

Chapter 2

Optimization Techniques for Multiple Centrality Computations

Christian von der Weth, Klemens Böhm, and Christian Hütter

Abstract A broad range of data has a graph structure, such as the Web link structure, online social networks, or online communities whose members rate each other (reputation systems) or rate items (recommender systems). In these contexts, a common task is to identify important vertices in the graph, e.g., influential users in a social network or trustworthy users in a reputation system, by means of centrality measures. In such scenarios, several centrality computations take place at the same time, as we will explain. With centrality computation being expensive, performance is crucial. While optimization techniques for single centrality computations exist, little attention so far has gone into the computation of several centrality measures in combination. In this paper, we investigate how to efficiently compute several centrality measures at a time. We propose two new optimization techniques and demonstrate their usefulness both theoretically as well as experimentally on synthetic and on real-world data sets.

2.1 Introduction

Centrality measures [30] allow identifying important vertices in graphs. Such measures assign a numerical score to each vertex to quantify its importance among all nodes, based on the graph structure. On a large scale, Eigenvector-based measures have been successfully applied to the Web graph to rank search

C. von der Weth (✉)

Digital Enterprise Research Institute (DERI), National University of Ireland, Galway (NUIG),
Galway, Ireland

e-mail: christian.vonderweth@deri.org

K. Böhm · C. Hütter

Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany

e-mail: klemens.boehm@kit.edu; christian.huetter@kit.edu

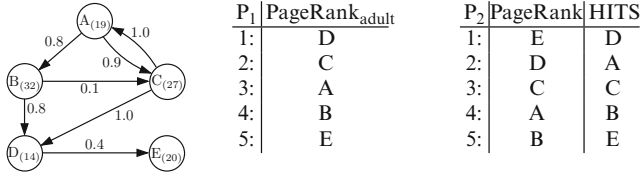


Fig. 2.1 Example of a feedback graph

results, e.g., [16, 19]. Further applications include online social network analysis to identify individuals that are influential, e.g., [6, 13], and recommender systems to find similar items, e.g., [1, 35]. In this work, we use reputation systems [21] as our running example. With such systems, participants of social networks or members of online communities can issue *feedback*, i.e., rate previous interactions with others. Reputation systems differ in the way they process feedback to derive the reputation of participants. Various existing reputation systems [15, 22, 33] make use of centrality measures, the feedback forms a graph, the *feedback graph*. Vertices are the participants, edges represent the feedback items. However, existing reputation systems use fixed evaluation schemes to determine the reputation of participants, i.e., a centrality measure together with fixed parameter values is given. This is restrictive from a user perspective. We envision a more flexible system where participants can formulate *behavioral trust policies*. Such policies allow participants to specify how to compute the reputation of others. This includes the flexible application of various centralities measures on arbitrary subgraphs of the feedback graph; see the following example.

Examples 1. Figure 2.1 (left) shows a feedback graph. The nodes A–D represent the individuals. The label of each node is its age. Edges represent feedback between individuals. Edge weights represent the rating scores of the corresponding feedback items. Various trust policies are conceivable, e.g.:

P_1 : “A participant is trustworthy if he belongs to the top 2 individuals according to PageRank based on the feedback from adult participants (PageRank_{adult}).”

P_2 : “A participant is trustworthy if he belongs to the top 2 individuals according to both PageRank and HITS.”

P_1 is true for members D and C, P_2 is true for member D; see Fig. 2.1 (right). This example illustrates that the evaluation of trust policies may require several centrality computations at the same time. \square

When participants formulate policies themselves, we observe that several centrality computations take place at the same time, for the following reasons: (a) While centrality measures in general are accepted to determine the reputation of individuals, it is not obvious which one should be used in which situation [27]. An approach from data mining is to compute several measures at a time and to pay attention to the data objects whose values regarding one measure are distinctive. (b) In environments with a large number of individuals, the frequency of interactions

is high; hence, the number of policies that need to be evaluated at the same time is large. (c) Literature has investigated attacks against centrality-based reputation systems and has proposed countermeasures [34]. A proposal that identifies vertices with different centrality indices for ‘similar’ centrality measures is particularly promising.

To illustrate that our concern is broad, we briefly leave aside our running example. In social network analysis (see Example 2), one might be interested in the centrality values of individuals according to different centrality measures (Q_1), in the centrality values of individuals with respect to different subgraphs of the network (Q_2), or in the temporal development of centrality values, e.g., by looking at snapshots of a social network (Q_3).

Examples 2. Consider the following analysis questions given the friendship network between members of a social networking platform like FACEBOOK:

Q_1 : “Who is most influential according to both PAGERANK and HITS?”

Q_2 : “What is the PAGERANK value of a female member with respect to the complete friendship network and with respect to the network consisting of female members only?”

Q_3 : “The PAGERANK value of which member shows the most significant increase over the last month?”

□

From a performance perspective, centrality computation is expensive. While the optimization of a single centrality computation has been addressed, e.g., [14, 17], the computation of more than one measure at a time is challenging and remains to be investigated. This work focuses on *Eigenvector-based centrality measures* where the centrality score of a vertex depends on the centrality scores of the vertices it is adjacent to. PageRank [19] and HITS [16] serve as examples. Algorithms for these measures incorporate the power method [10], a numerical method to compute the largest Eigenvector of a matrix. This chapter makes the following contributions.

Classification of combined centrality computation. Centrality computations may ‘overlap’ in various ways, i.e., they share different commonalities which give way to a combined computation. For instance, one may want to compute two (different) measures on the same data set or to compute the same measure on data sets that are slightly different. We provide a classification of these variants of overlap and say how centrality computations can be combined in each case, at least in theory.

Techniques for combined centrality computation. We propose two new optimization approaches: (1) *Loop fusion techniques* exploit commonalities of centrality computations on the physical level, i.e., the data structures representing the graph data as well as the algorithms, allowing for the computation of several measures within one execution of the power method. (2) The power method starts with an initialization of the centrality values and adapts them in subsequent iterations. *Re-use of computation results* means that the result of a previous centrality computation serves as initialization. This results in fewer iterations.

Theoretical analysis. The improvements due to loop fusion depend on the internal representation of the graph data. Thus, we first present the graph representation we have used for our evaluation. Based on this we provide a theoretical analysis of the performance of loop fusion in the worst and in the best case as well as in average cases. One result is that the improvement depends on how different graph data overlaps. While the analysis is general in its nature, we present specific numbers for our graph representation, to compare them with our experimental results.

Evaluation. We have carried out extensive experiments both with synthetic and with real-world data to investigate the performance improvements of our optimization techniques. We conducted experiments for each ‘overlap variant’ according to our classification. Our main results are as follows: While re-use yields an improvement in specific situations only, loop fusion always yields an improvement. However, its extent in quantitative terms depends on various parameters and may be difficult to predict. For example, fusing centrality computations on the same data set yields a much higher speed-up than fusing centrality computations on different sets.

Chapter outline: Section 2.2 reviews related work. We discuss centrality computation in Sect. 2.3. Section 2.4 presents our optimizations and Sect. 2.5 our implementation. Section 2.6 presents the theoretical analysis regarding the effect of loop fusion. Section 2.7 features an evaluation. Section 2.8 concludes. This article is an extended version of [28]. It contains a comprehensive theoretical analysis of loop fusion and an extended evaluation section.

2.2 Related Work

Researchers have successfully applied centrality measures on graphs in various settings, see [9]. PageRank [19] and HITS [16] are popular measures for web-graph analysis. Variants of those algorithms have been proposed, e.g., personalization of the ranking [5, 12], stability of the algorithms [18], and distributed environments [20, 29]. Applications of centrality measures include social network analysis [6, 13] to identify influential participants, and recommender systems [1, 35] to find similar items. Several reputation systems rely on centrality measures, e.g., [15, 22, 33]. While these systems use fixed evaluation schemes for trust, [26] proposes behavioral trust policies based on centrality. von derWeth and Böhm [27] compares centrality measures for reputation systems. It shows that various measures yield good results, but only Eigenvector-based ones have good runtime performance as well. Other centrality-based reputation systems take into account social relations between participants [23, 24] and propose decentralized trust models for P2P systems [8, 32].

