Knowledge-Based Systems 176 (2019) 147–158

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Incremental C-Rank: An effective and efficient ranking algorithm for dynamic Web environments

Jangwan Koo [a,1], Dong-Kyu Chae [a,1], Dong-Jin Kim [b], Sang-Wook Kim [a,*]

[a] Hanyang University, Seoul, Republic of Korea
[b] Brainsoft Inc., Seongnam, Republic of Korea

ARTICLE INFO

*Article history:*
Received 18 September 2018
Received in revised form 26 March 2019
Accepted 28 March 2019
Available online 1 April 2019

*Keywords:*
Information retrieval
Ranking algorithm
Dynamic ranking

ABSTRACT

Web page ranking is one of the core components of search engines. Given a user query, ranking aims to provide a ranked list of Web pages that the user is likely to prefer the most. By and large, the ranking algorithms can be categorized into content-based approaches, link-based approaches, and hybrid approaches. Hybrid ranking algorithms, which exploit both the content and link information, are the most popular and extensively studied techniques. Among the hybrid algorithms, *C-Rank* combines content and link information in a very effective way using the concept of *contribution*. This algorithm is known to provide high performance in terms of both accurate and prompt responses to user queries. However, C-Rank suffers from very high costs to reflect the highly dynamic and extremely frequent changes in the World Wide Web, because it re-computes all of the C-Rank scores used for ranking from scratch to reflect the changes. As a result, C-Rank may be considered inappropriate to provide users with accurate and up-to-date search results. This paper aims to remedy this limitation of C-Rank. We propose *incremental C-Rank*, which is designed to update the C-Rank scores of *only a carefully chosen portion of the Web pages* rather than those of all of the Web pages *without any accuracy loss*. Our experimental results on a real-world dataset confirm both the effectiveness and efficiency of our proposed method.



## 1. Introduction

With the burst of information currently available on the World Wide Web (WWW), various applications on the Web have been extensively studied where notable examples include Web page classification and ranking [1–5], social network analysis [6–9], and personalized item recommendation [10–12]. Among these applications, this paper focuses on *search engines* for Web pages (i.e., Web page ranking). Search engines consistently crawl, store, and index billions of Web pages [13]. Also, given a user query, they provide a list of a huge number of pages relevant to the search. Since not all of the pages in the list would satisfy the user's interest, it is important to place the pages that satisfy the user on the top of the list, which is the main goal of ranking algorithms employed in search engines [14].

Intuitively, as the retrieved pages (O1) are more relevant to the user query and (O2) are more authoritative on the Web, they are more likely to increase the user's satisfaction. *Content-based ranking algorithms* [15] such as *TF–IDF* [16], *BM25* [17,18], and *Language Model* [19] focus on objective O1 while *link-based ranking algorithms* such as *HITS* [20] and *PageRank* [21,22] focus on objective O2. However, rather than focusing on only one of the two objectives, pursuing them simultaneously using a so-called *hybrid ranking algorithm* has much more potential to satisfy the user's interest and thus has been widely studied [23–27]. Representative examples include a *hyperlink-based term propagation model* [24] and its variants [28–33], a *probabilistic relevance propagation model* [26], and a *contribution-based ranking algorithm* (*C-Rank*, in short) [34]. Among them, *C-Rank* performs the best in terms of both accuracy (i.e., effectiveness) and response time (i.e., efficiency) to the user query.

C-Rank combines content and link information using the concept of *contribution*, implying that a page may contribute to improving the content quality of neighboring pages that refer to it through hyperlinks [34]. A *C-Rank score* of each page on a term is computed as a linear combination of (1) its *relevance score* to the term and (2) its *contribution score* which quantifies the degree of its overall contributions to other pages on the term. The detailed descriptions of C-Rank are provided in Section 2. This notion of contribution has been successfully utilized not only in ranking Web pages [34] but also in estimating the similarities among scientific papers [35–37].

* Corresponding author.
*E-mail addresses:* koojwan@hanyang.ac.kr (J. Koo), kyu899@hanyang.ac.kr (D.-K. Chae), djkim@brainsoft.ai (D.-J. Kim), wook@hanyang.ac.kr (S.-W. Kim).

[1] Co-first authors having contributed equally to this work.

https://doi.org/10.1016/j.knosys.2019.03.034
0950-7051/

A C-Rank score of every term in every page is pre-computed in the offline phase. In the online phase, given a user query, the final C-Rank score of each page is computed by aggregating its pre-computed C-Rank scores of the terms included in the query. This ranking mechanism makes C-Rank efficiently respond to the query. However, the offline computation of all of the C-Rank scores requires a long time. These days, the status of Web pages is continuously changing as new pages may be generated, some existing pages may be deleted, and the contents of some existing pages may be partially updated. We refer to these three types of status changes as *Web changes*, hereafter. Accordingly, in addition to an efficient response to the query (i.e., in the online phase), the efficient reflection of the *Web changes* in search engines is also becoming important (i.e., in the offline phase). A simple and normal way to reflect these *Web changes* is to re-compute the C-Rank score of every term in every page from scratch. However, this process is extremely time consuming. This would cause C-Rank to reflect Web changes infrequently, thereby making C-Rank suffer from the problem of providing outdated and thus, inaccurate search results to users.

Motivated by this limitation of C-Rank (referred to as *static C-Rank*, hereafter), in this paper, we propose *incremental C-Rank*, which enables the *Web changes* to be reflected *incrementally* rather than re-computing the C-Rank scores of all of the pages from scratch, thereby efficiently maintaining the C-Rank scores of all of the pages up-to-date. When reflecting the *Web changes*, our method *iteratively* finds the pages whose C-Rank scores could be affected by the *Web changes*, and then selectively updates their C-Rank scores. More specifically, in each iteration, our method carefully identifies (A) the pages which need to update their C-Rank scores and (B) the pages which do not need to update their C-Rank scores but need to be involved in the process of updating the C-Rank scores of other pages. Then, it updates the C-Rank scores of the pages of type (A) by making the pages of type (B) propagate the *delta-contribution score* to (A), which is referred to as *delta-contribution score propagation (d-CSP)*. Here, the *delta-contribution score* represents the difference of the degree of a page's overall contributions to the other pages before and after the *Web changes* occur. A detailed description of *d-CSP* is provided in Section 4.

Since the number of pages involved in our incremental update is much smaller than that in the entire Web graph, our incremental C-Rank can perform much more efficiently in reflecting Web changes than the existing static C-Rank. Moreover, the proposed *d-CSP* makes our method provide the exact same results (i.e., a C-Rank score of every term in every page) as those provided by the original static C-Rank, which will be proven mathematically in Section 4.4. We verified the effectiveness and efficiency of our proposed method through extensive experiments. The experimental results reveal that our incremental C-Rank reflects *Web changes* up to 252 times faster than static C-Rank. We also show that our incremental C-Rank and static C-Rank resulted in identical C-Rank scores. These results confirm that our proposed method is suitable for real-world Web environments where *Web changes* occur frequently and rapidly since it can reflect such changes efficiently while not causing errors.

Moreover, our incremental C-Rank can be applied to a variety of portals or platforms as long as the concept of *contribution* is beneficial. For example, C-Rank can be employed in *scientific literature* search engines [38–41]. Here, each research paper would correspond to a web page, and each citation in the paper would correspond to a hyperlink in a web page similar to a paper that contributes to another paper via a citation [35–37]. Another example is search engines for *user contents on SNS* [6,8,9,42]. In this example, the contents uploaded by each user (e.g., Facebook posts or tweets) would be linked to other users or other posts through tags and links, which would result in contributing to each other's contents on their SNS. Therefore, these applications can provide more efficient and more accurate search results to users by employing our incremental C-Rank.

In summary, the main contributions of this paper are as follows:

1. We develop an *incremental C-Rank algorithm*, which remedies the efficiency problem in the static C-Rank algorithm and thus provides users with accurate and up-to-date search results.
2. We propose *d-CSP (delta-contribution score propagation)*, which is a core part of the proposed incremental C-Rank and makes the incremental updates more efficient than the naive method.
3. We mathematically prove that our incremental C-Rank always provides the exact same results as those of the static C-Rank.
4. We conduct extensive experiments on two real-world datasets and demonstrate the enhanced efficiency of our incremental C-Rank.

The rest of this paper is organized as follows. The static C-Rank is introduced in Section 2. The limitation of static C-Rank and a naive solution are provided in Section 3. The details of the proposed solution, incremental C-Rank, and *d-CSP* are explained in Section 4. The summary of the analysis of our experimental results is provided in Section 5. Finally, the results are summarized and conclusions are provided in Section 6.

## 2. Static C-Rank

C-Rank is a contribution-based ranking algorithm which exploits both the content of pages and the overall link structure on the Web [34]. This algorithm pre-computes a C-Rank score of every term in every page in the offline phase. For responding to a user query, it aggregates for every page its pre-computed C-Rank scores of those terms included in the query and finally ranks the pages according to the scores.

