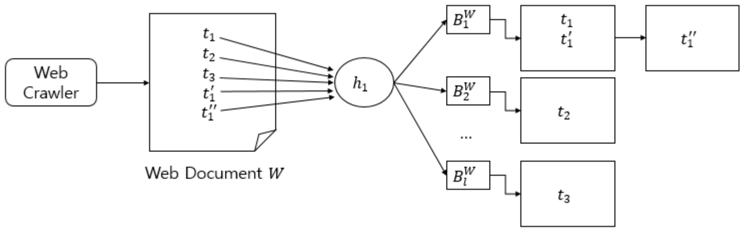
In [25], we proposed a new hash-join scheme, called bucket-sorted hash-join, to sort records only within a bucket in order to reduce the probing time, while the hybrid hash-join scheme sequentially scans records in a hash table until the corresponding records are matched. In this study, we extend it for the efficient sentiment-aware task.

Algorithm 4 presents the detailed process of an insert operation in the first and second phases in our proposed scheme. To insert a new record into a certain bucket in the proposed scheme, it first checks whether the bucket is full or not. If the bucket is full, it moves to the next bucket. If the bucket is not full, it starts comparing the record to insert with the last record in the bucket. If the key of the inserting record is smaller than the key of the last record in the bucket, it moves the last record to the next, and then compares the preceding record to the last record again. Otherwise, it simply inserts the new record next to the current record. The scheme repeats this process until a new record is properly added to a certain bucket and indicates that the new record will be inserted in the proper slot in the bucket by maintaining the ascending order among records within the bucket.

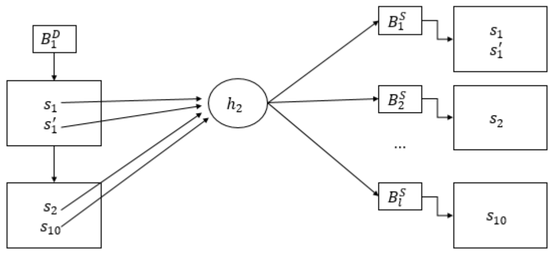
Algorithm 5 presents the detailed process of a probing operation in the second phase in our proposed scheme. In our



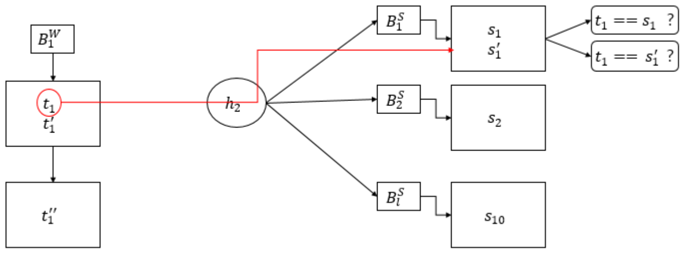
(a) The first phase: Hashing sentiment terms to the hash table H^D using h_1



(b) The first phase: Hashing terms in W to the hash table H^W using h_1



(c) The second phase: Constructing the hash table H^S from B_1^D



(d) The second phase: Probing $t \in B_1^W$ from the hash table H^S

FIGURE 3. **Efficient sentiment-aware task using hash join.**

proposed scheme, we use the binary search algorithm to find the record within a certain bucket instead of scanning records. To probe the record, the scheme visits the first bucket to find the corresponding record using the binary search algorithm. If the corresponding record is found, it combines these two records. Then, it moves to the next bucket to search, and keeps doing so until the last bucket is visited because multiple records with the same key value can exist in some tables.

Next, we explain the proposed scheme with a simple example, and Fig. 4 depicts how the bucket-sorted hash join works. While the proposed scheme is the same as that of the hybrid hash-join in the first phase, in the second phase the 0 directory entry (first partition) initially has one bucket to hold four records, 8, 16, 24 and 40, as shown in Fig. 4 (a). Within the bucket, the four records are stored in ascending order, and now a new record, 80, is inserted in this directory entry.

Because the current bucket has no space to store the new record, the new bucket is first allocated and linked, and then the record is inserted in the first slot in the second bucket. Fig. 4 (b) shows the status of the hash table after the new record (80) is added. Now, we add the new record 32 to the first directory entry in the hash table. Because the first bucket is full, the algorithm checks whether the next bucket has available space to store the new record. The record in the first slot is moved to the second slot, and the new record will be added to the first slot in the second bucket to maintain the ascending order among records within the second bucket. Fig. 4 (c) shows the status of the hash table after the new record 32 is added. Thus, a certain record can be easily searched for with our proposed scheme, using the binary search algorithm. This implies that our proposed scheme can significantly reduce the probing

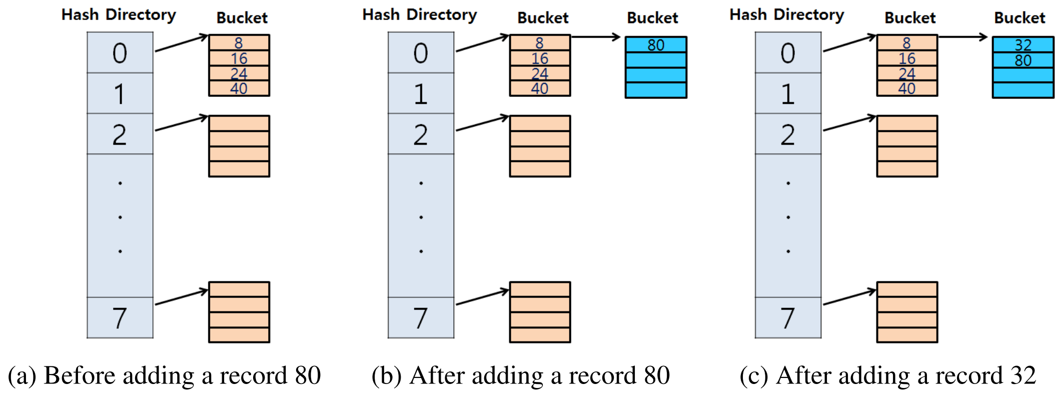

Hash Directory
Bucket
0
1
2
7
8
16
24
40
80
32
(a) Before adding a record 80
(b) After adding a record 80
(c) After adding a record 32

FIGURE 4. Bucket-sorted hash join.