Loop fusion is a technique for compiler optimization [2, 3]. It exploits commonalities of loops (e.g., centrality computations) on the physical level. Optimization techniques for Eigenvector-based centrality computation deal with the power method itself. There are two main research thrusts, to modify the method to reduce

the work per iteration [17], and to reduce the number of iterations [14]. However, these optimizations exploit a priori knowledge regarding the structure of the graph. In general – given dynamic data sets like feedback – this knowledge is not available. Other optimizations focus on parallelization, e.g., [7, 31]. While parallelization can bring down absolute runtimes, the total extent of resources required is not reduced. We are the first to address the combined computation of several centrality measures.

2.3 Centrality Computation

In the following, we first review the power method which serves as the basis of centrality computation. See [10] for the mathematical background. We then categorize how centrality computations may differ from each other.

2.3.1 The Power Method

Eigenvector-based centrality-index values are the largest Eigenvector of an $n \times n$ Matrix M representing the graph structure, and n is the number of vertices. To compute the centrality index values, numerical methods are typically used. One is the so-called power method (Algorithm 1). It consists of a loop which multiplies a vector with M in each iteration. According to theory [10], for an arbitrary non-zero start vector \vec{v}_0 , \vec{v}_t converges to the largest Eigenvector of M . The process stops when the norm of the difference of \vec{v}_t and \vec{v}_{t-1} is smaller than a user-specified threshold ϵ , i.e., \vec{v}_t does not change significantly any more. Different centrality measures lead to different matrices M . Some measures have a argument list L ; Function f in Line 5 reflects this. To give an example, the PageRank definition features the so-called damping factor d , i.e., $L = \langle d \rangle$, and function f is defined as $f(\vec{v}_t, d) = (1 - d) \cdot \vec{\mathbf{1}} + d \cdot \vec{v}_t$, where $\vec{\mathbf{1}}$ is the vector with all elements equal to 1.

Thus, four parameters describe the execution of a power method for centrality measures: M , f , the measure-specific argument list L , and error threshold ϵ . – Our evaluation will cover PageRank, HITS and Positional Weakness (PW). The power iterations for these three measures are as follows:

$$\begin{aligned}\vec{v}_{\text{PageRank}}^{(k+1)} &= (1 - d) \vec{\mathbf{1}} + d \cdot \mathbf{T} \vec{v}_{\text{PageRank}}^{(k)} \\ \vec{v}_{\text{Authority}}^{(k+1)} &= \mathbf{A}^T \vec{v}_{\text{Hub}}^{(k)}, \quad \vec{v}_{\text{Hub}}^{(k+1)} = \mathbf{A} \vec{v}_{\text{Authority}}^{(k)} \\ \vec{v}_{\text{PW}}^{(k+1)} &= \frac{1}{|V|} \mathbf{B}^T (\vec{\mathbf{1}} + \vec{v}_{\text{PW}}^{(k)})\end{aligned}$$

$\mathbf{A} = (a_{ij})$ is the *adjacency matrix* of a graph $G(V, E)$ with $a_{ij} = w_{ij}$, where w_{ij} is the weight on an edge from v_i to v_j , 0 if there is no such edge. \mathbf{A}^T is the transposed adjacency matrix. $\mathbf{T} = (t_{ij})$ is the *transition matrix* of G with $t_{ij} = w_{ji} / \sum_{v_k \in \text{Out}(v_j)} w_{jk}$; $\text{Out}(v)$ denotes the set of vertices with an edge from v . \mathbf{B}^T is a variant of the transposed adjacency matrix $\mathbf{B} = (b_{ij})$ with $b_{ij} = w_{ij} / \max_{i,k} (w_{ik})$. For all definitions, $1 \leq i, j, k \leq n$, with $n = |V|$.

Algorithm 1 Basic algorithm of power method

```

1:  $t=0$ ;
2: repeat
3:    $t=t+1$ ;
4:    $\vec{v}_t = \mathbf{M} \cdot \vec{v}_{(t-1)}$ ;
5:    $\vec{v}_t = f(\vec{v}_t, L)$ ;
6:    $\delta = \|\vec{v}_t - \vec{v}_{(t-1)}\|$ ;
7: until  $\delta < \epsilon$ 

```

2.3.2 Classes of Multiple Computations

We see three main situations where the computation of several centrality measures within one power iteration or other, similar optimizations are likely to be useful, or other optimizations might be possible: (1) *several measures*, i.e., computation of several measures on the same data set, (2) *multiple data sets*, i.e., the computation of one measure with identical parameter settings on different data sets, and (3) *multiple parameter settings*, i.e., the computation of one measure with different parameter settings on the same data set. Combinations of these cases can of course happen. We now discuss each case separately.

Several measures. We observe that behavioral trust policies which require the evaluation of several centrality measures are natural (cf. Policy P_2 in Example 1), or that a system frequently has to evaluate several trust policies containing different centrality measures at the same time (cf. Sect. 2.1).

Multiple data sets. Different data sets induce different graph structures, like two behavioral trust policies referring to the same centrality measure, but using different feedback (see P_1 and P_2 in Example 1). Two directed, weighted graphs $G(V, E)$ and $G'(V', E')$ can differ in several ways. The sets of vertices V/V' and the sets of edges E/E' can either be equal, be distinct, intersect or have a subset relationship. Note that there are dependencies between relationships, e.g., two distinct set of vertices imply, in general, distinct sets of edges. Finally, the weights of the edges may differ for otherwise identical graphs. While V determines the size of Matrix M , E and the weights determine its elements.

Multiple parameter settings. Eigenvector-based centrality computation using the power method depends on parameters, and different parameter settings yield

Table 2.1 Classes of multiple centrality computations

Type of overlap	M	f	L	ϵ
Several measures	✓	✓	✓	
Multiple data sets	✓			
Multiple parameter settings (MPS _{CM})			✓	
Multiple parameter settings (MPS _{ϵ})				✓

different results. There are two kinds of parameters: (1) Inherent arguments determining the semantics of a centrality measure, e.g., the damping factor d of PageRank. This is the measure-specific argument list L mentioned before. MPS_{CM} denotes this class of parameters. (2) Parameters controlling the centrality computation. In particular, the parameter ϵ (MPS _{ϵ}) of the power method specifies the stop condition. It does not depend on the centrality measure.

To sum up, Table 2.1 shows how the cases differ with regard to the parameters for the execution of a power method. A checkmark indicates where the parameters have to be different in order to fall into the respective category.

2.4 Optimization of Combined Centrality Computations

We now present *Loop Fusion* and *Re-use of Results* which optimize the combined computation of Eigenvector-based centrality measures.

2.4.1 Loop Fusion

With the number n of nodes typically being large, the resulting $n \times n$ Matrix M is large as well and may not fit into main memory. Thus, each matrix multiplication requires costly access to secondary storage, e.g., the hard disk. When computing several Eigenvector-based centrality measures, we therefore propose applying loop-fusion techniques to reduce the number of accesses to secondary storage. Loop fusion is a concept from compiler optimization to replace several loops with one. It aims to improve the performance by exploiting commonalities of the individual loops. For instance, if k loops require the same data from secondary storage, the data is read once and is used for all k loops.

Fusing two loops requires them to have the same number of iterations [3]. In general, two power-method executions do not satisfy this requirement. Since the stop condition of an execution is evaluated at runtime, the number of iterations cannot be computed a priori. Thus, the execution with the highest iteration count would result in further (unnecessary) iterations for all other executions. [3] overcomes this limitation via *guarding iterations*, i.e., the insertion of conditional statements to ensure that no unnecessary work is done in iterations. In this way

we extend the basic power-method algorithm for the parallel computation of two or more centrality measures; see Algorithm 2. We derive the additional conditions from the stop condition for each execution of the power method (Line 5). The algorithm stops if the stop conditions of all executions are fulfilled. Variable `finished` is `false` as long as at least one execution has not converged.

Algorithm 2 Extended power method with loop fusion

```

1: finished=false; t=0;
2: while (finished≠true) do
3:   finished=true; t=t+1;
4:   for all single computations  $c$  do
5:     if ( $\delta^{(c)} > \epsilon^{(c)}$ ) then
6:       finished=false;
7:        $\vec{v}_t^{(c)} = M^{(c)} \cdot \vec{v}_{(t-1)}^{(c)}$ ;
8:        $\vec{v}_t^{(c)} = f^{(c)}(\vec{v}_t^{(c)}, L^{(c)})$ ;
9:        $\delta^{(c)} = \|\vec{v}_t^{(c)} - \vec{v}_{(t-1)}^{(c)}\|$ ;
10:    end if
11:  end for
12: end while
  
```

The extended algorithm requires more computation steps than the basic version, cf. Algorithm 1, due to the additional conditions and the modification of `finished`. Another reason is the extended data structures required to distinguish between the parameters of the different executions. Thus, if both algorithms execute one power iteration, we expect the extended algorithm to be slightly slower than the basic one. With different classes of overlap, other parameters may be different (cf. Table 2.1). We now discuss the challenges and improvements expected from loop fusion for each class.