Formally, a C-Rank score of term $t$ in page $q$, denoted as $CR_t(q)$, is defined by the weighted sum of its *relevance score* and *contribution score* as follows:

$$CR_t(q) = \lambda R_t(q) + (1-\lambda)C_t(q), \qquad 0 \le \lambda \le 1 \tag{1}$$

where $R_t(q)$ is a relevance score of page $q$ to term $t$, and $C_t(q)$ is a contribution score of page $q$ to other pages on term $t$. The relevance score indicates how important and relevant a term is to a page. We can simply compute it by borrowing existing term weighting schemes such as TF–IDF [16] and BM25 [17]. The contribution score $C_t(q)$ implies how much page $q$ contributes to improving the quality of other pages in terms of term $t$, directly or indirectly, pointing to it through the path composed of one or more links. $C_t(q)$ is defined as follows:

$$C_t(q) = \begin{cases} \sum_{i=1}^{d} \sum_{p \in D(q,i)} \sum_{(q,p) \in p \xrightarrow{i} q} C_t(q,p) & \text{if } t \in key(q) \\ 0 & \text{if } t \notin key(q) \end{cases} \tag{2}$$

where $key(q)$ in Eq. (2) denotes a set of keywords in $q$. Since a page generally deals with one or a few topics, it can be summarized by a small number of keywords corresponding to the topics [34]. Here, the keywords in a page are defined as the top-$N$ terms according to their relevance scores to the page. $D(q,i)$ indicates a set of pages, each of which is a starting page of a path of length $i$ that ends at page $q$. $p \xrightarrow{i} q$ denotes a set of all possible paths of length $i$ from $p$ to $q$. $d$ indicates the *cutoff path length*,

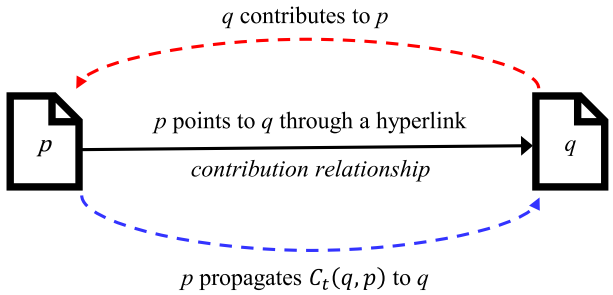


**Fig. 1.** $p$ points to $q$ through a hyperlink, indicating that $q$ could contribute to improving the content quality of $p$. In this relationship, $p$ sends a contribution score to $q$.

which restricts the maximum length of paths through which a page is allowed to contribute to others. $C_t(q,p)$ is the contribution score on common keyword $t$ (i.e., $t \in key(p) \cap key(q)$) from $q$ to $p$, computed as follows (see Fig. 1):

$$C_t(q,p) = \begin{cases} \alpha_t^i(q,p) R_t(p) & \text{if } t \in key(p) \cap key(q) \\ 0 & \text{if } t \notin key(p) \cap key(q) \end{cases} \quad (3)$$

$\alpha_t^i(q,p)$ is the contribution ratio of page $q$ to page $p$ on $t$. Here, $q$ can contribute directly ($i = 1$) or indirectly ($i \geq 2$) to $p$. For the direct contribution, $\alpha_t^1(q,p)$ is defined as follows:

$$\alpha_t^1(q,p) = \frac{R_t(q)}{R_t(p) + \sum_{r \in outlink(p)} R_t(r)} \quad (4)$$

Intuitively, the contribution of $q$ to $p$ on $t$ can be greater in the following cases: (a) the more relevant $p$ is to $t$, (b) the more relevant $q$ is to $t$, (c) the less relevant $p$ is to $t$ than $q$ is, and (d) the less relevant pages pointed by $p$ is to $t$. Eqs. (3) and (4) reflect the above intuitions.

For an indirect contribution, where $q$ points to $p$ through a path of $p \rightarrow r_1 \rightarrow \cdots \rightarrow r_{i-1} \rightarrow q$, $\alpha_t^i(q,p)$ is calculated as the product of the contribution ratio of each link included in the path, as follows:

$$\alpha_t^i(q,p) = \alpha_t^1(q, r_{i-1}) \prod_{j=1}^{i-2} \alpha_t^1(r_{j+1}, r_j) \times \alpha_t^1(r_1, p) \quad (5)$$

A C-Rank score of every term in every page can be effectively computed by the *contribution score propagation*, as described in Fig. 2 and Algorithm 1. First, Fig. 2 briefly shows the contribution score propagation starting from page $p$. On the 1st iteration, $p$ directly propagates the relevance score multiplied by the contribution ratio to the adjacent page, $q$. On the 2nd iteration, $q$ re-propagates the previously received contribution score to page $r$ by multiplying $q$'s own contribution ratio to $r$, i.e., $\alpha_t^1(r,q)$. Lastly, $r$ propagates the received contribution score with multiplication of its own contribution ratio to $s$ (see Table 1).

Algorithm 1 describes the full process of static C-Rank including the contribution score propagation and some initializations that should be prepared.

Lines 1 to 9 describe the initializations, including computing relevance scores $R_t(q)$ of every term in every page, identifying $key(q)$ of every page, and computing the direct contribution ratio $\alpha_t^1(q,p)$ among every pair of $p$ and $q$ by utilizing Eq. (4).

Lines 10 to 24 show the contribution score propagation. This process has $d$ iterations in total (see line 10): in each $i$th iteration, every page $q$ (1) receives the contribution scores cumulated in the previous iteration from its adjacent pages $p$ (i.e., pointing to $q$), (2) multiplies them with the corresponding contribution ratio $\alpha_t^1(q,p)$, and (3) cumulates them on its own temporary variable $C_t^i(q)$ (see line 20). Exceptionally, in the 1st iteration, $q$ cumulates $\alpha_t^1(q,p)R_t(p)$ since none of the pages receive any contribution scores cumulated in the previous iteration from its adjacent pages (see line 18).

Eventually, for each page $q$, their contribution score on term $t$ to other pages is the aggregation of $C_t^i(q)$ from $i = 1$ to $d$. Then, each page $q$'s final C-Rank score is simply computed as a linear

**Table 1**
Notations.

| Notation | Description |
|---|---|
| $CR_t(q)$ | A C-Rank score of term $t$ in page $q$ |
| $R_t(q)$ | A relevance score of page $q$ to term $t$ |
| $C_t(q)$ | A contribution score of page $q$ to other pages on term $t$ |
| $C_t(q,p)$ | A contribution score on common keyword $t$ from page $q$ to page $p$ |
| $\alpha_t^i(q,p)$ | A contribution ratio of page $q$ to page $p$ on term $t$ |
| $d$ | The cutoff path length |
| $N$ | The number of keywords |
| $\lambda$ | The weight |
| $key(q)$ | A set of keywords in $q$ |
| $D(q,i)$ | A set of pages, each of which is a starting page of a path of length $i$ that ends at page $q$ |
| $p \xrightarrow{i} q$ | A set of all possible paths of length $i$ from $p$ to $q$ |
| $T$ | A current time point |
| $T_w$ | A time point when the latest update process has been completed |
| $CR_{t,T}(q)$ | A C-Rank score of term $t$ in page $q$ at time point $T$ |
| $R_{t,T}(q)$ | A relevance score of page $q$ to term $t$ at time point $T$ |
| $C_{t,T}(q)$ | A contribution score of page $q$ to other pages on term $t$ at time point $T$ |
| $C_{t,T}(q,p)$ | A contribution score on common keyword $t$ from page $q$ to page $p$ until time point $T$ |
| $\alpha_{t,T}^i(q,p)$ | A contribution ratio of page $q$ to page $p$ on term $t$ at time point $T$ |
| $\Delta AC_{t,T}(q,p)$ | A delta-contribution score of page $q$ on page $p$ with respect to term $t$ at time point $T$ |
| $PC_{t,T}(q,p)$ | The sum of all the contribution scores that page $p$ propagates to page $q$ *directly* and *indirectly* until time point $T$ |
| $C_{t,T}^i(p)$ | The sum of the contribution scores that page $p$ has received from other pages pointing to it indirectly through the paths of length $i$ until time point $T$ |
| $P_{all}$ | An entire set of pages in a Web graph dataset |
| $P_{old}$ | A set of all the existing pages before any *Web changes* occur |
| $P_{new}$ | A set of all pages where the *Web changes* occurred after the latest update process |
| $P_{chunk}$ | A set of pages where the *Web changes* would be reflected in the current update process, and which is a part of $P_{new}$ |
| $P_{target}$ | A set of pages where the re-computation of C-Rank scores is required |
| $P_{engaged}$ | A set of pages that need to be involved in updating C-Rank scores of $P_{target}$ |

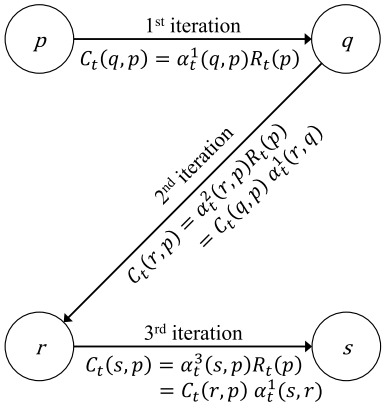


**Fig. 2.** A situation where every page contains term $t$ and the cutoff path length $d$ is 3.

combination of the contribution score and the relevance score (see line 24 and Eq. (1)).

## 3. Motivation

Let $P_{old}$ be the set of all the existing pages before any *Web changes* occur and $P_{chunk}$ be a set of pages where *Web changes* have occurred (i.e., newly added pages, deleted pages, and some existing pages whose contents have been modified). Note that a C-Rank score of every term in every page in $P_{old}$ is already computed and the emerging of $P_{chunk}$ may affect the C-Rank score of some terms in some pages in $P_{old}$. The problem we are going to solve is to update all of the C-Rank scores of $P_{old} \cup P_{chunk}$ correctly and efficiently.

To solve this problem, static C-Rank needs to re-compute all of the C-Rank scores from scratch, which takes a tremendous amount of time. Suppose a simple Web graph $m \rightarrow n \rightarrow o \rightarrow p \rightarrow q \rightarrow r \rightarrow s$ where all of the pages have the keyword "laptop". If page $p$ is modified in terms of the keyword "laptop" (e.g., more "laptop"s added), some C-Rank scores in the Web graph need to be updated. Although only one page has been changed with a few pages affected, static C-Rank has no way of selectively updating the C-Rank scores of those pages but it needs to re-compute the contribution scores of every page by conducting the contribution score propagation from scratch (see Algorithm 1). This would result in it being infeasible for static C-Rank to reflect *Web changes* frequently, thereby making it difficult to provide accurate and up-to-date search results.

Motivated by this limitation of static C-Rank, this study aimed to develop an incremental update method that can selectively update C-Rank scores of only those pages related to *Web changes* (i.e., emerging of $P_{chunk}$), rather than re-computing all of the C-Rank scores from scratch. The key component of the selective update is to carefully determine the pages whose C-Rank scores are affected by *Web changes* and then to update only their C-Rank scores accordingly. Hereafter, we denote those pages as $P_{target}$, indicating the *targets* that require re-computation of the C-Rank scores.