Algorithm 5 Probe Operation

```
1  Function Probe(Record R, Bucket B)
2  while (B ! = The Last Bucket) do
3      BinarySearch(R,B);
4      B := B->next;
5  Function BinarySearch(Record R, Bucket B)
6  First := the first record in Bucket B;
7  Middle := the middle record in Bucket B;
8  Last := the last record in Bucket B;
9  while (First <= Last) do
10     if (Middle.Key < R.Key) then
11         First := The next record of Middle;
12     else if (Middle.Key == R.Key) then
13         Combine A and B;
14     else
15         Last := The preceding record of Middle;
16     Middle = The middle record Between First and Last;
17 return NULL;
```

time in the second phase, compared to the hybrid hash-join scheme.

2) TIME AND MEMORY SPACE COMPLEXITY

Assume that we use a separate chaining to resolve a collision in a hash table, and additional memory spaces other than the key/value pairs in the hash table directory are not needed. The cost of a hash join includes i) partitioning the relations into blocks, ii) creating a hash table entry, iii) probing with a hash table, iv) reading and writing the disk blocks, and v) moving the data to the write memory buffer.

During hash table creation, the keys in the hash table are partially sorted in the bucket-sorted hash join. Therefore, instead of a linear search inside a page block, the bucket-sorted hash join uses a binary search on the partially sorted page block. Consequently, it requires more time to generate a partially sorted hash table entries within a bucket, but probing time is reduced by using a binary search within a bucket.

By using the notations in Table 1, because partition phase is the same as that of the hybrid hash join, the total cost is defined as

$$\begin{aligned} T_{Partition} = &(N_r + N_s) * (T_h * CL * O(B_n)) \\ &+(N_r + N_s) * (1-q) * T_m \\ &+(\lceil NB_r/(NB_m - NBq_r - 1)\rceil \\ &+\lceil NB_s/(NB_m - NBq_s - 1)\rceil) \\ &*(T_r + (1-q) * T_w) \end{aligned}$$

During a hash table creation phase, blocks should be partially sorted, which requires $O(B_n^2)$ instead of $O(B_n)$ as in hybrid hash join. The cost to create a hash table is defined as

$$\begin{aligned} T_{Hash} = N_s * (T_h * CL * O(B_n^2) + (1-q) * T_m) \\ +(\lceil (NB_s - NBq_s)/(NB_m - 2)\rceil) * T_r \end{aligned}$$

Since the hash table is partially sorted, we can use a binary search within a bucket during the probing phase which requires $O(logB_n)$ rather than $O(B_n)$. Then, the probing cost is defined as

$$\begin{aligned} T_{Probe} = &(1-q) * N_r * (T_p * CL * O(log(B_n)) \\ &+(\lceil (NB_s - NBq_s)/(NB_m - 2)\rceil - 1) \\ &*(NB_r - NBq_r) * T_r \end{aligned}$$

The cost associated with writing the result blocks to the disk remains the same as

$$T_{Write} = NB_o * T_w$$

In this method, if relations R and S are already sorted, then we can replace the sorting terms $O(B_n^2)$ by $O(1)$ during the hash table creation. The probing phase would also then require $O(logB_n)$ with a binary search.

3) SENTIMENT-AWARE TASK USING BUCKET-SORTED HASH JOIN

Figure 5 depicts the sentiment–aware web crawling process based on bucket–sorted hash join. In the pre–processing step,

TABLE 1. Notations.

Notation	Description
R, S, O	Two relations to join R and S, output relation O.
N_r, N_s	Number of records in relation R and S. Assume $N_s \ll N_r$.
NB_r, NB_s, NB_o	Number of page blocks in relation R and S.
B_n	Number of entries in a bucket.
CL	Average chain length, $\lceil N_s/B_n \rceil$.
q	Utilization factor for a hash table, $q = N_{s_0}/N_s$, i,e, the ratio of the first page block.
NBq_s, NBq_r	Number of pages for the first partition block.
R_i,S_i, O_i	ith partition of relation R, S and O.
NB_m	Number of page blocks in main memory.
T_m	Time to move to a write buffer.
T_h	Time to create a hash table entry for a record, $T_h * CL * O(B_n)$ to generate unsorted bucket and $T_h * CL * O(B_n^2)$ to generate partially sorted bucket.
T_p	Probing time for a record, $T_p * CL * O(B_n)$ for sequential search bucket and $T_p * CL * O(logB_n)$ for binary search bucket.
T_r	Read one page block from disk to memory time.
T_w	Write one page block from memory to disk time.

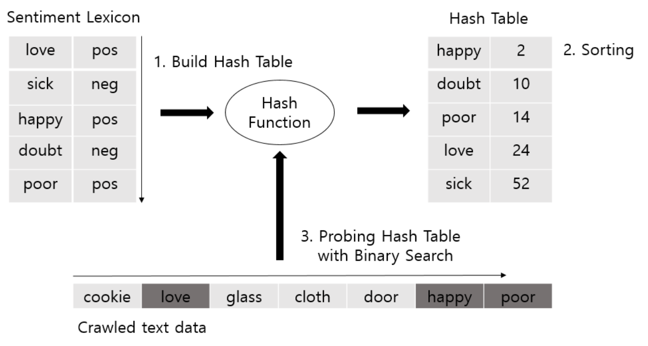


FIGURE 5. Sentiment–aware web crawling algorithm using bucket–sorted hash join.

a hash table is created with the sentiment words in the sentiment lexicon. When a hash table is created, a bucket address is assigned to each word through a hash function. The number next to the word in the hash table is the allocated bucket address. Then, the assigned bucket addresses are sorted in ascending order. During web crawling, each word w in crawled web page (a list of words in the web page [1]) is assigned to the bucket address using the hash function. To see whether w is in the hash table or not, the binary search is performed in the hash table. In this process, the words 'love', 'happy', and 'poor' are considered as sentiment words among 'cookie', 'love', 'glass', 'cloth', 'door', and 'happy'.