Several measures. Different measures result in different matrices M . However, their processing is identical: In one iteration, any execution requires access to the entries at the same positions of the matrices. This congruence suggests having one integrated data structure, the so-called *multi matrix*. Each matrix entry consists of several values, one from each underlying matrix.

Examples 3. In the following, two $n \times n$ matrices ($n = 3$) are transformed into one multi matrix:

$$\begin{pmatrix} 0 & 5 & 1 \\ 0 & 0 & 2 \\ 8 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & 3 \\ 4 & 1 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 5|0 & 1|2 \\ 0|1 & 0 & 2|3 \\ 8|4 & 1|1 & 0 \end{pmatrix}$$

□

On the logical level, ‘single’ matrices and multi matrices do not show any difference, so the total number of entries does not change. Thus, regarding the entries, the

number of accesses to secondary storage required is the same in both cases. On the physical level however, the data structure representing a matrix also contains meta-data, e.g., data describing the rows and the columns. Hence, the size of a multi matrix is less than the sum of the sizes of the single matrices. The effective improvement depends on the ratio between the size of the multi matrix and the overall size of all single matrices. This ratio in turn depends on the internal representation of the graph. See Sect. 2.7.

Multiple data sets. Different data sets result in different matrices for the power method. While the multi-matrix concept is applicable in general, it is less obvious how to construct the multi matrix. First, if two graphs have a different number of vertices, the corresponding matrices are of different size. In this case, the larger matrix determines the size of the multi matrix. Further, there typically is more than one matrix representing a graph; it depends on the assignment of the vertices to the rows/columns of the matrix.

Examples 4. The following two matrices are possible adjacency matrices for the same graph.

$$\begin{array}{c} \mathbf{A} \quad \mathbf{B} \quad \mathbf{C} \quad \mathbf{D} \\ \mathbf{A} \quad \left(\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array} \right) \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \end{array}$$

$$\begin{array}{c} \mathbf{B} \quad \mathbf{C} \quad \mathbf{D} \quad \mathbf{A} \\ \mathbf{B} \quad \left(\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{array} \right) \\ \mathbf{C} \\ \mathbf{D} \\ \mathbf{A} \end{array}$$

□

Since there are many representations of a graph, there also are various multi matrices. The optimal multi matrix maximizes the ratio of its size and the added size of all corresponding single matrices. In other words, a multi matrix is optimal if the number of non-zero entries in the multi matrix is minimal. From an algorithmic perspective, the problem of computing the optimal multi matrix for a set of single matrices is related to the problem of finding an isomorphism between (sub-)graphs.

While the (sub-)graph isomorphism problem concerns the question whether a (sub-)graph isomorphism between graphs exists, finding an optimal multi matrix concerns to the task of finding the largest sub-graph isomorphism. Since the complexity of finding graph isomorphisms is still unknown – finding subgraph isomorphisms is NP-hard – and beyond the scope of this work. We therefore consider in our theoretical analysis and evaluation only the cases where the sets of vertices V_1 and V_2 in two graphs are either identical or form subset relationship. To be more precise, to construct a multi matrix, we map each vertex $v \in V_1 \cap V_2$ to itself. Note that his restriction does not guarantee an optimal multi matrix, since mapping a vertex to another one might result in a better overlap of two graphs. It allows, however, for comparable results when evaluating the effect of the remaining cases two graphs can differ (see Sect. 2.3.2).

Multiple parameter settings. If only the parameter settings of different centrality computations differ, all computations make use of the same Matrix M , i.e., the same accesses to matrix entries when executing the combined power method. The implementation is straightforward, and we expect the highest improvement, compared to the other classes.

2.4.2 Re-Use of Computation Results

With the power method, the actual choice of the start vector affects the number of iterations required. We expect a start vector that is similar to the result vector to require fewer iterations to converge. Here, we quantify the similarity of two vectors as the norm of their difference, cf. the stop condition in Algorithm 1. With one centrality computation, no a priori knowledge is available to identify a ‘good’ start vector. Thus, single centrality computations use ‘naive’ start vectors, e.g., vectors where all elements have the same value. With multiple centrality computations, the idea now is that the result vector of one computation can serve as the start vector of another one. The rationale is that both result vectors might be similar so that the second computation requires fewer iterations. ‘Bad’ start vectors in turn, i.e., very different from the result, yield a slower rate of convergence than naive ones. To assess a priori whether two results are similar, we have to look at the kinds of overlap. For each class of multiple centrality computation we now discuss the difficulties and expected improvement.

Similarity of measures. The potential of the re-use of results seems to be limited. For two measures to yield similar result vectors on the same graph, (a) they have to assign a similar relative importance to the vertices, and (b) the absolute values of the centrality indices must be from a similar range. Speaking casually, *two measures are similar if they give similar importance to the various characteristics of graphs*. For example, PageRank and Positional Weakness are relatively similar, since they take the in-links of a vertex into account. In general, the absolute values of the centrality indices vary significantly between measures. To overcome this limitation, we normalize the centrality indices. However, normalization requires knowing the range of the result vector. This is the case for PageRank, where the centrality indices of the vertices add up to the number of vertices n . The result vector of Positional Weakness in turn does not have this characteristic, to give an example. To summarize, re-use is only meaningful here if both measures are similar, and if at least the second measure allows for normalization.

Similarity of data sets. Two data sets can be different because the vertices, the edges or the weights are different. These kinds of differences may affect the centrality indices to different extents: For instance, multiplying each weight with the same factor has no effect on the resulting PageRank values. In contrast, one additional edge, from a vertex with a high centrality index to one with a low

index, can alter the result significantly. Thus, a reliable prediction whether the re-use of results is beneficial would require an analysis of the underlying graphs. The overhead of such an analysis, even though it is only preliminary, is likely to be high; we therefore do not consider this variant here.

Similarity of parameter settings. In general, the same measure with different parameters yields different results. We expect that the more similar two parameter settings, the more similar are the results. However, the impact of the inherent parameters (MPS_{CM}) is hard to assess without experiments. For values of ϵ in turn (MPS_{ϵ}) we can estimate the improvement a priori. Consider two power-method executions p_1 and p_2 that only differ with regard to ϵ such that $\epsilon_1 > \epsilon_2$. Let i_1 be the number of iterations for p_1 and i_2 the one for p_2 , with $i_1 \leq i_2$. If the result of p_1 is used as start vector of p_2 , p_2 requires $i_2 - i_1$ iterations. Thus, experiments concerning the effect of various error thresholds are not necessary.

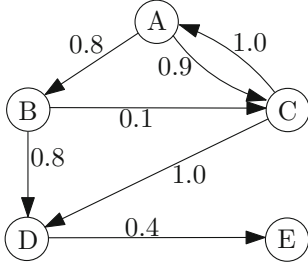
2.5 Implementation

Most of our optimizations for multiple centrality computation do not depend on the graph representation. This is particularly true for the re-use of results, since it aims for an optimization on a logical level. For loop fusion, the impact of the graph representation on performance depends on the type of overlap between the graphs (cf. Sect. 2.3). In case of multiple parameter settings (MPS_{CM} and MPS_{ϵ}), the combined centrality computation takes place on the same matrix for all single computations. Here the performance benefit is obvious, all single computations access the same matrix entries at the same time. However, for multiple measures or multiple datasets the graph representation matters, since it affects the ratio of the size of the resulting multi matrix and the corresponding single matrices. We now describe the graph representation we use for our theoretical analysis and evaluation.

Representation of graphs. There are various ways to store graphs, e.g., adjacency matrices/lists/tables, incidence matrices/lists/tables, and others, cf. [11]. The expected improvement depends on the sum of the sizes of the individual matrices divided by the size of the multi matrix. This ratio might differ with different graph representations. However, all representation techniques require meta-data, be it the rows/columns of a matrix or the links between list elements. A combined representation of several graphs saves storage space by using the same meta-data, and the more similar the graphs, the higher these savings.

In our evaluation we store all feedback data in a relational database, and represent graphs by means of an *incidence table* [4, 25]. An incidence table stores the information on the edges, i.e., each tuple stores the source (v_s), the target (v_t) and the weight (w) of an edge. Thus, an incidence table might not contain all vertices. This requires an additional *vertex table*.