**Algorithm 1** Static C-Rank

**Input:** A set of pages $P_all$ and their hyperlinks, the number of keywords $N$, the cutoff path length $d$, and the weight $\lambda$
**Output:** A C-Rank score of every term in every page

```
    Initialization
1:  foreach q ∈ P_all do
2:      foreach t ∈ term(q) do
3:          Compute R_t(q)
4:          C_t(q) = 0
5:      Extract top-N terms in page q as key(q)
6:  foreach p ∈ P_all do
7:      foreach q ∈ outlink(p) do
8:          foreach t ∈ key(p) ∩ key(q) do
9:              Compute α_t^1(q, p) with Eq. (4)
    Contribution score propagation
10: for i = 1 to d do
11:     foreach q ∈ P_all do
12:         foreach t ∈ term(q) do
13:             C_t^i(q) = 0 // initializing temporary variables
14:     foreach p ∈ P_all do
15:         foreach q ∈ outlink(p) do
16:             foreach t ∈ key(p) ∩ key(q) do
17:                 if (i == 1) then
18:                     C_t^i(q) += α_t^1(q, p) R_t(p)
19:                 else
20:                     C_t^i(q) += α_t^1(q, p) C_t^{i-1}(q)
21:     foreach q ∈ P_all do
22:         foreach t ∈ term(q) do
23:             C_t(q) += C_t^i(q)
24:         CR_t(q) = λ R_t(q) + (1 − λ) C_t(q)
    return CR(P_all)
```

### 3.1. Naive solution

As for the selective update, we can consider a naive solution that can be realized by slightly modifying static C-Rank, without much extension. Its specific steps are the following.

1. Identify $P_{target}$ and $P_{engaged}$.
   - In the case of this naive solution, $P_{target}$ corresponds to those pages whose relevance scores and/or contribution scores are changed due to the *Web changes*, according to Eqs. (3) and (4). $P_{engaged}$ corresponds to those pages reachable to any of the pages in $P_{target}$ within $d$ hops.
2. Build a subgraph consisting of $P_{target}$ and $P_{engaged}$.
3. Save the C-Rank scores of the pages in $P_{engaged}$ on temporary variables to avoid the zero-initialization of the static C-Rank algorithm.
4. Apply the static C-Rank algorithm to the subgraph and update all of the pages in the subgraph.
5. Restore the C-Rank scores of $P_{engaged}$.

With this naive solution, we can update the C-Rank scores of $P_{target}$. In the above simple Web graph example, if page $p$ is modified in terms of the keyword "laptop" and $d = 2$, $P_{target}$ corresponds to a set of pages $p$, $q$, and $r$ since $p$'s relevance score changes and $q$ and $r$'s contribution scores change as well due to the change of $p$'s relevance score (see Eqs. (3) and (4)). $P_{engaged}$ corresponds to a set of pages $n$ and $o$ since they can reach $p$, $q$, and $r$, and thereby capable of passing the contribution scores to them. By building a subgraph $n \rightarrow o \rightarrow p \rightarrow q \rightarrow r$ and running static C-Rank on it according to the above procedure, we can perform a selective update on the C-Rank scores of $p$, $q$, and $r$. We refer to this solution as a *naive method* hereafter.[2]

## 4. Incremental C-Rank

Our proposed solution, named *incremental C-Rank*, is quite different in terms of its philosophy from that of the naive method. We observed that the computation time of a page's C-Rank score is *closely related to the number of computations and propagations for contribution scores*. Specifically, to compute page $q$'s contribution score, the contribution score of any page that can reach $q$ within $d$ hops needs to be computed and then propagated to $q$ through at most $d$ hops. In this sense, our key objective is to reduce the number of computations and propagations for the contribution scores.

The proposed incremental C-Rank consists of a series of iterations where in each iteration, it identifies its own $P_{target}$ and $P_{engaged}$ first, and updates $P_{target}$ by making $P_{engaged}$ propagate the *delta-contribution score* to $P_{target}$, which we refer to as *delta-contribution score propagation (d-CSP)*. This iterative update process continues until no pages exist whose C-Rank score changes.

[2] This idea has been presented as a 5 page paper including some preliminary experimental results in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, entitled "Incremental Maintenance of C-Rank Scores in Dynamic Web Environment" [43]. This paper is an extended version of this idea and contains fairly new ideas and much more extensive experimental results.

In each iteration, our method selects a significantly reduced number of pages as $P_{engaged}$ compared to the naive method as well as static C-Rank, thereby minimizing the number of computations for the contribution scores. Also, our proposed *d-CSP* allows only *direct* (i.e., through one hop) propagations, which also makes the overall C-Rank update process much more efficient. Moreover, our incremental C-Rank also ensures the exact same results as those obtained by re-computing all of the C-Rank scores of $P_{old} \cup P_{chunk}$ by employing static C-Rank from scratch.

The following subsections provide more detailed explanations. In Section 4.1, we provide some preliminaries to introduce the concept of the *time point*. Then, in Section 4.2, we explain the methodology employed to determine $P_{target}$ and $P_{engaged}$ efficiently in each iteration, which is quite different from that used in the naive method. In Section 4.3, we describe the proposed *d-CSP*. We provide the detailed algorithm of our incremental C-Rank equipped with *d-CSP* in Section 4.4.

### 4.1. Preliminaries

Since we are dealing with a dynamic update of C-Rank scores, the concept of the *time point* should be additionally introduced. This section provides some notations and equations which the concept of the time point involves.

First, we assume that, during an update process, additional *Web changes* could occur. However, we cumulate them in the next $P_{chunk}$ and deal with them after the current update process is completed. Therefore, we ensure that all of the C-Rank scores are up to date when an update process starts.

Let $T$ denote a current time point and $T_w$ be an initial time point when the latest update process has been completed. We regard the scale of the time between two adjacent time points (e.g., $T_w$ and $T_w + 1$) as *one iteration* of an updating process. For example, if the first iteration of updating $P_{target}$ starts at $T_w$, then it terminates at $T_w + 1$. Accordingly, $T = T_w + i$ corresponds to the end of the $i$th iteration.

Next, we re-write all of the equations of static C-Rank by adding the notion of the time point to them.

$CR_{t,T}(q)$ denotes a C-Rank score of term $t$ in page $q$ at time point $T$, which is computed as follows:

$$CR_{t,T}(q) = \lambda R_{t,T}(q) + (1-\lambda) C_{t,T}(q), \qquad 0 \le \lambda \le 1 \tag{6}$$

where $R_{t,T}(q)$ is the relevance score of $q$ to $t$ at $T$ and $C_{t,T}(q)$ is a contribution score of $q$ at $T$. $C_{t,T}(q)$ is computed as follows:

$$C_{t,T}(q) = \begin{cases} \sum_{i=1}^{d} \sum_{p \in D(q,i)} \sum_{(q,p) \in p \xrightarrow{i} q} C_{t,T}(q,p) & \text{if } t \in key(q) \\ 0 & \text{if } t \notin key(q) \end{cases} \tag{7}$$

where $C_{t,T}(q,p)$ is the sum of contribution scores on $t$ which $p$ has propagated to $q$ until $T$. $C_{t,T}(q,p)$ can be computed as shown below (for simplicity, we suppose $t \in key(p) \cap key(q)$).

$$C_{t,T}(q,p) = \begin{cases} \alpha^{i}_{t,T}(q,p) R_{t,T-i+1}(p) & \text{if } T \ge T_w + i \\ 0 & \text{if } T < T_w + i \end{cases} \tag{8}$$

If the current time point $T$ is earlier than $T_w + i$ (note that $i$ denotes the distance between $p$ and $q$), $p$'s contribution score has not reached $q$ yet in the current update process and as a result, $C_{t,T}(q,p)$ is zero. $\alpha^{i}_{t,T}(q,p)$ denotes the contribution ratio of $q$ to $p$ on $t$ at $T$, consisting of direct contribution ($i = 1$) or indirect contribution ($i \ge 2$), each of which is computed as follows:

$$\alpha^{1}_{t,T}(q,p) = \frac{R_{t,T}(q)}{R_{t,T}(p) + \sum_{r \in outlink(p)} R_{t,T}(r)} \tag{9}$$

$$\alpha^{i}_{t,T}(q,p) = \alpha^{1}_{t,T}(q, r_{i-1}) \prod_{j=1}^{i-2} \alpha^{1}_{t,T-i+j+1}(r_{j+1}, r_j) \times \alpha^{1}_{t,T-i+1}(r_1, p) \tag{10}$$

### 4.2. Finding $P_{target}$ and $P_{engaged}$

This section addresses how to find $P_{target}$ and $P_{engaged}$ at every time point. At the initial time point, i.e., at time point $T_w$ $P_{target}$ and $P_{engaged}$ are determined via the following steps.

- **Step A: Finding the pages whose relevance scores are changed.** According to Eq. (6), a page's C-Rank score is a weighted sum of its relevance score and contribution score. Therefore, if a page's relevance score to a term changes, its C-Rank score of the term should change as well. We denote a set of these pages as $P_{step_A}$.
- **Step B: Finding the pages whose contribution scores are changed.** Once we identify $P_{step_A}$, we can also identify the pages whose contribution score changes according to Eqs. (8) and (9): **(B-1)** the pages pointed by $P_{step_A}$ and **(B-2)** the pages pointed by those pages which point to $P_{step_A}$. We denote the two kinds of pages as $P_{step_{B-1}}$ and $P_{step_{B-2}}$, respectively. We also denote $P_{step_{B-1}} \cup P_{step_{B-2}}$ as $P_{step_B}$. Finally, we obtain $P_{target}$ by $P_{step_A} \cup P_{step_B}$.
- **Step C: Finding** $P_{engaged}$. $P_{engaged}$ plays a role in computing and propagating the *delta-contribution scores* to $P_{target}$, which we will describe in the next section. Therefore, those pages directly pointing to $P_{target}$ are the members of $P_{engaged}$.