Algorithm 6 summarizes the pseudo code of the proposed sentiment–aware web crawling method using the bucket–sorted hash join algorithm. Each tuple of the hash table is key and value, where *key* is a word and *value* is the hash value that hashes the word. The bucket address is assigned to this hash value. A hash table is built by the sentiment words in the given sentiment lexicon and the tuples are sorted by the bucket addresses in the hash table. Meanwhile, each word w in the crawled web page is represented as the format of key and value. *key* is w and *value* is the hash value of hashing w. The hash value is assigned to a bucket address for w. To quickly find whether w is involved in the hash table or not, the binary search is executed by the bucket addresses. If the same bucket address is searched, the algorithm returns 'true' as output and 'false', otherwise. The sentiment score of a web page is computed by $\frac{\text{Number of true counts}}{\text{Number of true counts+Number of false counts}}$. If the score is greater than 0.1 ,[2] it is considered that the web page contains many possible sentiment words and is stored in the database. Once the hash table is created in the pre–processing step, only words in the crawled web pages are checked to see whether or not they contain in the hash table or not. Please note that we can ignore the running time of building the hash table because the size of the sentiment lexicon is small in general. The time complexity of probing the hash table is $O(kmlogn)$, where k, m, and $logn$ mean the number of web pages crawled, the average number of words per web page, and the total time of both computation of hash function and binary search.

V. EXPERIMENTAL VALIDATION

In this section, we explain the experimental set-up and show the performance evaluation.

A. EXPERIMENTAL SET-UP

Each method was in standalone executed in a high-performance workstation server with Intel Xeon 3.6GHz CPU with eight cores, 24GB RAM, 2TB HDD, and TITAN-X GPU with 3,072 CUDA cores, 12GB RAM, and 7Gbps memory clock. The operating system of the computer for the experiment was Ubuntu 18.04.3, and the proposed sentiment-aware web crawlers based on hash join and bucket-sorted hash join were implemented using Python 3.6.

To crawl web pages, web crawlers like Algorithm 1 started at three seed sites: (1) http://newdaily.com, (2) http://joongang.joins.com, and (3) http://hani.co.kr. The web crawlers went around neighboring web pages via hyperlink. These web pages were collected for one day on February 1, 2021, and all web pages with a sentiment score $\alpha = 0.1$ or

[1] Both non–text data and stop words are removed and each word is transformed to the root form through stemming

[2] The optimal value is found through several experiments.

less were filtered out because such web pages are not likely to have a number of sentiment words.

Through Beautiful Soup 4.9.1 [26], a Python library, we removed html or xml tags from each crawled web page and extracted text data within the web page. We also used MeCab 0.9.9 and its dictionary 2.1.1 [27] in Korean NLP in Python (Konlpy) 4.9.1 [28] for morpheme analysis of the parsed text data as above. The reason is: To match a word w in the text and the sentiment words registered in Kunsan National University (KNU) Korean Sentiment Dictionary [29], w is required to change to the root form of w.

Algorithm 6 Sentiment–Aware Web Crawling Algorithm Using Bucket–Sorted Hash Join

```
1  Input: seed_word, collected_word;
2  Sentiment_Score $\alpha$;
3  Build hash_table1;
4  for see_word in hash_table1 do
5      hash_table1 key = seed_word;
6      hash_table1 value = hash_function(seed_word);
7  Sorting hash_table1 value;
8  Insert bucket of hash_table1 value;
9  for seed_word in hash_table2 do
10     hash_table1 key = collected_word;
11     hash_table1 value = hash function(collected_word);
12 Probing with Binary Search;
13 $\alpha = \frac{\text{count(true)}}{\text{count(true) + count(false)}}$;
14 if $\alpha > 0.1$ then
15     Store the document;
```

In our previous work, we constructed the KNU Korean Sentiment Dictionary, which is composed of positive and negative words that represent the general expression of human emotions rather than positive and negative words used in specific domains such as foods, travels, movies, musics, cars, smart phones, lectures, computers, and so on. For example, common positive expressions include 'to be impressed,' 'to have value,' and 'thank you,' whereas general negative expressions include 'just that,' 'can't do it,' and 'to be pissed.' The purpose of the sentiment dictionary is to use it as a basis to automatically build the domain-dependent sentiment dictionary, which can be used for market research in a specific domain. The KNU sentiment dictionary has the following characteristics.

- The general sentiment words were collected by the glosses of each word constituting the standard Korean dictionary [30] [31] that contains about 500,000 words. First, we selected only glosses that include adjective, adverb, verb, or noun. Next, the Bi-LSTM model was trained with the selected glosses and then classified each gloss to one of positive, neutral, and negative classes. Finally, for each class, we selected top-k glosses and three evaluators manually extract sentiment words from the gloss and determined the polarity of each sentiment word through voting.
- The polarity of each sentiment word was classified into five levels: very positive, positive, neutral, negative, and very negative.
- This dictionary includes various types of positive and negative words such as 1-gram, 2-gram, n-gram (i.e., phrase and sentence pattern), and abbreviated words.

Table 2 shows the statistics of the sentiment expressions in KNU Korean Sentiment Dictionary. Using these sentiment expressions, Hash Tables H^D and H^S were constructed in the first and second phases of hash join and bucket-sorted hash join.

TABLE 2. Statistics of KNU korean sentiment dictionary.

Polarity	Number of sentiment expressions
Very Positive	2,597
Positive	2,266
Neutral	154
Negative	5,029
Very Negative	4,797
Total	14,843

To validate the effectiveness of the proposed methods, we experimented five methods. The first method is the traditional web crawling method in Algorithm 1 and is used as the baseline method in our experiment. This method does not work with the sentiment-aware task. The web crawler visits any web page and stores it to the database. Therefore, the database has all the crawling web pages. As the advantage of this method, it does not take any additional time to conduct the sentiment-aware task. In contrast, to construct the sentiment dictionary from the crawled web pages, it needs to scan the entire database. Thus, it takes a lot of time to build the sentiment dictionary. In addition, the database space is wasted because unnecessary web pages have to be stored. The second method is Algorithm 2 in which each word in the crawled web page is compared to all the sentiment words in KNU Sentiment Dictionary. This method needs pair-wise comparisons between the words in the crawled web page and the sentiment words in the sentiment dictionary. In the third method, the first phase of hash join is just completed and the pair-wise comparison between two blocks $B_i^D \in H^D$ and $B_i^W \in H^W$ in the second phase. In this method, hash function h_2 is not used. The Fourth method is the sentiment-aware task in Algorithm 3 based on hash join. In this method, we borrowed the idea of hash join from the database area and modify it to the sentiment-aware web crawling algorithm. The final method is the sentiment-aware task in Algorithm 3 based on bucket-sorted hash join. Unlike the fourth method, in this method, we propose a new bucket-sorted hash join and extend to the sentiment-aware web crawler. In next section, we denote the first method by *baseline*, the second method by *nest*, the third method by *hj1*, the fourth method by *hj2*, and the fifth method by *bshj*.