Examples 5. The following figure (see Example 1) shows an example graph and its incidence table.



Incidence table

v_s	v_t	w
A	B	0.8
A	C	0.9
B	C	0.1
B	D	0.8
C	A	0.9
C	D	1.0
D	C	1.0
D	E	0.4

Vertex table

v
A
B
C
D
E

□

From the structure of an incidence table we can now derive the one of a multi incidence table. Since an incidence table represents the edges of a graph, storing the edges of several graphs in one table requires a Column w_i for each single graph. However, all combined graphs can share Columns v_s and v_t .

Examples 6. The following multi incidence table contains n graphs. The values of Column w_i represent the graph in Example 5.

Multi incidence table

v_s	v_t	w_1	...	w_i	...	w_n
A	B	0	...	0.8	...	0
A	C	0.5	...	0.9	...	0
A	D	0.9	...	0	...	0
B	C	0	...	0.1	...	0.8
B	D	0	...	0.8	...	0.3
...

Vertex table

v
A
B
C
D
E

Obviously, the vertex table remains unchanged.

□

In the multi incidence table ‘zero’ values for Attribute w_i are possible if there is an edge between two vertices in one graph, but not in another one.

Computing centrality measures. With a relational database as back-end, we have implemented Algorithm 1 of the original power method and Algorithm 2 of our extended power method using PL/SQL. We also have experimented with Java stored procedures. The results are very similar, so we will omit them in our evaluation.

2.6 Theoretical Analysis

The expected benefits of the re-use of results depend on the convergence behavior of the power method. The number of required iterations, in turn, depends on the actual input, i.e. the graph data. Qualifying the convergence behavior of the power method w.r.t the input, and therefore a theoretical analysis of the benefits due to the

re-use of results, is beyond the scope of this work. In contrast, loop fusion exploits the commonalities of two or more centrality computations on a physical level. The notion of the multi matrix as combined data structure (cf. Sect. 2.4.1) is particularly attractive in this context. The speed-up due to loop fusion depends on the ratio of the physical size of a multi matrix and the sum of the sizes of the data structures for each graph in isolation. The more compact the combined graph representation, the fewer accesses to secondary storage are necessary.

Depending on the graphs to be combined, however, loop fusion does not necessarily yield an improvement. In general, the more similar the graphs are, the more compact the combined representation. The question arises how to determine a-priori whether loop fusion should be applied or not. We now offer a respective analysis on graphs with varying degrees of similarity. While our examples and figures are limited to the incidence table, the analysis is general.

Examples 7. Below are fragments of a multi incidence table representing the sets of edges of n graphs and the corresponding incidence tables.

Multi incidence table

v_s	v_t	w_1	...	w_i	...	w_n
A	B	0	...	0.8	...	0
A	C	0.5	...	0.9	...	0
...

n single incidence tables

v_s	v_t	w_1		v_s	v_t	w_i
A	C	0.9	...	A	C	0.9
B	C	0.1		B	C	0.1
...

v_s	v_t	w_n
B	C	0.8
B	D	0.3
...

All single incidence tables share the Columns v_s and v_t , representing the start and target vertex of an edge. Thus, the multi incidence table has fewer columns than all single incidence tables together. However, the number of tuples varies depending on the similarity of the graphs. If an edge is not present in all graphs, the corresponding tuple in the multi incidence table contains ‘zero’ values. This increases the physical size of the table. \square

For any representation of a graph, we can distinguish between meta data and the actual data describing the graph. Meta data refers to the part of a graph representation that does not depend on the actual graph and can be shared among several graphs. Regarding incidence tables, for example, the meta data are the columns describing the source and the target vertices of edges. For an edge, s_m denotes the required physical size required to store the meta data, and s_w denotes the physical size required to store the weight. We assume that the values of s_m and

s_w are fixed for a certain graph representation. Further, let \mathcal{E} denote the set of edges $\{E_1, E_2, \dots, E_n\}$. Now we can express the ratio q of the size of a combined data structure and the n single data structures as follows:

$$q = \frac{|\bigcup_{i=1}^n E_i| \cdot (s_m + n \cdot s_w)}{\sum_{i=1}^n [|E_i| \cdot (s_m + s_w)]} = \frac{s_m + n \cdot s_w}{s_m + s_w} \cdot \frac{|\bigcup_{i=1}^n E_i|}{\sum_{i=1}^n |E_i|} \quad (2.1)$$

If $q < 1$, the multi incidence table is smaller than all corresponding single incidence tables. This makes a combined centrality computation using loop fusion on different data sets beneficial.

The factor $K(n) = \frac{s_m + n \cdot s_w}{s_m + s_w}$ in Eq. 2.1 describes the ratio of the required storage of an edge in a single and in a multi incidence table, with n being the number of combined data structures. For a given graph representation, i.e., fixed values for s_m and s_w , $K(n)$ only depends on the number of combined graphs, but not on the data, i.e., the appearance of the graph itself. Naturally, $K(1) = 1$, and $K(n)$ increases monotonously for larger n . Further, $K(n)$ decreases for an increasing ratio of $\frac{s_m}{s_w}$. This is intuitive, since the larger the storage required for the meta data, the more a combined representation can save.

Worst case and best case analysis. Factor $F(\mathcal{E}) = \frac{|\bigcup_{i=1}^n E_i|}{\sum_{i=1}^n |E_i|}$ in Eq. 2.1 depends on the actual graph data and reflects the similarities between a set of edges \mathcal{E} . Intuitively, the more similar the sets $E_i \in \mathcal{E}$ are, the smaller is $F(\mathcal{E})$. In the worst case, the sets of edges are completely disjoint. Note that this is a rather hypothetical case, since by re-labeling vertices, subsets of edges always can be mapped between all E_i , even in the absence of a (complete) graph isomorphism. In this case, each edge $e_i \in E_i$ results in a distinct edge in the combined data structure. This means that, in a multi incidence table, each tuple has only one weight column w_i with a non-zero entry. Thus, with no similarities between the sets of edges, $|\bigcup_{i=1}^n E_i| = \sum_{i=1}^n |E_i|$ holds. This yields a worst-case ratio of

$$q^{worst} = K(n) \cdot \frac{|\bigcup_{i=1}^n E_i|}{\sum_{i=1}^n |E_i|} = K(n) \cdot \frac{\sum_{i=1}^n |E_i|}{\sum_{i=1}^n |E_i|} = K(n) \quad (2.2)$$

In the best case, the sets of edges of all n graphs are identical, and therefore the n single incidence matrices are identical. As a result, no tuple in the corresponding multi incidence table has a ‘zero’ value for all weight columns w_i . Thus, $|\bigcup_{i=1}^n E_i| = |E|$, and $F(\mathcal{E}) = \frac{|E|}{|E| \sum_{i=1}^n 1} = \frac{1}{n}$ respectively. Now, in the best case, q is as follows:

$$q^{best} = \frac{K(n)}{n} = \frac{s_m + n s_w}{n(s_m + s_w)} = \frac{s_w}{s_m + s_w} + \frac{s_m}{n(s_m + s_w)} \quad (2.3)$$

The latter addend in the previous formula decreases monotonously with n . Thus, for $n \rightarrow \infty$, $q_{n \rightarrow \infty}^{best} = \frac{s_w}{s_m + s_w} \cdot q_{n \rightarrow \infty}^{best}$ defines the lower bound for ratio q , and it

depends on the given graph representation. Again, higher values of ratio $\frac{s_m}{s_w}$ are more beneficial since the absolute value for the lower bound $q_{n \rightarrow \infty}^{best}$ decreases.

Analysis of expected improvements. Besides the worst case, i.e., completely distinct sets of edges, and the best case, i.e., identical sets of edges, arbitrary relationships between the sets $E_i \in \mathcal{E}$ are conceivable. In the following we consider three different commonalities of sets of edges: (1) *Different edge weights*, i.e., only the weights of the edges differ among all E_i . (2) *Subset relationships*, i.e., all sets E_i form subset relationships. (3) *Intersecting sets of edges*, i.e., the sets E_i do not have subset relationships but have non-empty intersections. In general, these three commonalities appear in combination. However, we now discuss the expected performance benefits in the three cases separately.

Different edge weights. The sets of edges $E_i \in \mathcal{E}$ differ only in their weights. Varying the weights has no impact on the overall size of a (multi) incidence table, since it does not alter the number of tuples within the table. Thus, the resulting multi incidence matrix is always optimal, i.e., storing no ‘zero’ values. As a result, in case of different edge weights, the expected improvement equals the best case q^{best} , making loop fusion always beneficial.