Once $P_{target}$ and $P_{engaged}$ are identified at $T_w$, $P_{engaged}$ propagates the scores to $P_{target}$ in the first iteration (i.e., from $T_w$ to $T_w + 1$). Then, in the following $i$th iteration (i.e., from $T_w + i - 1$ to $T_w + i$, where $i > 1$), $P_{target}$ of the previous $(i-1)$th iteration becomes $P_{engaged}$ so as to re-propagate the previously received *delta-contribution scores* to the pages pointed by them in the current iteration. Therefore, the new members of $P_{target}$ in the current iteration will be those pointed by $P_{engaged}$.

Note that, in any iteration, we allow a page to belong to both $P_{target}$ and $P_{engaged}$. This page propagates the *delta-contribution scores* to the pages pointed by it via their outlinks while receiving the *delta-contribution scores* from the pages pointing to it via their inlinks and then updating its own C-Rank score.

Fig. 3 shows a toy example where all the pages contain "laptop" as a keyword and the cutoff path length $d$ is assumed to be 2. Here, page $p$ is modified in terms of the term "laptop". In the first iteration to reflect this change (i.e., at $T_w + 1$), page $p$ is a member of $P_{step_A}$, page $q_0$ pointed by page $p$ is a member of $P_{step_{B-1}}$, and pages $m_0$ and $p$, which are pointed by pages $l_0$ and $o_0$ pointing to page $p$, are members of $P_{step_{B-2}}$. As a result, pages $m_0$, $p$, and $q_0$ are members of $P_{target}$ and pages $p$, $l_0$, $q_1$, $m_1$, and $o_0$ are members of $P_{engaged}$.

After identifying $P_{target}$ and $P_{engaged}$ in this way, $P_{engaged}$ propagates the *delta-contribution scores* to $P_{target}$. In the next iteration, pages $m_0$, $p$, and $q_0$ change their roles from $P_{target}$ to $P_{engaged}$ and pages $n_0$, $q_0$, and $r_0$ become the new $P_{target}$. Then, the process is repeated in the same way as in the first iteration.

### 4.3. $d-CSP$

We now present (in an arbitrary iteration) how our *delta-contribution score propagation (d-CSP)* updates the C-Rank scores of each page in $P_{target}$, i.e., computing $C_{t,T}(q)$ where $q \in P_{target}$. Since $CR_{t,T}(q)$ is a weighted sum of $R_{t,T}(q)$ and $C_{t,T}(q)$, $R_{t,T}(q)$ and $C_{t,T}(q)$ need to be re-computed. While the re-computation

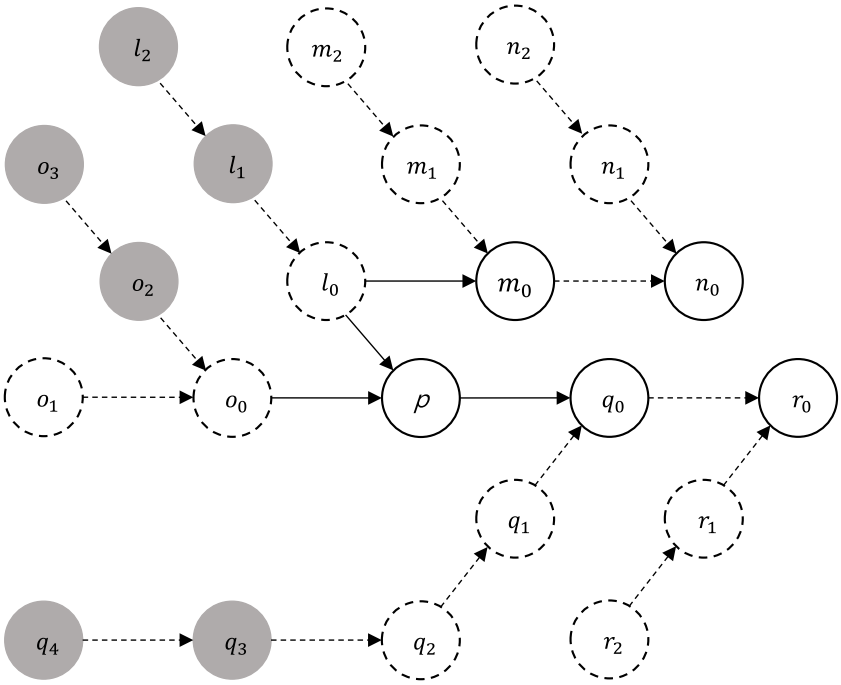


**Fig. 3.** A toy example where the pages with a solid line correspond to $P_{target}$ or $P_{engaged}$. The shaded pages are not included in $P_{target}$ nor $P_{engaged}$.

of $R_{t,T}(q)$ is straightforward (e.g., simply re-computing TF–IDF of term $t$), the incremental update of $C_{t,T}(q)$ is non-trivial. Thus, we focus on how to update $C_{t,T}(q)$ incrementally by exploiting its previous value $C_{t,T-1}(q)$. In other words, we formulate $C_{t,T}(q) = C_{t,T-1}(q) + \alpha$ and focus on how to accurately compute $\alpha$ so as to produce the result which is the same as that obtained by computing $C_{t,T}(q)$ from scratch. Towards this end, we propose to let $\alpha$ be an aggregation of *delta-contribution scores* propagated by $q$'s adjacent (via inlinks) pages $p$ ($p \in P_{engaged}$), denoted as follows:

$$\alpha = \sum_{p} \Delta AC_{t,T}(q,p)$$

In other words,

$$C_{t,T}(q) = C_{t,T-1}(q) + \sum_{p} \Delta AC_{t,T}(q,p) \tag{11}$$

where $\Delta AC_{t,T}(q,p)$ is a *delta-contribution score* of $q$ on $p$ with respect to $t$ (i.e., $p$ and $q$'s common keyword) at $T$. $\Delta AC_{t,T}(q,p)$ is computed as follows:

$$\Delta AC_{t,T}(q,p) = PC_{t,T}(q,p) - PC_{t,T-1}(q,p) \tag{12}$$

where $PC_{t,T}(q,p)$ is the sum of all of the contribution scores that $p$ propagates to $q$ *directly* and *indirectly* (i.e., the scores that other pages propagate to $q$ through $p$) until $T$, computed as follows:

$$PC_{t,T}(q,p) = \alpha^1_{t,T}(q,p)(R_{t,T}(p) + \sum_{i=1}^{d-1} C^i_{t,T-1}(p)) \tag{13}$$

where $\alpha^1_{t,T}(q,p)R_{t,T}(p)$ represents the contribution score propagated directly to $q$ and $\alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p)$ denotes the scores indirectly propagated to $q$. Here, $C^i_{t,T-1}(p)$ is the sum of the contribution scores that $p$ has received from other pages pointing to it indirectly through the paths of length $i$ until $T-1$, computed as follows:

$$C^i_{t,T-1}(q) = \sum_{p \in D(q,i)} \sum_{(q,p) \in p \xrightarrow{i} q} C_{t,T-1}(q,p) \tag{14}$$

For completeness, in the Appendix section, we provide a proof that Eq. (11), which is our proposal, *always* holds true.

### 4.4. Algorithm

This section provides the detailed algorithm of our incremental C-Rank equipped with *d-CSP*. Lines 1 to 17 describe the initialization process.

Note that, at $T_w$, $R_{t,T_w}(p)$, $C^i_{t,T_w}(p)$, $\alpha^1_{t,T_w}(q,p)$, and $CR_{t,T_w}(p)$ have already been computed. At $T_w + 1$, we first update the relevance scores, extract a set of keywords, and update the contribution ratio (see lines 1 to 13). Then, we figure out which pages should be included in $P_{target}$ and $P_{engaged}$ for the first iteration (see lines 14 to 17). Then, lines 20 to 39 describe the details of our *d-CSP*, which can be summarized as follows:

1. Iterate the following steps 2 to 5 unless $P_{engaged}$ is empty.
2. Identify $P_{target}$ form $P_{engaged}$ which are *specific to this iteration* (see Section 4.2).
3. Make $P_{engaged}$ propagate the *delta-contribution scores* to $P_{target}$.
4. Compute the C-Rank scores of pages in $P_{target}$.
5. Make the pages included in $P_{target}$ form $P_{engaged}$ for *the next iteration*, if they received delta-contribution scores larger than zero.

More specifically, a *delta-contribution score* from $p$ to $q$, $\Delta AC_{t,T}(q,p)$, is computed using Eqs. (12) and (13) and then propagated to $q$. In the next iteration, the pages in $P_{target}$ become $P_{engaged}$ to re-propagate the received scores. If the received score is zero, the page cannot be a member of $P_{engaged}$ in the next iteration, as shown in lines 28 to 30, and the pages in the new $P_{target}$ are obtained. Then, $P_{engaged}$ propagates *delta-contribution scores* to $P_{target}$. *d-CSP* ends when the sum of all of the *delta-contribution scores* propagated are zero, which indicates that there is no further change of any C-Rank scores of $P_{target}$. Line 18 controls such termination condition of *d-CSP* (also see lines 28 to 30).

**Algorithm 2** Incremental C-Rank

**Input:** $N$, $d$, $\lambda$, $T_w$, $P_{old}$, $P_{chunk}$, $R_{t,T_w}(p)$, $C^i_{t,T_w}(p)$, $\alpha^1_{t,T_w}(q,p)$, $CR_{t,T_w}(p)$ of every term in every page in $P_{old}$
**Output:** A C-Rank score of every term in every page in $P_{old} \cup P_{chunk}$.