For the above five methods, the effectiveness of the proposed method is shown by comparing the time it takes to perform the sentiment-aware task, the number of matching words according to different sentiment scores αs, and the size of the storage space of the database.

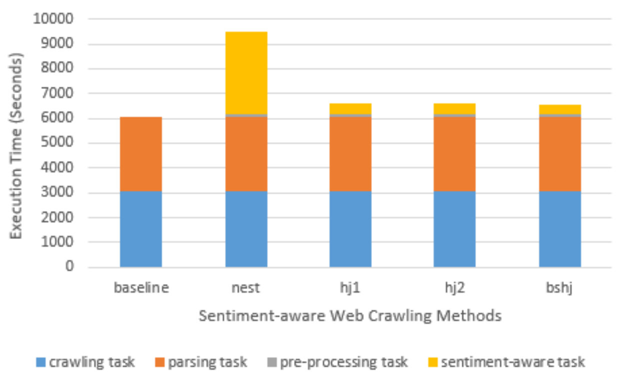
If the number of web pages to be crawled increases, the storage cost of the database will increase proportionally. All web pages that do not contain a lot of sentiment vocabulary are also stored in the database. In addition, it takes a lot of time because the entire database is scanned when a sentiment dictionary is constructed by determining new words or sentiment words dependent on a particular domain based on the stored web pages.

B. EXPERIMENTAL RESULTS

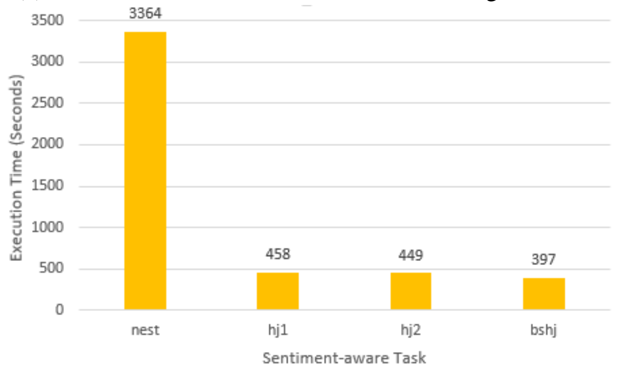
1) EXECUTION TIME OF SENTIMENT-AWARE WEB CRAWLING METHODS

In the previous section, we used the five method. The other methods except the *baseline* method perform the sentiment-aware task suggested in Algorithm 2. Specifically, it calculates how many words in a crawled web page match the sentiment words of the KNU sentiment dictionary. For example, suppose that the text content of the web page is a set of five words such as $W = \{t_1, t_2, t_3, t_4, t_5\}$. If t_2 and t_3 are determined as sentiment words after all words in W are compared with the sentiment words registered in the sentiment dictionary, the sentiment score of W is $\alpha = \frac{|\{t2,t3\}|}{|W|} = \frac{2}{5} = 0.4$. As α is higher, it is considered that W is likely to contain a number of sentiment words.

When the web crawler collected 30,000 web pages for the experiment, the *baseline* method did not perform the sentiment-aware task, and it took a total of 6,059 seconds to store all web pages in the database, as shown in Figure 6(a). This includes not only the time to crawl the web pages (*Crawling task*), but also the time to remove the html tags and extract only the text data (*Parsing task*). If the number of web pages to be crawled increases, the storage cost of the database will increase proportionally. All web pages that do not contain a lot of sentiment vocabulary are also stored in the database. In addition, it takes a lot of time because the entire database is scanned when a sentiment dictionary is constructed by determining new words or sentiment words dependent on a particular domain based on the stored web pages. In Figure 6(a), *Pre-processing task* and *Sentiment-aware task* correspond to morpheme analysis and Algorithm 2∼3, respectively. Interestingly, the execution times of the crawling, parsing, and pre-processing tasks were very close to all the methods. Moreover, most of the time was spent performing the crawling and parsing tasks among the four tasks. Relatively, the time it took to perform the pre-processing and sentiment-aware tasks was insignificant. For instance, in the *nest* method, the respective proportions of the four tasks to the total execution time were 32%, 32%, 1%, and 35%, respectively, while those were 47%, 46%, 1%, and 6%, respectively in the *bshj* method.



(a) Detailed Execution Times of Web Crawling Methods



(b) Execution Times of Four Sentiment-aware Methods

FIGURE 6. Execution times of five web crawling methods.

Figure 6(b) shows the time it takes when methods *nest*, *hj1*, *hj2*, and *bshj* perform the sentiment-aware task. *nest* takes 3,364 seconds to execute the sentiment-aware task for the 30,000 web pages, while *hj1*, *hj2*, and *bshj* take 458 seconds, 449 seconds, and 397 seconds, respectively. The average running time of the three methods is about 434 seconds and the hash-based methods are 6.8 times faster than *nest*. This is because *nest* performs the pair-wise comparisons between the words of the web page and the sentiment words of KNU sentiment dictionary. On the other hand, *hj1*, *hj2*, and *bshj* are efficient, compared to *nest* because hash function, hash join, and bucket-sorted hash join are used to quickly perform the sentiment-aware task. Since *hj1* uses one hash function (h_1) in the first phase of hash join and performs the pair-wise comparison on the results of h_1 in the second phase of hash join. This method is slightly slower than the methods based on hash join and bucket-sorted hash join. As shown in Figure 6(b), among the four sentiment-aware methods, *bshj*, the method using bucket-sorted hash join proposed in this paper is the most fastest. The reason is that each bucket in the hash table is sorted, and a fast search is performed through a binary search tree.