Subset relationships. In this case, one set of edges $E^{sup} \in \mathcal{E}$ is a superset of all other sets. Note that we do not make any further restrictions. Since E^{sup} contains all edges, it follows that $|\bigcup_{i=1}^n E_i| = |E^{sup}|$, i.e., E^{sup} determines the number of tuples in the multi incidence table. For the sum of the sizes of each single set E_i , the lower and upper limit are $|E^{sup}| \leq \sum_{i=1}^n |E_i| \leq n \cdot |E^{sup}|$. Now, let $t^{sup} = |E^{sup}| / \sum_{i=1}^n |E_i|$, $0 < t^{sup} \leq 1$, be the relative coverage of all subsets $E_i \in \mathcal{E} \setminus E^{sup}$ by superset E^{sup} . With that, the expected improvement for subset relationships is $q = \frac{1}{t^{sup}(n-1)+1} \cdot K(n)$. If we set $q \stackrel{!}{=} 1$ to evaluate the minimum threshold of similarity, given a subset relationship between all E_i to determine when loop fusion is beneficial we can solve $1 = \frac{1}{t^{sup}(n-1)+1} \cdot K(n)$ for t^{sup} , and denote the solution as t_{min}^{sup} .

$$t_{min}^{sup} = \frac{K(n) - 1}{n - 1} = \frac{\frac{s_m + n s_w}{s_m + s_w} - 1}{n - 1} = \frac{s_w(n - 1)}{(s_m + 2s_w)(n - 1)} = \frac{s_w}{s_m + s_w} \quad (2.4)$$

If the relative coverage $t^{sup} > t_{min}^{sup}$, loop fusion yields a performance improvement. It is worth noting that for subset relationships the threshold depends only on the graph representation but not on the number of combined data sets. Figure 2.2 illustrate this for various values of n . For this example we set $s_m = 2s_w$; this is the situation of the attributes v_s, v_t and w_i in our incidence tables have the same data type (cf. Example 7). With these values $K(n) = \frac{n+2}{3}$. Given $K(n)$ we can calculate the worst and best case, e.g. for $n = 2$, $q^{worst} = \frac{4}{3}$ and $q^{best} = \frac{2}{3}$. Further, we can calculate threshold $t^{sup} = \frac{1}{3}$. Naturally, all these values show up in Fig. 2.2.

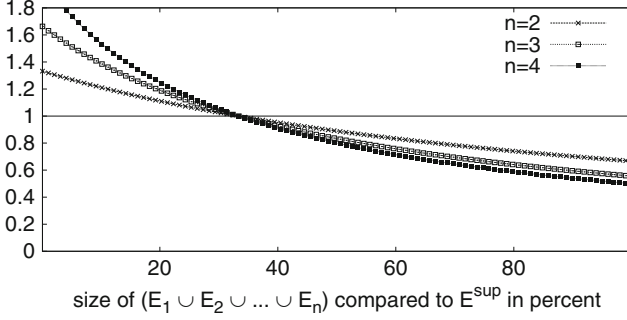


Fig. 2.2 Development of q for different subset relationships

Intersecting sets of edges. This case refers to sets of edges E_i that overlap to some degree, without explicitly forming a subset relationship. To ease the analysis, and to pronounce the distinction to the case subset relationships, we assume that all sets of edges are of an equal size l , i.e., $\forall E_i \in \mathcal{E} : |E_i| = l$. As a consequence, regarding Factor $F(\mathcal{E})$, $\sum_{i=1}^n |E_i| = n \cdot l$ is fixed. $|\bigcup_{i=1}^n E_i|$ depends on the size of the intersection, but is bounded by $l \leq |\bigcup_{i=1}^n E_i| \leq n \cdot l$. Compared to subset relationships, quantifying the relative overlap of intersecting sets of edges E_i is slightly more complex. Firstly, let $E_i^* = E_i \setminus \bigcup_{j=1, j \neq i}^n E_j$ be the subset of E_i that contains all edges which are only in E_i . We can then define the relative overlap t^\wedge of all sets E_i as $t^\wedge = \frac{n \cdot l - \sum_{i=1}^n |E_i^*|}{n \cdot l}$, where $0 \leq t^\wedge \leq 1$. With this, the expected improvement for intersecting sets of edges is $q = \frac{t^\wedge(n-1)+1}{n} \cdot K(n)$. Setting $q \stackrel{!}{=} 1$ and solving equation $1 = \frac{t^\wedge(n-1)+1}{n} \cdot K(n)$ for t^\wedge , and denoting the solution as t_{min}^\wedge , yields

$$t_{min}^\wedge = \frac{\frac{n}{K(n)} - 1}{n - 1} = \frac{\frac{n(s_m + s_w)}{s_m + s_w} - 1}{n - 1} = \frac{s_m(n - 1)}{(s_m + ns_w)(n - 1)} = \frac{s_w}{s_m + ns_w} \quad (2.5)$$

Again, if $t^\wedge > t_{min}^\wedge$, the application of loop fusion is beneficial. Compared to t_{min}^{sup} , threshold t_{min}^\wedge does not only depend on the graph representation but also on the number n of combined data sets. The more data sets one intends to combine the higher has to be their similarity (here: their intersection) between them to gain a speed-up due to loop fusion. Figure 2.3 shows the development of q for various values of n ; again we set $s_m = 2s_w$ yielding $K(n) = \frac{n+2}{3}$. Since both q^{worst} and q^{best} to not depend on the actual data sets, their values are the same as for the subset relationships (cf. Fig. 2.2). However, threshold t^\wedge is no longer constant but decreases for larger values of n . For example, for $n = 2$, $t^\wedge = \frac{1}{2}$.

Our analysis provides some interesting insight into the performance of loop fusion. Considering the parameters s_m and s_w for a given graph representation as fixed values emphasizes the effect of storing unnecessary ‘zero’ values. In real

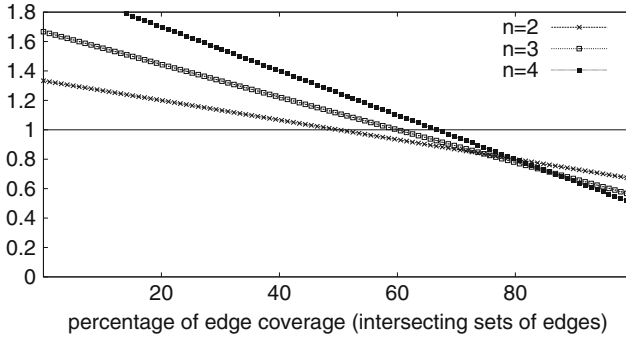


Fig. 2.3 Development of q for different intersection sizes

systems like relational databases, ‘zero’ values are stored more efficiently. This results in a better performance of loop fusion techniques.

2.7 Evaluation

We compare the isolated computation of centrality measures to the combined execution with loop fusion or re-use. The testbed for our experiments was a dual 2.2 GHz Opteron 64 bit platform, 2 GB main memory, SCSI Ultra 320 hard drive running Red Hat Enterprise Linux 3 and Oracle Database 10g Release 2. In contrast to the worst-case assumptions in our theoretical analysis, i.e., a fixed number of bytes required to store the edge in a (multi) incidence table, Oracle allocates storage according to the actual number of bytes required to store a data value. Thus, Oracle stores ‘zero’ values using a minimum number of bytes, compared to non-zero values. We therefore expect much better results in the worst-case (no or hardly any similarities between the combined graphs) compared to our theoretical analysis for loop fusion.

For the experiments we use both synthetic and real-world data. The real-world data is from two platforms: ADVOGATO (www.advogato.org) is a platform for a community of software developers. Participants can certify others on four different trust levels: *Observer* (0.0), *Apprentice* (0.6), *Journeyer* (0.8), *Master* (1.0). The data set¹ contains 7,383 participants and 51,797 trust relationships. EPINIONS (www.epinions.com) is a consumer-review site. A participant can add others to his “Web of Trust” whose reviews are likely to be of value. The data set² consists of 49,290 participants and 487,182 trust relationships. Regarding the synthetic data, we have generated it with a *feedback generator* that simulates cooperation among

¹www.advogato.org/person/graph.dot

²www.trustlet.org/datasets/downloaded_epinions/

individuals [27]. This lets us study the influence of specific characteristics of the data set in a much more controlled way. We have conducted experiments in graphs of various sizes, up to 100,000 vertices. Our ‘default’ data set consists of three graphs, each with 1,000 vertices, but with different numbers of edges. They have the densities 0.05/0.15/0.25, where the density D of a directed graph $G(V, E)$ is $D(G) = |E|/|V|^2$. The results for graphs with a low density are particularly relevant, since graphs derived from online scenarios tend to be sparse.