**Initialization**
$T = T_w + 1$
**foreach** $p \in P_{chunk}$ **do**
  **foreach** $t \in term(p)$ **do**
    Compute $R_{t,T}(p)$
  Extract top-$N$ terms in page $p$ as $key(p)$
**foreach** $p \in P_{chunk}$ **do**
  **foreach** $q \in outlink(p)$ **do**
    **foreach** $t \in key(p) \cap key(q)$ **do**
      Compute $\alpha^1_{t,T}(q,p)$ with Eq. (9)
  **foreach** $l \in inlink(p)$ **do**
    **foreach** $m \in outlink(l)$ **do**
      **foreach** $t \in key(l) \cap key(m)$ **do**
        Compute $\alpha^1_{t,T}(m,l)$ with Eq. (9)
Find $P_{target}$ with Steps A and B
**foreach** $u \in P_{target}$ **do**
  **foreach** $v \in outlink(u)$ **do**
    Insert page $v$ into $P_{engaged}$ // see Step C

**delta-Contribution Score Propagation**
**while** $P_{engaged} \neq \emptyset$ **do**
  **foreach** $p \in P_{engaged}$ **do**
    **foreach** $q \in outlink(p)$ **do**
      **foreach** $t \in key(q)$ **do**
        $tempC_t(q) = 0$
  **foreach** $p \in P_{engaged}$ **do**
    **foreach** $q \in outlink(p)$ **do**
      **foreach** $t \in key(p) \cap key(q)$ **do**
        Compute $PC_{t,T}(q,p)$ with Eq. (13)
        Compute $\Delta AC_{t,T}(q,p)$ with Eq. (12)
        **if** $\Delta AC_{t,T}(q,p) \neq 0$ **then**
          $tempC_t(q) += \Delta AC_{t,T}(q,p)$
          Insert page $q$ into $P_{target}$
  **foreach** $q \in P_{target}$ **do**
    **foreach** $t \in key(q)$ **do**
      $C_{t,T}(q) = C_{t,T-1}(q) + tempC_t(q)$
      $CR_{t,T}(q) = \lambda R_{t,T}(q) + (1-\lambda) C_{t,T}(q)$
  $P_{engaged} = P_{target}$
  Clear $P_{target}$
  $T = T + 1$
**return** $CR(P)$

## 5. Evaluation

This section evaluates the efficiency and effectiveness of our proposed method through extensive experiments with a real-life dataset.

- **Datasets.** We used two real-world datasets in the evaluation: (1) the .GOV Web graph dataset introduced by the Web Track of TREC 2003 and 2004 [44,45], which consists of approximately 1.25M Web pages and 11.2M hyperlinks, and (2) the DBLP academic paper dataset containing 18,276 scientific papers and 112,198 citations, introduced by Microsoft Academic Search. We preprocessed this data in the same way as has been done in other works [35–37]. For convenience, we denote $P_{all}$ as the entire set of pages in both datasets.
- **Parameters.** We set the cutoff path length $d$ at a value of 3, the number of common keywords $N$ at 10, and $\lambda$ at 0.8, which are the best performing parameters used in static C-Rank [34]. We also used *Okapi BM25* [17] as a calculator of relevance scores, following a previous method [34].
- **Environment.** All of the experiments were conducted on a Windows7 64-bit operating system equipped with an i7-4770k Intel CPU and 32 GB RAM.

### 5.1. Computation time analysis

In this experiment, we first analyzed how efficiently our method reflects *Web changes*. For this experiment, we selected 90% of pages of $P_{all}$ as $P_{old}$ and the remaining 10% pages as $P_{new}$. We chose the pages to be included in $P_{old}$ and $P_{new}$ in a random way. Then, we randomly divided $P_{new}$ into multiple $P_{chunk}$ having a certain number of pages, e.g., 0.1%, 1%, and 10% of $P_{new}$. We regard each $P_{chunk}$ as *Web changes* and thus, we incrementally added each $P_{chunk}$ to $P_{old}$. For each addition, we measured the time to update the C-Rank scores of pages in $P_{old} \cup P_{chunk}$. Since we randomly built $P_{old}$ and $P_{new}$, we independently carried out the experiments 3 times and took the average of their update times. We compared the update time consumed by our proposed incremental C-Rank with those obtained by baselines including static C-Rank and the naive method introduced in Section 3. In addition, to the best of our knowledge, no *hybrid* ranking method has been proposed to incrementally reflect dynamic Web changes. Instead, some *static* hybrid methods which do not consider an incremental update have been proposed. We have already performed comparative experiments of these methods in our prior work [34]. Based on the previous experimental results [34], these hybrid ranking methods were less efficient than the static C-Rank algorithm proposed previously [34]. Moreover, to the best of our knowledge, none of these hybrid ranking methods have been extended to the incremental version. For this reason, it seems best to compare our method with the static C-Rank and the naive (but dynamic) C-Rank. In the case of the static C-Rank, we only measured the computation time taken to reflect the first chunk, since it would take too much time to reflect every chunk from scratch.

First of all, to see the overall tendency at a glance, we chose one chunk randomly and measured its update time using each method. Fig. 4 shows the results, where each graph reports the results on each dataset. The $x$-axis represents the size of $P_{chunk}$ compared to $P_{new}$ (i.e., $\frac{|P_{chunk}|}{|P_{new}|}$) and the $y$-axis shows the update time on a log scale for TREC and on a linear scale for DBLP. We confirmed that, in all cases, our incremental C-Rank outperformed both baselines significantly. In particular, when $\frac{|P_{chunk}|}{|P_{new}|}$ is 0.01 in Fig. 4(a), our proposed method is about 252 times faster than static C-Rank, as well as demonstrating approximately a 35% reduced computation time compared to the naive method. We also confirmed that the updated C-Rank score of every term in every page is exactly the same in static C-Rank, the naive method, and our incremental C-Rank. This demonstrates that our incremental C-Rank does not trade accuracy for higher performance.

More detailed results are provided in Figs. 5, 6 and 7, which each contains two graphs showing the computation time when reflecting the stream of $P_{chunk}$ on the two datasets. The $x$-axis represents the ID of each $P_{chunk}$ and the $y$-axis is the computation time to reflect the corresponding $P_{chunk}$. From the experimental results, we again confirm that our *incremental C-Rank* significantly outperforms static C-Rank and the naive method. The main reasons for such a difference are that (1) our method selects a greatly reduced number of pages as $P_{engaged}$ compared to the baselines, and (2) our *d-CSP* allows only score propagation between two