Please note that the running time of *baseline* (Algorithm 1) is 6,059 seconds but that of *bshj*, one of the proposed methods, is 6,548 seconds. Compared to the traditional web crawling method, when the proposed hash-based sentiment-aware task is performed for 30,000 web pages, the additional

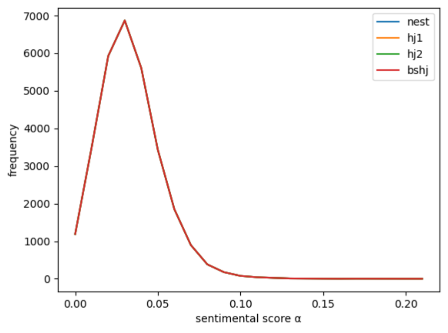


FIGURE 7. Frequency of web pages according to different α values.

time is at most 489 seconds, and the time to perform the sentiment-aware task per web page is 0.016 seconds. This means that adding the sentiment-aware task to the existing web crawler algorithm hardly affects the overall crawl time.

In the experiment, web crawlers started at three seed sites: (1) http://newdaily.com, (2) http://joongang.joins.com, and (3) http://hani.co.kr. Table 3 summarizes the execution time for the proposed sentiment-aware web crawling method using bucket-sorted hash join for each site. It seems that the execution times of each task for the three different sites are almost similar. They are in between 2,000 and 2,500 seconds.

2) EFFECTIVENESS OF THE PROPOSED METHODS

Unlike the existing web crawling method, the proposed methods have the advantages of saving the time to build the sentiment dictionary and the space of the entire database. To find out the need to perform the sentiment-aware task in web crawling, we measured the frequency of the crawled web pages for sentiment score α, which refers to the percentage of how many sentiment vocabulary each web page contains.

Figure 7 shows the frequency of web pages according to different α values. The x-axis and y-axis mean the α value and the frequency of web pages, respectively. For example, in the figure, the number of web pages with $\alpha = 0.1$ is up to 100. Most web pages have the sentiment score $\alpha = 0.04$. In addition, the four sentiment-aware methods, *nest*, *hj1*, *hj2*, and *bshj*, show the same results because data loss due to hash collision does not occur even if a hash function is used for quick retrieval.

TABLE 3. Execution times of *bshj* for each site (Unit: Seconds).

Task	Site (1)	Site (2)	Site (3)	Total
Crawling	988	1,011	1,073	3,072
Parsing	975	1,205	807	2,987
Pre-processing	27	34	31	92
bshj	125	132	140	397
Total	2,115	2,381	2,051	6,547

Most importantly, the sentiment score α of most web pages is in between 0 and 0.07, indicating that most web pages are not likely to have a lot of sentiment words. Consequently, there is no need to store such web pages in the database in advance to build the sentiment dictionary. From the result of Figure 7, we set a threshold value θ to 0.1. In the sentiment-aware task, if α of the crawled web page is less than θ, the web page is filtered, otherwise it is stored in the database.

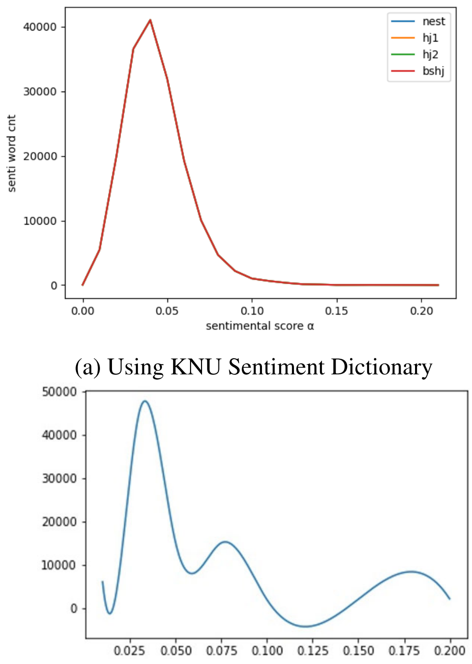


FIGURE 8. Frequency of sentiment words according to different α values.

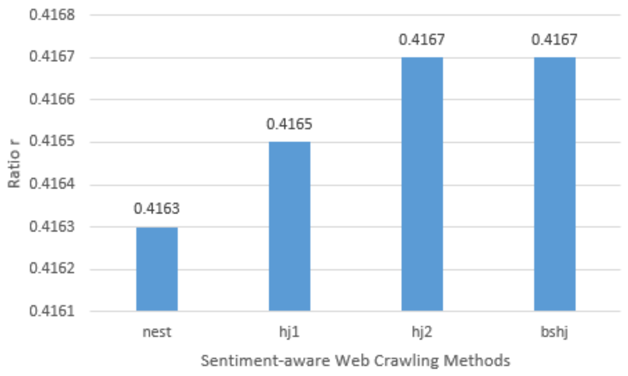
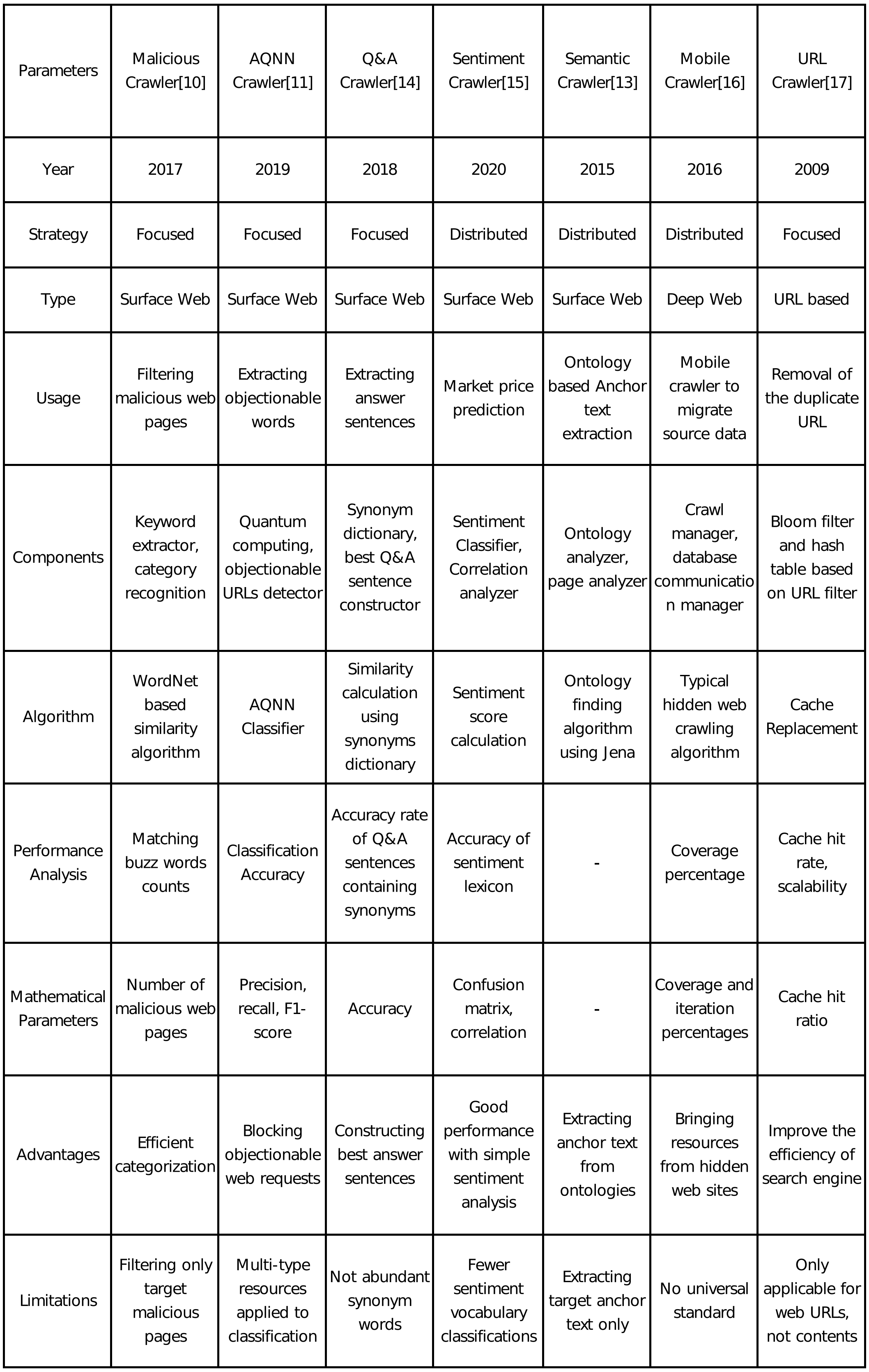