In this work we are interested in the speed-up of our optimization techniques. In addition, focusing on the absolute runtimes would be less helpful and might even be misleading. This is because (a) our datasets have different sizes and therefore give way to different absolute runtimes and (b) different datasets as well as modifications of the datasets throughout the evaluation – e.g., the removal of edges to generate subset relationships – result in different numbers of required iterations of the power method. To ease comparability of our results, if not stated otherwise, we normalize the runtimes so that the one of an isolated computation is 1. This also allows to limit ourselves to presenting only the results for the synthetic data. This is because the relative improvement due to our optimization techniques has always been the same for both the synthetic and the real-world data, as we will demonstrate.

2.7.1 Loop Fusion Techniques

To quantify the benefit of loop fusion, we compare the runtime to the total duration of the isolated executions.

Size of graphs. We first investigate whether the size of a graph influences the effectiveness of loop fusion in combination with a multi matrix. This experiment tells us how much attention we should put to the graph size in subsequent experiments. We perform two single PageRank computations and a combined computation, each on both synthetic and on real-world data. The graphs generated are of various sizes, from 10,000 to 100,000 vertices. For the real-world data, we start with the complete data set and gradually reduce the number of participants/vertices together with the adjacent edges. To fill the multi matrix, which contains two or more different graphs, we pretend that there are two graphs (even though there actually is only one), i.e., each matrix cell contains the same value twice. The advantage of using the same graph twice is that both an individual and a combined computation require the same number of iterations to converge. If the number of iterations was very different, the centrality computation with more iterations would dominate the combined one. This in turn would make seeing the effective improvement more complicated.

Figure 2.4a shows the runtimes of the power method for the synthetic data. (The results for the real-world data are very similar and are omitted here.) Naturally, the runtimes increase with the number of vertices. Figure 2.4b shows that there is no significant difference between the sizes regarding the relative improvement due to loop fusion. The other centrality measures yield the same behavior. The similarities

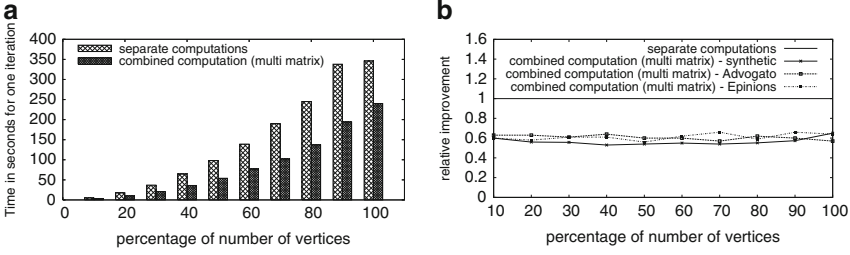


Fig. 2.4 Loop fusion: two PageRank computations in graphs of various sizes. **(a)** Absolute runtimes for the synthetic dataset. **(b)** Relative improvements for the synthetic dataset and the real-world datasets from ADVOGATO and EPINIONS

between the results for the various datasets appear in all of our experiments. We therefore only present the results for the synthetic data from now on.

Several measures. In our evaluation we explicitly consider three Eigenvector-based centrality measures, PageRank (PR), Positional Weakness (PW) and HITS. Table 2.2 shows the relative improvement using loop fusion. The values are normalized with 1. This corresponds to the runtime of the isolated computation of the two measures. Loop fusion yields an improvement, and the density of a graph has no significant impact. The differences between the combinations depend on the runtimes of the computations of the individual measures. In general, different measures on the same data set require a different number of iterations to converge. If these numbers differ significantly, the measure with the larger number dominates the runtime of the combined computation. The results are very similar for other data sets.

Multiple data sets. Two directed weighted graphs have different vertices, edges, or edge weights. We first consider the edges. Two sets E and E' can either be equal, be distinct, intersect or have a subset relationship. We carry out two experiments: (1) We start with two identical graphs and successively modify the edges to make them intersect at various degrees (from equal to distinct). The number of edges stays the same in both sets. (2) For the subset relationship we again start with two identical graphs and then delete the edges of one graph one by one. Figure 2.5a shows the results for PageRank and intersecting set of edges, Fig. 2.5b for subset relationships between sets of edges. The results for Positional Weakness and HITS are similar. The more similar both sets of edges, the greater is the improvement. Compared to the results of our theoretical analysis, the most noticeable difference is the much better performance in the worst case (no similarities between graphs.) The reason for this is the efficient handling of ‘zero’ values; the ‘zero’ values in the multi incidence case affect the physical size only slightly. However, the improvement depending on the degree of similarity – for both subset relationships and intersecting sets of edges – is very similar in theory and in the experiments.

Table 2.2 Relative improvement using loop fusion for a combined computation of two different centrality measures

	PR + PW	PR + HITS	PW + HITS
$D = 0.05$	0.66	0.8	0.73
$D = 0.15$	0.58	0.81	0.76
$D = 0.25$	0.58	0.83	0.79

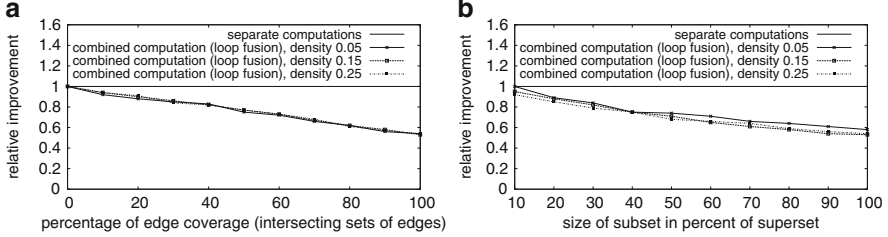


Fig. 2.5 Loop fusion: two PageRank computation on different sets of edges. (a) Intersecting sets. (b) Subset relationships

As mentioned, we consider only graphs whose sets of vertices are disjoint or have a subset relationship. To obtain sets of vertices with different degrees of overlap, we start with two identical graphs and remove vertices step by step. Figure 2.6 shows the result for PageRank. The other measures yield very similar results. Again, the improvement depends on the similarity of the sets of vertices. When the subset becomes smaller, the computation for the larger graphs dominates the combined computation more and more. Still, results for the worst case are better than expected, compared to theory. (Note that subset relationships between sets of vertices generally imply subset relationships between the corresponding sets of edges.) This has two reasons. Firstly, again, the negative effect of ‘zero’ values in the multi incidence table is much less pronounced in the implementation. Secondly, if the subset becomes very small, loop fusion does not have a noticeable impact any more. When a single computation completely dominates the combined one, the overhead of the extended power method starts to affect the runtime, decreasing the performance.

Finally, we investigate the impact of the weights of the edges. To do so, we run an experiment where we randomly modify the weights of a random subset of edges. On both the source and the modified graphs we compute the relative speedup with loop fusion, compared to sequences of isolated computations. In all runs – independently from the size of the subset or from the densities – loop fusion reduces the runtime by Factor 0.55–0.6. This is expected and in line with our theoretical findings: Modifying the edges does not change the size of the multi incidence table nor the number of its entries. So the costs of an iteration should not change.