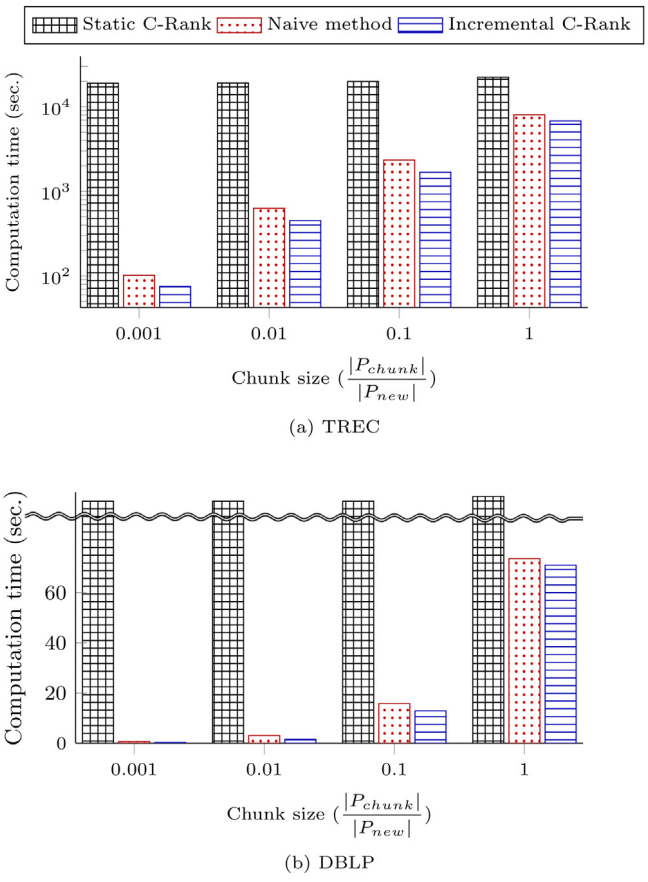


**Fig. 4.** Computation time according to $\frac{|P_{chunk}|}{|P_{new}|}$.

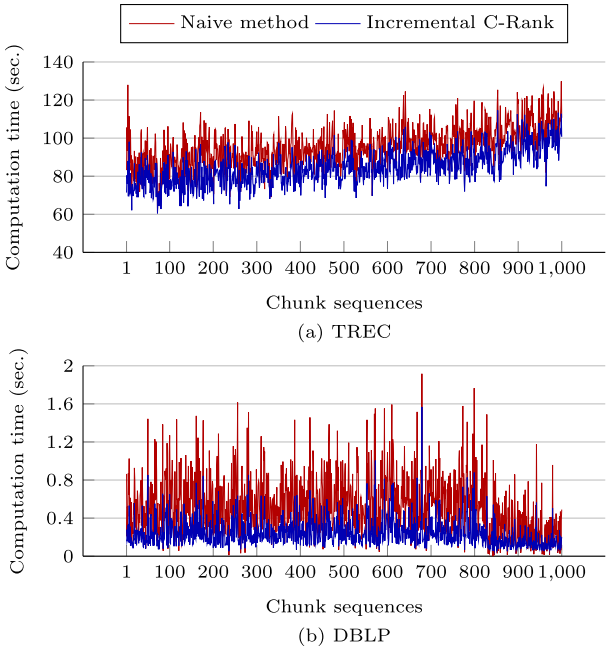


**Fig. 5.** Computation time ($\frac{|P_{old}|}{|P_{all}|} = 0.9$, $\frac{|P_{chunk}|}{|P_{new}|} = 0.001$).

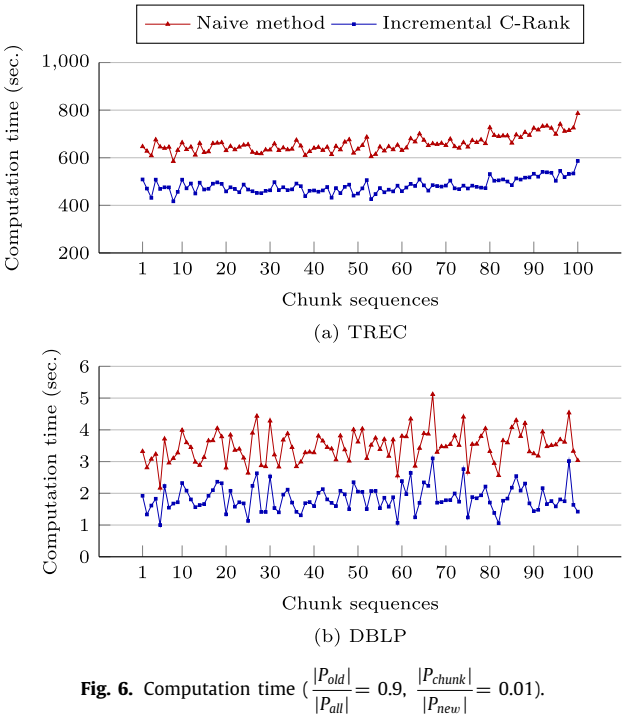


**Fig. 6.** Computation time ($\frac{|P_{old}|}{|P_{all}|} = 0.9$, $\frac{|P_{chunk}|}{|P_{new}|} = 0.01$).

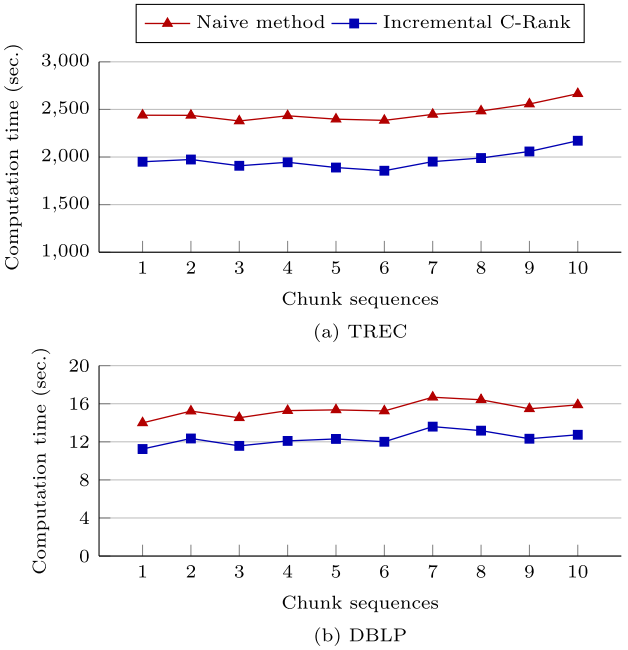


**Fig. 7.** Computation time ($\frac{|P_{old}|}{|P_{all}|} = 0.9$, $\frac{|P_{chunk}|}{|P_{new}|} = 0.1$).

pages directly connected with each other. These two reasons result in a significantly reduced number of computations for contribution scores, which makes the overall C-Rank update process much more efficient.

Lastly, we empirically examined whether static C-Rank and the proposed incremental C-Rank produce the exact same results through experiments. Towards this goal, whenever *Web changes* occurred, we tested whether the C-Rank scores from each page calculated from static C-Rank and incremental C-Rank were equal or not. As a result, we found that, in the two datasets, both algorithms always return the same C-Rank scores without any exceptions.

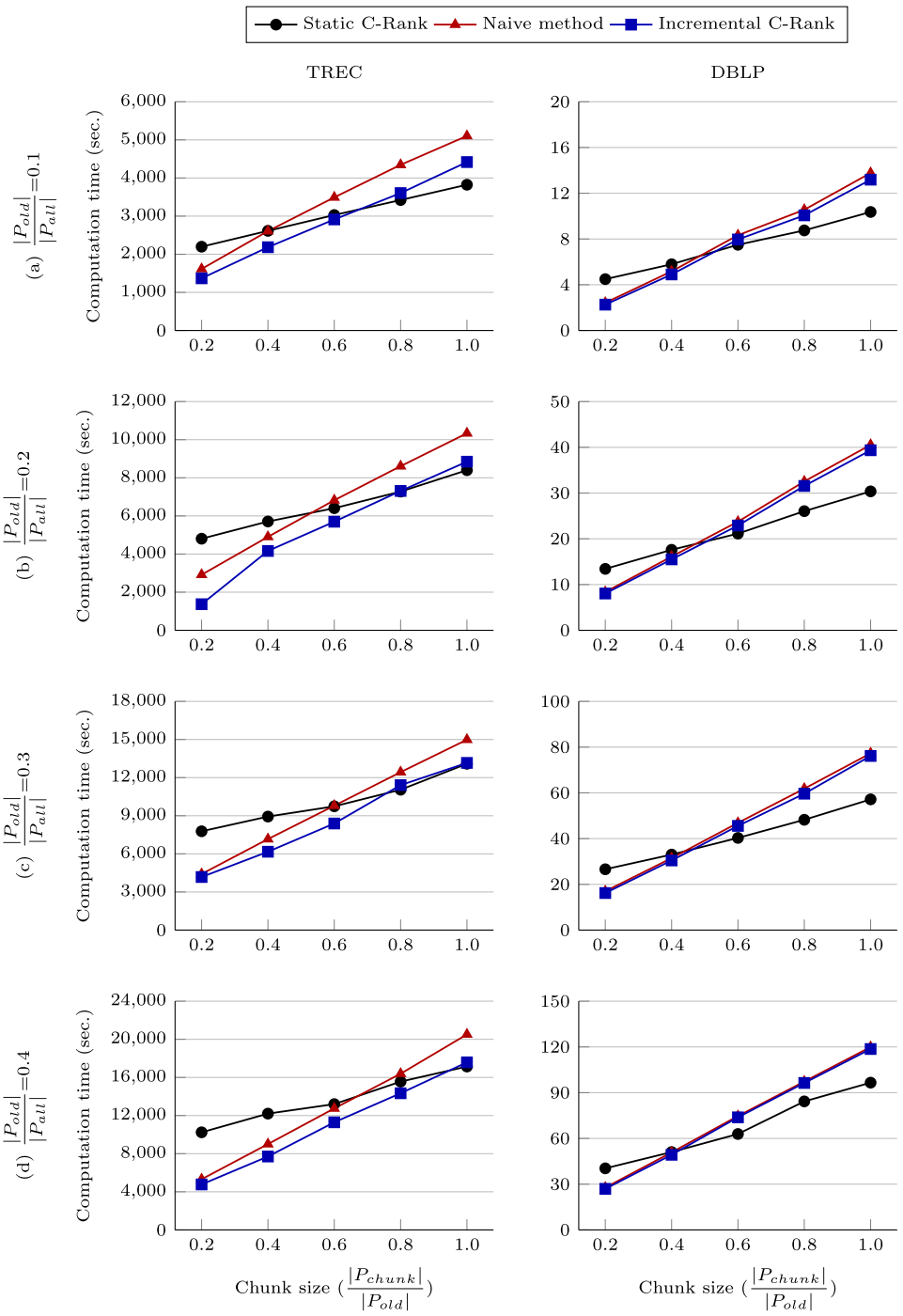


**Fig. 8.** Computation time according to $\frac{|P_{chunk}|}{|P_{old}|}$.

### 5.2. Scalability analysis

In the previous experiment, we focused on the computation time required for reflecting $P_{chunk}$ of different sizes on $P_{old}$ of a fixed size. In this experiment, we examined the tendency of the computation time under $P_{old}$ of different sizes to evaluate the scalability of each method. To this end, we composed $P_{old}$ of different sizes by varying $\frac{|P_{old}|}{|P_{all}|}$ from 0.1 to 0.4. Given each $P_{old}$, we then made a chunk of its size set with values of $\frac{|P_{chunk}|}{|P_{old}|}$ = 0.2, 0.4, 0.6, 0.8, and 1.0, and measured the computation time.

Table 2 summarizes the experimental results on each dataset. In a cell, 'naive/static' and 'incremental/static' are the ratios of the computation time of the naive method and incremental C-Rank to that of static C-Rank, respectively (the smaller, the better). We observed that the ratio for both the naive method and incremental C-Rank decreases as the size of $P_{old}$ increases, which indicates good scalability. In addition, our incremental C-Rank consistently provides smaller values of the ratio than the naive method in all cases, which indicates that the scalability of our incremental C-Rank is better than the naive method. Note that, as the size of $P_{old}$ increases, the number of pages and paths engaged in the computations or propagations of contribution scores tends to increase, and thus the computation time increases as well.

**Table 2**
Scalability analysis results.

(a) TREC

| $\frac{\lvert P_{chunk}\rvert}{\lvert P_{old}\rvert}$ | Ratio | $\frac{\lvert P_{old}\rvert}{\lvert P_{all}\rvert}$ 0.1 | 0.2 | 0.3 | 0.4 |
|---|---|---|---|---|---|
| 0.2 | naive/static | 0.7345 | 0.6066 | 0.5677 | 0.5191 |
| | incremental/static | 0.6227 | 0.5264 | 0.5372 | 0.4659 |
| 0.4 | naive/static | 0.9974 | 0.8587 | 0.8017 | 0.7380 |
| | incremental/static | 0.8358 | 0.7292 | 0.6902 | 0.6313 |
| 0.6 | naive/static | 1.1529 | 1.0652 | 1.0051 | 0.9659 |
| | incremental/static | 0.9619 | 0.8894 | 0.8612 | 0.8555 |
| 0.8 | naive/static | 1.2700 | 1.1831 | 1.1251 | 1.0531 |
| | incremental/static | 1.0534 | 1.0045 | 1.0327 | 0.9215 |
| 1.0 | naive/static | 1.3346 | 1.2306 | 1.1447 | 1.1965 |
| | incremental/static | 1.1555 | 1.0532 | 1.0051 | 1.0257 |