FIGURE 9. Space saving ratios of four hash-based sentiment-aware methods.

Figure 8(a) and (b) show the numbers of sentiment words in web pages according to different α values. Specifically, in Figure 8(a), the sentiment words in the crawled web pages are determined through KNU sentiment dictionary. On the other hand, in Figure 8(b), the sentiment words in the crawled web pages are identified by human evaluators.

Note that some word is definitely the sentiment vocabulary but is also missing in the sentiment dictionary. It seems that the frequency distributions of sentiment words according to different values of α are very different. The curve



Parameters	Malicious Crawler[10]	AQNN Crawler[11]	Q&A Crawler[14]	Sentiment Crawler[15]	Semantic Crawler[13]	Mobile Crawler[16]	URL Crawler[17]
Year	2017	2019	2018	2020	2015	2016	2009
Strategy	Focused	Focused	Focused	Distributed	Distributed	Distributed	Focused
Type	Surface Web	Surface Web	Surface Web	Surface Web	Surface Web	Deep Web	URL based
Usage	Filtering malicious web pages	Extracting objectionable words	Extracting answer sentences	Market price prediction	Ontology based Anchor text extraction	Mobile crawler to migrate source data	Removal of the duplicate URL
Components	Keyword extractor, category recognition	Quantum computing, objectionable URLs detector	Synonym dictionary, best Q&A sentence constructor	Sentiment Classifier, Correlation analyzer	Ontology analyzer, page analyzer	Crawl manager, database communicatio n manager	Bloom filter and hash table based on URL filter
Algorithm	WordNet based similarity algorithm	AQNN Classifier	Similarity calculation using synonyms dictionary	Sentiment score calculation	Ontology finding algorithm using Jena	Typical hidden web crawling algorithm	Cache Replacement
Performance Analysis	Matching buzz words counts	Classification Accuracy	Accuracy rate of Q&A sentences containing synonyms	Accuracy of sentiment lexicon	-	Coverage percentage	Cache hit rate, scalability
Mathematical Parameters	Number of malicious web pages	Precision, recall, F1-score	Accuracy	Confusion matrix, correlation	-	Coverage and iteration percentages	Cache hit ratio
Advantages	Efficient categorization	Blocking objectionable web requests	Constructing best answer sentences	Good performance with simple sentiment analysis	Extracting anchor text from ontologies	Bringing resources from hidden web sites	Improve the efficiency of search engine
Limitations	Filtering only target malicious pages	Multi-type resources applied to classification	Not abundant synonym words	Fewer sentiment vocabulary classifications	Extracting target anchor text only	No universal standard	Only applicable for web URLs, not contents

FIGURE 10. **Comparison of methodologies of the content-based web crawlers.**

of Figure 8(a) is very similar to that of Figure 7. On the other hand, the curve of Figure 8(b) is quite different from Figure 8(a). In particular, pay attention to the frequency of the sentiment words in case of $\alpha = 0.15$ or more. Unlike Figure 8(a), we can find many sentiment words that are missing in KNU Sentiment Dictionary. As a result, we can conclude that web pages with many sentiment vocabulary matched in KNU Sentiment Dictionary contain proportionally more unknown sentiment words that will be of great help when constructing an sentiment dictionary.

In addition, in Figure 8(a), the reason why the number of sentiment words is high at low α values is that the number of web pages is large. According to our deep investigation, the average sentiment words per web page is about 5. Even though the number of the sentiment words per web page is small, the frequency of sentiment words is high in small values of α because the number of web pages is large.