Multiple parameter settings. We investigate both inherent parameters of centrality measures (MPS_{CM}) and the error threshold ϵ of the power method (MPS_{ϵ}) in one series of experiments. We use PageRank for two reasons. Firstly, only PageRank has an inherent parameter (damping factor d) and, secondly, the results concerning ϵ are very similar among all measures. See Fig. 2.7. The graph density has no

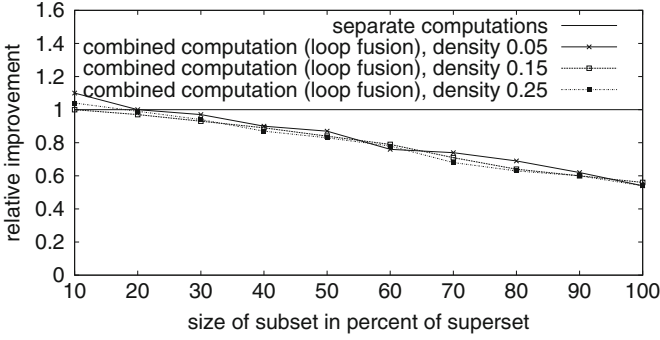


Fig. 2.6 Loop fusion: PageRank computation; different sets of vertices

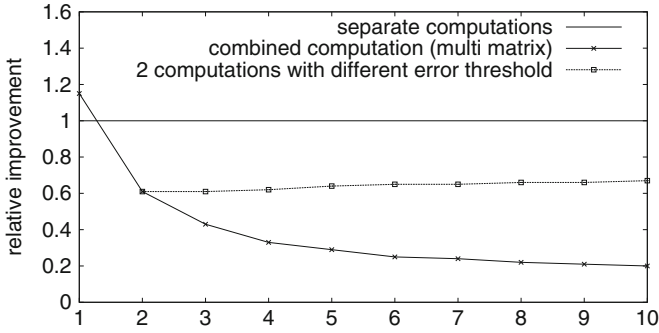


Fig. 2.7 Loop fusion: PageRank computations with different parameter settings

significant effect, and we omit the plots. The dashed line shows the improvement when executing various numbers of PageRank computations with different damping factors using loop fusion. As expected, a single computation using the extended power method is slightly slower, due to the additional computation steps in the extended algorithm. But the more centrality computations are done at the same time, the more loop fusion can reduce the runtime. The dotted line stands for two PageRank computations where we make the error threshold ϵ of the second computation smaller step by step. Starting with $\epsilon = 10^{-4}$ for both computations, we decrease ϵ for the second computation down to 10^{-10} . As expected, the improvement due to loop fusion becomes less. The computation with the decreasing threshold value requires more and more iterations and dominates the combined computation more and more.

The sum up, loop fusion is suited to improve the performance of the computation of several Eigenvector-based centrality measures. Due to the efficient handling of ‘zero’ values in particular, the improvement is significant even with different data

sets. Only if the costs of one computation stand out by much, the overhead of the extended power-method algorithm can lead to slightly weaker results.

2.7.2 Re-Use of Results

In each experiment we consider two subsequent centrality computations, comparing the isolated computation to the combined one with re-use of results. The improvement is the number of iterations saved with the second computation. We normalize the number of iterations, such that the number of iterations for the second centrality-computation using an isolated computation is 1.

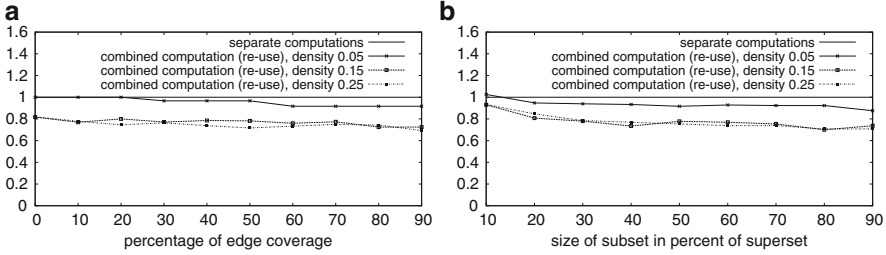
Several measures. In Section 2.4.2 we have argued that the re-use of results is promising only if two measures are similar, and the range of the scores of the second measure is known. Among the centrality measures considered, the latter condition is only true for PageRank. We carry out an experiment where we re-use the centrality computation result obtained with other measures (Positional Weakness, HITS) as start vector for the PageRank computation on the same data set. HITS provides two centrality indices, the authority index ($\text{HITS}_{\text{Authorities}}$) and the hub index ($\text{HITS}_{\text{Hubs}}$). PageRank, Positional Weakness and $\text{HITS}_{\text{Authorities}}$ are similar measures, they all consider the in-links of a vertex. For better comparability, we also re-use the result of $\text{HITS}_{\text{Hubs}}$ which considers the out-links of the vertices. As data sets, we use our three graphs with different densities. Table 2.3 shows the results. In all cases, the re-use of the result of PositionalWeakness and $\text{HITS}_{\text{Authorities}}$ for a PageRank computation yields no or only a small improvement, although these measures are similar. A rather unexpected result is that $\text{HITS}_{\text{Hubs}}$, which is not similar to PageRank, does not necessarily lead to additional iterations for the PageRank computation. Our explanation is that the rate of convergence of a power method is logarithmic, being high in the first iterations. In other words, the result of the first centrality computation must already be very close to the one of the second computation to see an improvement. At least, re-use of results does not increase the number of iterations among similar measures.

Multiple data sets. To evaluate the effect of re-use of results on the computation of one centrality measure on two different data sets, we proceed analogously to Sect. 2.7.1: For two graphs $G(V, E)$ and $G'(V', E')$ we consider (a) different degrees of the overlapping and different subset relationships of E and E' , (b) different subset relationships of V and V' , and (c) different weights of the edges. Again, we only show the results for PageRank.

For two overlapping sets of edges E and E' we use the result of the PageRank computation on E as the start vector for the PageRank computation on E' . To study the case of a subset relationship of two sets of edges ($E' \subseteq E$), we compute PageRank on the subset using the result of the PageRank computation on the superset as the start vector. Figure 2.8a shows the results for overlapping sets of edges, Fig. 2.8b

Table 2.3 Relative improvement of the combined computation of different centrality measures using the re-use of results

	Positional Weakness	HITS _{Authorities}	HITS _{Hubs}
$D = 0.05$	1	1	1
$D = 0.15$	1	1	1.06
$D = 0.25$	0.92	0.92	1

**Fig. 2.8** Re-use of results: two PageRank computation on different sets of edges. (a) Intersecting sets. (b) Subset relationships

for a subset relationship between the sets of edges. The more similar the set of edges, the greater is the improvement. Another finding is that for sparse graphs the improvement is rather small, even if the sets of edges are very similar.

Analogously, we consider the subset relationship of two sets of vertices ($V' \subseteq V$). We first compute PageRank on V and use its result as the start vector for the second computation on V' . Note that, here, re-use requires the normalization of the start vector. Figure 2.9 shows the result. For dense graphs the improvement is about 0.8–0.85, independently from the size of the subset. For sparse graphs however, re-use can worsen the runtime of the second centrality computation, and for larger subsets the improvement is less than for dense graphs.

Regarding the third issue, we randomly change the edge weights. We first compute PageRank on the original graph and use the resulting vector for the PageRank computation on the modified graph. Figure 2.10 shows the result. Again, the two key findings are as follows: (1) The more similar the weights are, the greater the improvement, and (2) the sparser the graph, the less the improvement.

Multiple parameter settings. Since we can assess the improvement with re-use for two centrality computations that differ only with regard to ϵ (MPS_ϵ) a priori, we focus on the inherent parameters of centrality measures (MPS_{CM}). Among the measures considered here, only PageRank features such a parameter, damping factor d . To quantify its effect, we conduct several runs where we perform two computations with different damping factors, using the result of the first computation as start vector for the second one. Figure 2.11 shows the result for a single run. Here, $d = 0.85$ for the first computation and $d \in [0.75 \dots 0.95]$ for

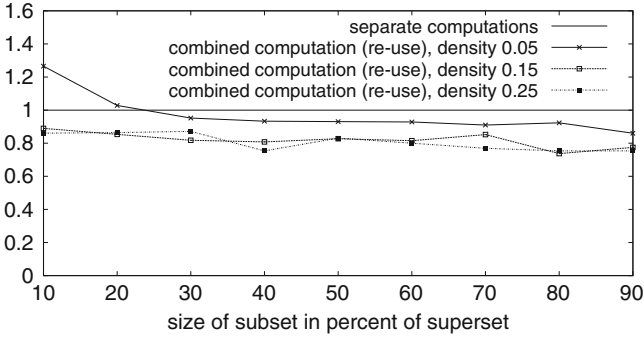


Fig. 2.9 Re-use of results: Two PageRank computation on two graphs with different sets of vertices (subset relationship)

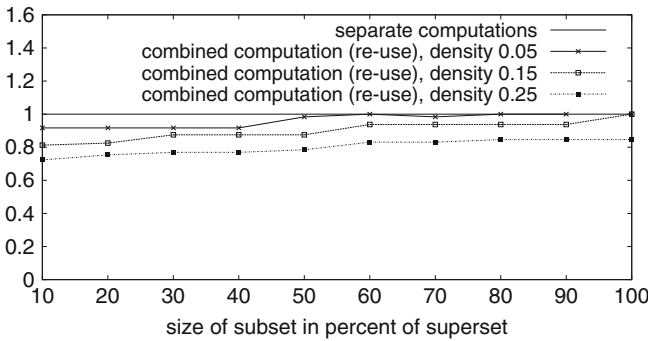


Fig. 2.10 Re-use of results: Two PageRank computations on two graphs with different edge weights

the second computation. The results for other runs with various combinations of d are very similar.