(b) DBLP

| $\frac{\lvert P_{chunk}\rvert}{\lvert P_{old}\rvert}$ | Ratio | $\frac{\lvert P_{old}\rvert}{\lvert P_{all}\rvert}$ 0.1 | 0.2 | 0.3 | 0.4 |
|---|---|---|---|---|---|
| 0.2 | naive/static | 0.5395 | 0.6234 | 0.6386 | 0.6872 |
| | incremental/static | 0.5034 | 0.5989 | 0.6092 | 0.6672 |
| 0.4 | naive/static | 0.8930 | 0.9163 | 0.9532 | 0.9959 |
| | incremental/static | 0.8462 | 0.8813 | 0.9223 | 0.9649 |
| 0.6 | naive/static | 1.1111 | 1.1235 | 1.1632 | 1.1871 |
| | incremental/static | 1.0618 | 1.0838 | 1.1303 | 1.1738 |
| 0.8 | naive/static | 1.2065 | 1.2496 | 1.2822 | 1.1547 |
| | incremental/static | 1.1505 | 1.2116 | 1.2365 | 1.1436 |
| 1.0 | naive/static | 1.3287 | 1.3351 | 1.3522 | 1.2419 |
| | incremental/static | 1.2724 | 1.2962 | 1.3321 | 1.2285 |

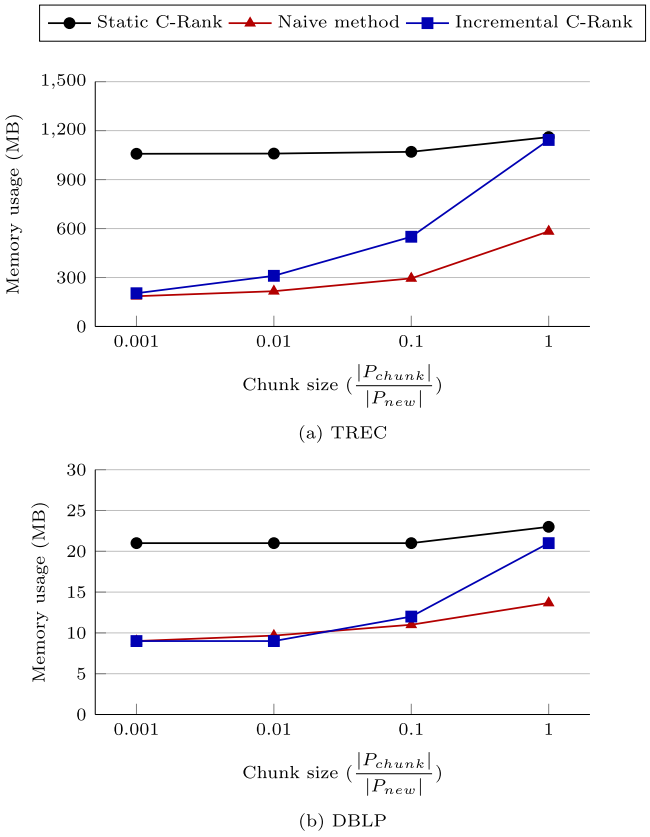


**Fig. 9.** Memory usage according to $\frac{|P_{chunk}|}{|P_{new}|}$.

However, since our method aims to reduce the number of score computations and propagations as much as possible, while the baselines do not, it is robust to the increased size of $P_{old}$.

### 5.3. Break-even point analysis

Our method (and the naive method) is optimized to make the number of updated pages as small as possible but requires additional computations for pre-processing to achieve such a selective update. Thus, it is obvious that, if $|P_{chunk}|$ is sufficiently large, static C-Rank may perform even faster than our incremental C-Rank (or the naive method). We can see such a tendency in Table 2 where when $\frac{|P_{chunk}|}{|P_{old}|}$ is 0.6 and 0.8 in TREC, static C-Rank result in a computation time comparable to or even faster than our method and the naive method. In this regard, a *break-even point* exists below/above which static C-Rank performs faster/slower than the other two methods.

Fig. 8 reports the break-even points obtained under various experimental settings, where the *x*-axis represents the ratio of the size of $P_{chunk}$ to that of $P_{old}$ (i.e., $\frac{|P_{chunk}|}{|P_{old}|}$) and the *y*-axis is the computation time. The graphs on the left side show the results for the TREC dataset and those on the right side are the results for the DBLP dataset. In each figure, the intersections of the two lines indicate the break-even points. For instance, in TREC in Fig. 8(a), the break-even point between our method and static C-Rank appears when $\frac{|P_{chunk}|}{|P_{old}|}$ is close to 0.6 and that between the naive method and static C-Rank is around 0.4. Remarkably, in all cases for both TREC and DBLP, we observe that the break-even points between our method and static C-Rank appear on the right side of those between the naive method and static C-Rank, which again implies the superiority of our method in reflecting *Web changes*.

### 5.4. Memory usage analysis

Based on previous experiments, we empirically showed that the number of pages/paths engaged in the computations/propagations of contribution scores is the critical factor for a high update performance. On the other side, these factors may have effects on the memory usage, which is another important consideration in real-life search engines. To validate this point, in this experiment, we measured how much memory each method requires to update the C-Rank scores. Note that the experimental settings employed here are the same as those used in Section 5.1.

Fig. 9 shows the amount of memory usage required by static C-Rank, the naive method, and our incremental C-Rank on each dataset. We confirmed that incremental C-Rank and the naive method require less memory space than static C-Rank when reflecting each chunk. This is because static C-Rank needs to deal with all of the pages in $P_{old} \cup P_{chunk}$ regardless of whether a page is affected by $P_{chunk}$, whereas our incremental C-Rank and the naive method first carefully identify the pages necessary to reflect the *Web changes* and then load only those pages to memory. We also observe that our incremental C-Rank consumes more memory than the naive method. This is because it needs to load more information related to the computation of C-Rank scores than the naive method.

## 6. Conclusions

*C-Rank* is a contribution-based ranking algorithm which combines content and link information effectively by using the concept of contribution. C-Rank is known to provide high performance in terms of both accurate and prompt responses to user queries. However, it suffers from a very high cost to reflect *Web*

*changes* because it re-computes all the C-Rank scores used for ranking from scratch. Motivated by the limitation, this paper discusses how to promptly update C-Ranks scores according to the *Web changes*. We first illustrated a simple, naive solution which can selectively update the pages affected by *Web changes* with slight modification of static C-Rank. We then proposed our incremental C-Rank, which was carefully designed to be more efficient at updating C-Rank scores, under the objectives of reducing the number of contribution score computations and reducing the number of score propagations. We proposed *d-CSP* as a core part for realizing our proposal. We also formally proved that using *d-CSP* produces the same results as static C-Rank. We performed extensive experiments including analyses of the efficiency, scalability, break-even point, and memory usage of our method as well as those of baselines. Remarkably, our incremental C-Rank reflects *Web changes* up to 252 times faster than static C-Rank according to the batch size. Also, our incremental C-Rank effectively reflects *Web changes* without any accuracy loss and thereby it is suitable for dynamic Web environments.

## Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP; Ministry of Science and ICT) (No. NRF-2017R1A2B3004581).

## Appendix

We provide a proof that our incremental C-Rank preserves C-Rank scores which are exactly the same as those obtained from static C-Rank after *d-CSP* through Eq. (11), as follows:

**Lemma 1.**

$$\sum_{p\in D(q,1)} PC_{t,T}(q,p) = \sum_{i=1}^{d} C^i_{t,T}(q)$$

**Proof.** According to Eq. (13), the following is obtained:

$$\begin{aligned}\sum_{p\in D(q,1)} PC_{t,T}(q,p) &= \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\Big(R_{t,T}(p) + \sum_{i=1}^{d-1} C^i_{t,T-1}(p)\Big)\\ &= \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\Big(R_{t,T}(p)\\ &\quad + \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p)\Big)\\ &= \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)R_{t,T}(p)\\ &\quad + \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p)\end{aligned}$$

First of all, according to Eq. (8), the first term of Lemma 1 can be rewritten as follows:

$$\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)R_{t,T}(p) = \sum_{p\in D(q,1)} C_{t,T}(q,p)$$

Here, we know that there exists only one path from $p$ to $q$ when they are directly connected, so the above equation can be rewritten as follows:

$$\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)R_{t,T}(p) = \sum_{p\in D(q,1)} \sum_{(q,p)\in p \xrightarrow{i} q} C_{t,T}(q,p)$$

Then, by Eq. (14), we can obtain the following:

$$\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)R_{t,T}(p) = C^1_{t,T}(q)$$

Second, according to Eq. (14), the second term of Lemma 1 can be rewritten as follows:

$$\begin{aligned}&\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p)\\ &= \sum_{p\in D(q,1)} \alpha^1_{t,T}\sum_{i=1}^{d-1}\sum_{r\in D(p,i)}\sum_{(p,r)\in r\xrightarrow{i} p} C_{t,T-1}(p,r)\end{aligned}$$

If $p \to q$, we can switch $D(p,i)$ and $(p,r) \in r \xrightarrow{i} p$ as $D(q,i+1)$ and $(q,r) \in r \xrightarrow{i+1} q$, respectively. Then, we have

$$\begin{aligned}\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p) &= \sum_{i=1}^{d-1}\sum_{r\in D(q,i)}\sum_{(q,r)\in r\xrightarrow{i+1} q} C_{t,T}(q,r)\\ &= \sum_{i=2}^{d}\sum_{r\in D(q,i)}\sum_{(q,r)\in r\xrightarrow{i} q} C_{t,T}(q,r)\end{aligned}$$

By Eq. (14), the above equation can be rewritten as follows:

$$\sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p) = \sum_{i=2}^{d} C^i_{t,T}(q)$$

Finally, by combining the above two results, we can derive the following:

$$\begin{aligned}\sum_{p\in D(q,1)} PC_{t,T}(q,p) &= \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)R_{t,T}(p)\\ &\quad + \sum_{p\in D(q,1)} \alpha^1_{t,T}(q,p)\sum_{i=1}^{d-1} C^i_{t,T-1}(p)\\ &= C^1_{t,T}(q) + \sum_{i=2}^{d} C^i_{t,T}(q) = \sum_{i=1}^{d} C^i_{t,T}(q) \quad \square\end{aligned}$$

From Lemma 1, we can obtain the following Theorem 2.