3) DATABASE STORAGE SPACE COMPARISON

In this paper, we experimented to see how much of the proposed methods save database space compared to the existing web crawling methods. Among the 30,000 web pages we crawled, four hash-based methods filtered out unnecessary web pages and calculated the web pages stored in the actual database by dividing them into 30,000 web pages. *nest*, *hj1*, *hj2*, and *bshj* store the web pages of 12,489, 12,489, 12,489, and 12,489, respectively. Thus, the storage rates of web pages in the database are $\frac{12{,}489}{30{,}000} = 0.4163$, $\frac{12{,}489}{30{,}000} = 0.4165$, $\frac{12{,}489}{30{,}000} = 0.4167$, and $\frac{12{,}489}{30{,}000} = 0.4167$. The average rate is about 0.41. The proposed hash-based methods save 1.44 times more database space than the existing web crawling method. Figure 9 shows the space saving ratios of *nest*, *hj1*, *hj2*, and *bshj*, compared to the existing web crawling method. In the figure, the x-axis is the four proposed hash-based schemes and the y-axis is the ratio $r = \frac{|\{W|\alpha \geq \theta\}|}{30{,}000}$. For this experiment, we set α to 0.1. Through this figure, we clearly understand that the proposed schemes can significantly save space in the database compared to the existing web crawling scheme.

VI. DISCUSSION

To construct the sentiment dictionary that contains new sentiment words or domain-dependent sentiment words, at first, the existing web crawling method collects web pages and stores them to the database. In the post-processing step, new sentiment vocabulary or domain-dependent sentiment expressions are extracted from the entire database. This approach is not efficient because it takes time to scan the entire database, and the database space is wasted because of unnecessary web pages.

To address this problem, we add the sentiment-aware task to the existing web crawling method. The goal of the sentiment-aware task is to match some words in the crawled web page with sentiment words in the sentiment dictionary. If unnecessary web pages can be removed by performing this sentiment-aware task during the crawling process, various problems occurring in the existing web crawling method can be solved. However, as the disadvantage of the proposed sentiment-aware task, it takes additional time to perform the sentiment-aware task in web crawling. We propose a hash-based sentiment-aware task to minimize the running time of the sentiment-aware task. Specifically, we propose two sentiment-aware web crawling methods using hash join and bucket-sorted hash join.

Our experimental results show that the time taken to execute the sentiment-aware task per web page is 0.016 seconds and the database space can be saved by 59% compared to the existing web crawling method. Furthermore, we observed that most web pages do not have a lot of sentiment words and the average number of sentiment words on a web page is 5 or so. When the sentiment score of a web page is more than 0.15, there are many unknown sentiment words that are not included in the sentiment dictionary. These experimental results indicate the necessity of the method proposed in this article in addition to the general web crawling method, and it is an effective method to construct a sentiment dictionary for a new word or domain-dependent sentiment vocabulary.

There are two limitations in the proposed hash join-based methods. First, by comparing α and θ values in Algorithm 6, it is decided whether to store the crawled web page to the database or filter it. This θ value should be adjusted across domains. In this work, we just found out the optimal value by setting θ differently and repeating the experiment. In our future research, it is necessary to study an automatic method to identify the optimal θ value across domains. Second, we focused merely on identifying whether a given web page contains a number of sentiment words or not. In practice, it is necessary to study how to extract new sentiment words that are not in the KNU sentiment lexicon after Algorithm 6 is performed. Since such a study is beyond the scope of this article, the in-depth study is needed in the future.

VII. CONCLUSION

In this work, we focus on the problem of using existing web crawling methods to build a sentiment dictionary. Through this method, it takes a lot of time to build the sentiment dictionary and unnecessary waste of database space. To solve these problems, for the first time, we propose a new sentiment-aware task in addition to the existing web crawling scheme. In the task, the words in the crawled web page are matched with the sentiment words in the sentiment dictionary, and the sentiment score of the web page is calculated. In this case, additional time is required due to the pair-wise word matching process between the web page and the sentiment dictionary. In this work, we propose two hash-based methods to minimize the time to perform this sentiment-aware task. One is based on hash join and the other is based on bucket-sorted hash join. In particular, we propose a new bucket-sorted hash join algorithm and apply it to the sentiment-aware task in web crawling.

Our experimental results shows that the proposed sentiment-aware web crawling methods outperform the existing web crawling method in the perspective of the execution time and the storage space. The time taken to execute the sentiment-aware task per web page is 0.016 seconds and the database space can be saved by 59% compared to the existing web crawling method.

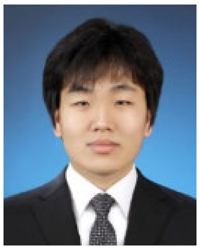
REFERENCES

[1] B. Liu, *Sentiment Analysis and Opinion Mining*. San Rafael, CA, USA: Morgan & Claypool, 2012.
[2] R. Wang, D. Zhou, M. Jiang, J. Si, and Y. Yang, "A survey on opinion mining: From stance to product aspect," *IEEE Access*, vol. 7, pp. 41101–41124, 2019.
[3] Q. Wang, L. Sun, and Z. Chen, "Sentiment analysis of reviews based on deep learning model," in *Proc. IEEE/ACIS 18th Int. Conf. Comput. Inf. Sci. (ICIS)*, Shanghai, China, Jun. 2019, pp. 258–261.
[4] S. Huang, Z. Niu, and C. Shi, "Automatic construction of domain-specific sentiment lexicon based on constrained label propagation," *Knowl.-Based Syst.*, vol. 56, no. 3, pp. 191–200, Jan. 2014.
[5] S. Cagnoni, P. Fornacciari, J. Kavaja, M. Mordonini, A. Poggi, A. Solimeo, and M. Tomaiuolo, "Automatic creation of a large and polished training set for sentiment analysis on Twitter," in *Proc. 3rd Int. Conf. Mach. Learn., Optim., Big Data*, vol.terra, Italy, 2017, pp. 146–157.
[6] W. Maharani, D. Widyantoro, and M. Khodra, "Aspect-based opinion summarization: A survey," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 2, pp. 448–456, 2018.
[7] Q. Lu, B. Liu, and H. Hu, "LCrawler: An enhanced measurement tool for peer-to-peer content sharing networks," in *Proc. 5th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Changchun, China, Dec. 2016, pp. 507–511.
[8] C. Na and B. On, "A proposal on a proactive crawling approach with analysis of state-of-the-art web crawling algorithms," *J. Internet Comput. Services*, vol. 20, no. 3, pp. 43–59, 2019.
[9] C. Saini and V. Arora, "Information retrieval in web crawling: A survey," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Jaipur, India, Sep. 2016, pp. 2635–2643.
[10] N. Agrawal and S. Johari, "A survey on content based crawling for deep and surface web," in *Proc. 5th Int. Conf. Image Inf. Process. (ICIIP)*, Shimla, India, Nov. 2019, pp. 491–496.
[11] S. Mali and B. B. Meshram, "Focused web crawler with revisit policy," in *Proc. Int. Conf. Workshop Emerg. Trends Technol.*, Mumbai, India, 2011, pp. 474–479.
[12] G. Agre and S. Dongre, "A keyword focused web crawler using domain engineering and ontology," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 3, pp. 463–465, 2015.
[13] A. Dilip Patel and V. N. Pandya, "Web page classification based on context to the content extraction of articles," in *Proc. 2nd Int. Conf. for Converg. Technol. (I2CT)*, Mumbai, India, Apr. 2017, pp. 539–541.
[14] O. Patel, N. Bharill, A. Tiwari, V. Patel, O. Gupta, J. Cao, J. Li, and M. Prasad, "Advanced quantum based neural network classifier and its application for objectionable web content filtering," *IEEE Access*, vol. 7, pp. 98069–98082, 2019.
[15] H. Dong and F. K. Hussain, "Self-adaptive semantic focused crawler for mining services information discovery," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1616–1626, May 2014.
[16] N. Kumar and M. Singh, "Framework for distributed semantic web crawler," in *Proc. Int. Conf. Comput. Intell. Commun. Netw. (CICN)*, Jabalpur, India, Dec. 2015, pp. 1403–1407.
[17] L. Gao and H. Chen, "An automatic extraction method based on synonym dictionary for web reptile question and answer," in *Proc. 13rd IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Wuhan, China, May 2018, pp. 375–378.
[18] J. Wang, H. Wang, and L. Wang, "Dependency parsing of financial news to improve sentiment analysis for predicting market prices," in *Proc. Int. Conf. Technol. Appl. Artif. Intell.*, Taipei, Taiwan, Dec. 2020, pp. 1–7.
[19] L. D. Shapiro, "Join processing in database systems with large main memories," *ACM Trans. Database Syst.*, vol. 11, no. 3, pp. 239–264, Aug. 1986.
[20] R. Lawrence, "Early hash join: A configurable algorithm for the efficient and early production of join results," in *Proc. 31st Int. Conf. Very Large Data Bases*, Trondheim, Norway, Aug. 2005, pp. 841–852.
[21] M. Kitsuregawa, M. Nakayama, and M. Takagi, "The effect of bucket size tuning in the dynamic hybrid grace hash join method," in *Proc. 15th Int. Conf. Very Large Data Bases*, Amsterdam, The Netherlands, Jul. 1989, pp. 257–266.
[22] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, "Application of hash to data base machine and its architecture," *New Gener. Comput.*, vol. 1, no. 1, pp. 63–74, Mar. 1983.
[23] J. M. Patel, M. J. Carey, and M. K. Vernon, "Accurate modeling of the hybrid hash join algorithm," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 22, no. 1, pp. 56–66, May 1994.
[24] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. A. Wood, "Implementation techniques for main memory database systems," *ACM SIGMOD Rec.*, vol. 14, no. 2, pp. 1–8, Jun. 1984.
[25] H. Shin, B. On, I. Lee, and G. Choi, "Bucket-sorted hash join," *J. Inf. Sci. Eng.*, vol. 36, no. 1, pp. 171–190, 2020.
[26] (Feb. 2021). *Beautiful Soup 4.9.1*. [Online]. Available: https://beautiful-soup-4.readthedocs.io/en/latest/
[27] (Feb. 2021). *Mecab 0.9.9*. [Online]. Available: https://bitbucket.org/eunjeon/mecab-ko-dic/src/master/
[28] (Feb. 2021). *Konlpy 4.9, 1*. [Online]. Available: https://konlpy.org/en/latest/
[29] (Feb. 2021). *Knu Korean Sentiment Dictionary*. [Online]. Available: https://github.com/park1200656/KnuSentiLex
[30] (Feb. 2021). *Standard Korean Dictionary*. [Online]. Available: https://stdict.korean.go.kr
[31] S. Park, C. Na, M. Choi, D. Lee, and B. On, "KNU Korean sentiment lexicon: Bi-LSTM-based method for building a Korean sentiment lexicon," *J. Intell. Inf. Syst.*, vol. 24, no. 4, pp. 219–240, 2018.

BYUNG-WON ON received the Ph.D. degree from the Department of Computer Science and Engineering, The Pennsylvania State University at University Park, PA, USA, in 2007. For several years, he worked as a full-time Researcher with The University of British Columbia and the Advanced Institution of Convergence Technology. Since 2014, he has been a Faculty Member with the Department of Software Convergence Engineering, Kunsan National University, South Korea. His recent research interests include data mining, especially probability theory and applications, and artificial intelligence, mainly working on abstractive summarization, creative computing, and multi-agent reinforcement learning.

JUN-YOUNG JO is currently pursuing the bachelor's degree with the Department of Software Convergence Engineering, Kunsan National University. His research interests include data mining, natural language processing, reinforcement learning, and deep learning.

HYUNKWANG SHIN is currently pursuing the Ph.D. degree in information and communication engineering with Yeungnam University. His research interests include text mining, sentiment analysis, social network analysis, machine learning, and databases systems.



JANGWON GIM received the Ph.D. degree in computer and radio communications engineering from Korea University, South Korea, in 2013. He worked as a Senior Researcher at the Korea Institute of Science and Technology Information. Since 2017, he has been a Faculty Member with the Department of Software Convergence Engineering, Kunsan National University, South Korea. His research interests include graph neural networks, knowledge embedding, natural language understanding, and semantic web for artificial intelligence.

GYU SANG CHOI (Member, IEEE) received the Ph.D. degree in computer science and engineering from The Pennsylvania State University. He was a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT), Samsung Electronics, from 2006 to 2009. Since 2009, he has been with Yeungnam University, where he is currently a Professor. His current research interests include text mining and reinforcement learning and deep learning with computer vision, whereas his prior research mainly focused on improving the performance of clusters. He is a member of ACM.

SOO-MOK JUNG received the Ph.D. degree from the Department of Computer Science and Engineering, Korea University, in 2002. For five years, he worked as a Researcher at the LG Information and Communication Research Center. Since 1998, he has been a Faculty Member with the Division of Computer Science and Engineering, Sahmyook University, South Korea. His research interests include image processing, computer systems, and data mining.