**Theorem 2.**

$$C_{t,T}(q) - C_{t,T-1}(q) = \sum_{p\in D(q,1)} \Delta AC_{t,T}(q,p)$$

**Proof.** First, it is obvious that the sum of $C^i_{t,T}(q)$ with respect to $i$ (i.e., the path length) is equal to $C_{t,T}(q)$. Based on this, the left side of the above equation can be rewritten as follows:

$$C_{t,T}(q) - C_{t,T-1}(q) = \sum_{i=1}^{d} C^i_{t,T}(q) - \sum_{i=1}^{d} C^i_{t,T-1}(q)$$

Second, by Eq. (12), the right side can be rewritten as follows:

$$\sum_{p\in D(q,1)} \Delta AC_{t,T}(q,p) = \sum_{p\in D(q,1)} \big(PC_{t,T}(q,p) - PC_{t,T-1}(q,p)\big)$$

Also, by Lemma 1, we can obtain the following equation:

$$\sum_{p\in D(q,1)} PC_{t,T}(q,p) = \sum_{p\in D(q,1)} C^i_{t,T}(q)$$

Hence,

$$\sum_{p\in D(q,1)} \Delta AC_{t,T}(q,p) = \sum_{i=1}^{d} C^i_{t,T}(q) - \sum_{i=1}^{d} C^i_{t,T-1}(q)$$

and finally, by combining the above two results, we can derive the following:

$$C_{t,T}(q) - C_{t,T-1}(q) = \sum_{i=1}^{d} C^i_{t,T}(q) - \sum_{i=1}^{d} C^i_{t,T-1}(q) = \sum_{p \in D(q,1)} \Delta AC_{t,T}(q,p) \quad \square$$

Consequently, we prove that $C_{t,T}(q)$ can be computed incrementally and the computed $C_{t,T}(q)$ is identical to that computed from scratch by static C-Rank.

## References


[1] C. Zhang, J. Bi, S. Xu, E. Ramento, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: An open-source software for multi-class imbalance learning, Knowl.-Based Syst. (2019).

[2] L. Bing, Z.-Y. Niu, P. Li, W. Lam, H. Wang, Learning a unified embedding space of web search from large-scale query log, Knowl.-Based Syst. 150 (2018) 38–48.

[3] M. Huang, J. Lin, Y. Peng, X. Xie, Design a batched information retrieval system based on a concept-lattice-like structure, Knowl.-Based Syst. 150 (2018) 74–84.

[4] J. Tekli, R. Chbeir, A.J. Traina, C. Traina Jr, Semindex+: A semantic indexing scheme for structured, unstructured, and partly structured data, Knowl.-Based Syst. 164 (2019) 378–403.

[5] E. Vicente-López, L.M. de Campos, J.M. Fernández-Luna, J.F. Huete, Use of textual and conceptual profiles for personalized retrieval of political documents, Knowl.-Based Syst. 112 (2016) 127–141.

[6] F. Huang, X. Zhang, Z. Li, Z. Zhao, Y. He, From content to links: social image embedding with deep multimodal model, Knowl.-Based Syst. 160 (2018) 251–264.

[7] H.-J. Kim, J. Lee, D.-K. Chae, S.-W. Kim, Crowdsourced promotions in doubt: analyzing effective crowdsourced promotions, Inform. Sci. 432 (2018) 185–198.

[8] X. Zhu, J. Huang, B. Zhou, A. Li, Y. Jia, Real-time personalized twitter search based on semantic expansion and quality model, Neurocomputing 254 (2017) 13–21.

[9] A. Konstantinidis, S. Pericleous, C. Charalambous, Meta-lamarckian learning in multi-objective optimization for mobile social network search, Appl. Soft Comput. 67 (2018) 70–93.

[10] W.-S. Hwang, J. Parc, S.-W. Kim, J. Lee, D. Lee, "Told you i didn't like it": Exploiting uninteresting items for effective collaborative filtering, in: Proceedings of the IEEE 32nd International Conference on Data Engineering, 2016, pp. 349–360.

[11] J. Lee, D. Lee, Y.-C. Lee, W.-S. Hwang, S.-W. Kim, Improving the accuracy of top-n recommendation using a preference model, Inform. Sci. 348 (2016) 290–304.

[12] D.-K. Chae, J.-S. Kang, S.-W. Kim, J.-T. Lee, Cfgan: A generic collaborative filtering framework based on generative adversarial networks, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 2018, pp. 137–146.

[13] N. Duhan, A. Sharma, K.K. Bhatia, Page ranking algorithms: A survey, in: Proceedings of the IEEE International Advance Computing Conference (IEEE IACC), 2009, pp. 1530–1537.

[14] N. Hochstotter, D. Lewandowski, What users see–structures in search engine results pages, Inform. Sci. 179 (12) (2009) 1796–1812.

[15] Q. Ai, K. Bi, J. Guo, W.B. Croft, Learning a deep listwise context model for ranking refinement, in: Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, 2018, pp. 135–144.

[16] C.D. Manning, P. Raghavan, H. Schutze, Introduction to Information Retrieval, Vol. 1, Cambridge University Press, 2008.

[17] K.S. Jones, S. Walker, S.E. Robertson, A probabilistic model of information retrieval: development and comparative experiments: Part 2, Inf. Process. Manage. 36 (6) (2000) 809–840.

[18] B. He, J.X. Huang, X. Zhou, Modeling term proximity for probabilistic information retrieval models, Inform. Sci. 181 (14) (2011) 3017–3031.

[19] O. Kurland, L. Lee, Pagerank without hyperlinks: Structural reranking using links induced by language models, ACM Trans. Inf. Syst. (TOIS) 28 (4) (2010) 18.

[20] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632.

[21] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[22] M. Yoon, W. Jin, U. Kang, Fast and accurate random walk with restart on dynamic graphs with guarantees, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, 2018, pp. 409–418.

[23] I. Chibane, B.-L. Doan, Relevance propagation model for large hypertext document collections, in: Large Scale Semantic Access to Content (text, image, video, and sound), 2007, pp. 585–595.

[24] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, W.-Y. Ma, A study of relevance propagation for web search, in: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2005, pp. 408–415.

[25] A. Shakery, C. Zhai, Relevance propagation for topic distillation uiuc trec 2003 web track experiments, in: TREC, 2003, pp. 673–677.

[26] A. Shakery, C. Zhai, A probabilistic relevance propagation model for hypertext retrieval, in: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, ACM, 2006, pp. 550–558.

[27] R. Song, J.-R. Wen, S. Shi, G. Xin, T.-Y. Liu, T. Qin, X. Zheng, J. Zhang, G.-R. Xue, W.-Y. Ma, Microsoft research asia at web track and terabyte track of trec 2004, in: TREC, 2004.

[28] T.H. Haveliwala, Topic-sensitive pagerank, in: Proceedings of the 11th International Conference on World Wide Web, 2002, pp. 517–526.

[29] S.K. Pal, B.L. Narayan, A web surfer model incorporating topic continuity, IEEE Trans. Knowl. Data Eng. 17 (5) (2005) 726–729.

[30] P. Boldi, M. Santini, S. Vigna, Pagerank as a function of the damping factor, in: Proceedings of the 14th International Conference on World Wide Web, 2005, pp. 557–566.

[31] R. Baeza-Yates, P. Boldi, C. Castillo, Generalizing pagerank: Damping functions for link-based ranking algorithms, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2006, pp. 308–315.

[32] L. Nie, B.D. Davison, X. Qi, Topical link analysis for web search, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2006, pp. 91–98.

[33] L. Nie, B.D. Davison, Separate and inequal: Preserving heterogeneity in topical authority flows, in: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2008, pp. 443–450.

[34] D.-J. Kim, S.-C. Lee, H.-Y. Son, S.-W. Kim, J.B. Lee, C-rank and its variants: A contribution-based ranking approach exploiting links and content, J. Inf. Sci. 40 (6) (2014) 761–778.

[35] M.R. Hamedani, S.-W. Kim, Simcs: An effective method to compute similarity of scientific papers based on contribution scores, IEICE Trans. Inf. Syst. 98 (12) (2015) 2328–2332.

[36] M.R. Hamedani, S.-W. Kim, D.-J. Kim, Simcc: A novel method to consider both content and citations for computing similarity of scientific papers, Inform. Sci. 334 (2016) 273–292.

[37] M.R. Hamedani, S.-W. Kim, Simcc-at: A method to compute similarity of scientific papers with automatic parameter tuning, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2016, pp. 1005–1008.

[38] S. Marcos-Pablos, F.J. García-Peñalvo, Information retrieval methodology for aiding scientific database search, Soft Comput. (2019) 1–10.

[39] B. Shi, L. Yang, T. Weninger, Forward backward similarity search in knowledge networks, Knowl.-Based Syst. 119 (2017) 20–31.

[40] M. Trevisani, A. Tuzzi, Learning the evolution of disciplines from scientific literature: A functional clustering approach to normalized keyword count trajectories, Knowl.-Based Syst. 146 (2018) 129–141.

[41] G. Wang, X. He, C.I. Ishuga, Har-si: A novel hybrid article recommendation approach integrating with social information in scientific social network, Knowl.-Based Syst. 148 (2018) 85–99.

[42] Y.A. Kim, G.W. Park, Topic-driven socialrank: Personalized search result ranking by identifying similar, credible users in a social network, Knowl.-Based Syst. 54 (2013) 230–242.

[43] J. Koo, D.-J. Kim, D.-K. Chae, S.-W. Kim, Incremental maintenance of c-rank scores in dynamic web environment, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 1570–1574.

[44] N. Craswell, D. Hawking, Overview of the trec 2004 web track, in: TREC, 2004, pp. 1–9.

[45] M.R. Hamedani, S.-W. Kim, Jacsim: An accurate and efficient link-based similarity measure in graphs, Inform. Sci. 414 (2017) 203–224.