There only is a significant improvement if the damping factor in both computations is very close. For larger differences between the damping factors of two PageRank computations, the improvement for the second computation vanishes. However, re-use never requires more iterations than using a generic start vector.

To sum up, re-use only achieves a significant speedup if the results of two centrality computations are very similar. The evaluation shows that this is typically not true for several measures (SM). For the other classes MDS and MPS the improvement depends on the density of the graph. For sparse graphs in particular – note that the graphs of online scenarios tend to be sparse – re-use often cannot reduce the number of iterations of subsequent centrality computations, but increases it in some cases.

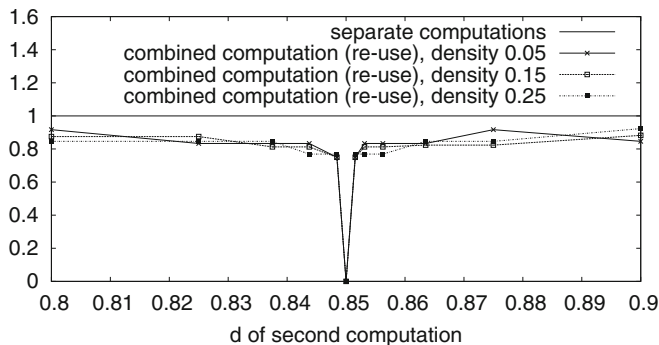


Fig. 2.11 Re-use of results: Two PageRank computations with various damping factors d

2.8 Conclusions

An accepted way to identify of trustworthy participants in online communities relies on Eigenvector-based centrality measures. However, centrality computation is expensive. While existing work has focused on the optimization of one centrality computation, little attention has gone into the computation of several centrality measures in combination. We have proposed and evaluated two optimization schemes, namely loop fusion and re-use of computation results. Re-use yields an improvement in distinct cases only. Loop fusion improves the performance in almost all situations, except when the cost of one computation is much higher than the costs of the other computations. Additionally, re-use is bound to the power method and therefore to Eigenvector-based centrality measures. The improvements due to loop fusion, result from the compact representation of the multi matrix. Thus, one can apply loop fusion to various graph algorithms at least in principle. Evaluating the actual benefit however requires further investigation.

References

1. Abbassi, Z., Mirrokni, V.S.: A recommender system based on local random walks and spectral methods. In: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, WebKDD/SNA-KDD '07. ACM, New York (2007)
2. Bacon, D.F., Graham, S.L., Sharp, O.J.: Compiler transformations for high-performance computing. *ACM Comput. Surv.* **26** (1994)
3. Blainey, B., Barton, C., Amaral, J.: Removing impediments to loop fusion through code transformations. In: Languages and Compilers for Parallel Computing, Springer, Berlin/Heidelberg (2002)
4. Bowen, B., Kocura, P.: Implementing conceptual graphs in a RDBMS. In: Proceedings on Conceptual Graphs for Knowledge Representation. Springer, London (1993)

5. Cho, J., Schonfeld, U.: RankMass crawler: a crawler with high personalized PageRank coverage guarantee. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07. VLDB Endowment (2007)
6. Cummings, J.N.: NetVis module: Dynamic visualization of social networks. Available online at <http://www.netvis.org> (2001–2012)
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008)
8. Delaviz, R., Andrade, N., Pouwelse, J.: Improving accuracy and coverage in an internet-deployed reputation mechanism. In: Peer-to-Peer Computing. IEEE, New York (2010)
9. Getoor, L., Diehl, C.P.: Link mining: a survey. *SIGKDD Explor. Newsl.* **7**, 3–12 (2005)
10. Golub, G.H., van Loan, C.F.: Matrix Computations (Johns Hopkins Studies in Mathematical Sciences), 3rd edn. Johns Hopkins University Press, Baltimore (1996)
11. Gross, J., Yellen, J.: Graph Theory and its Applications. CRC, Boca Raton (1999)
12. Gyöngyi, Z., Garcia-Molina, H., Pedersen, J.: Combating web spam with TrustRank. In: Proceedings of the 13th International Conference on Very Large Data Bases, VLDB '04. VLDB Endowment (2004)
13. Hütter, C., Hartmann, B.O., Böhm, K., Heistermann, T., Kohlmeyer, K.S., Reckling, R., Reiche, M., Parra, D.S.: SONAR: towards user-centric social network analysis and visualization. In: Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT '10. IEEE Computer Society, New York (2010)
14. Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Extrapolation methods for accelerating PageRank computations. In: Proceedings of the 12th International Conference on World Wide Web, WWW '03. ACM, New York (2003)
15. Kamvar, S.D., et al.: The eigen trust algorithm for reputation management in P2P networks. In: Proceedings of the 12th International Conference on World Wide Web, WWW '03. ACM, New York (2003)
16. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. *J. ACM* **46**, 604–632 (1999)
17. Lee, C.P., Golub, G.H., Zenios, S.A.: A fast two-stage algorithm for computing PageRank and its extensions. Tech. rep., Stanford University (2004)
18. Ng, A.Y., Zheng, A.X., Jordan, M.I.: Link analysis, eigenvectors and stability. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Francisco (2001)
19. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. Tech. rep., Stanford (1998)
20. Parreira, J.X., et al.: Efficient and decentralized PageRank approximation in a P2P web search network. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB '06. VLDB Endowment (2006)
21. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems: facilitating trust in internet interactions. *Commun. ACM* **43**, 45–48 (2000). ACM
22. Richardson, M., Agrawal, R., Domingos, P.: Trust management for the semantic web. In: Proceedings of the International Semantic Web Conference, ISWC'03. Springer, Berlin/Heidelberg (2003)
23. Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: Proceedings of the 1st International Joint Conference on Autonomous Agent and Multi-Agent Systems, AAMAS '02. ACM, New York (2002)
24. Sierra, C., Debenham, J.: Information-based reputation. In: 1st International Conference on Reputation Theory and Technology, ICORE '09. New York (2009)
25. Stephens, S., Rung, J., Lopez, X.: Graph data representation in oracle database 10g: case studies in life sciences. *IEEE Data Eng. Bull.* **27**, 61–66 (2004)
26. von der Weth, C., Böhm, K.: A unifying framework for behavior-based trust models. In: Proceedings of the International Conference on Cooperative Information Systems, CoopIS '06. Springer, Berlin/Heidelberg (2006)

27. von der Weth, C., Böhm, K.: Towards an objective assessment of centrality measures in reputation systems. In: Proceedings of the 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services, CEC/EEE '07. IEEE Computer Society, New York (2007)
28. von der Weth, C., Böhm, K., Hütter, C.: Optimizing multiple centrality computations for reputation systems. In: Proceedings of the International Conference on Advances in Social Networks Analysis and Mining, ASONAM '10. IEEE Computer Society, New York (2010)
29. Wang, Y., DeWitt, D.: Computing PageRank in a distributed internet search system. In: Proceedings of the 30th International Conference on Very Large Data Bases, VLDB '04. VLDB Endowment (2004)
30. Wassermann, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge/New York (1994)
31. Wicks, J., Greenwald, A.: More efficient parallel computation of PageRank. In: Proceedings of the 30th International Conference on Research and Development in Information Retrieval, SIGIR '07. ACM, New York (2007)
32. Xiong, L., Liu, L.: PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. Knowl. Data Eng.* **16**(7), 843–857 (2004)
33. Yamamoto, A., Asahara, D., Itao, T., Tanaka, S., Suda, T.: Distributed PageRank: a distributed reputation model for open peer-to-peer networks. In: Proceedings of the 2004 Symposium on Applications and the Internet-Workshops, SAINT-W '04. IEEE Computer Society (2004)
34. Zhang, H., Goel, A., Govindan, R., Mason, K., Roy, B.V.: Improving eigenvector-based reputation systems against collusions. In: Workshop on Algorithms and Models for the Web Graph. Springer, New York (2004)
35. Zhang, L., Zhang, K., Li, C.: A topical PageRank based algorithm for recommender systems. In: Proceedings of the 31st International Conference on Research and Development in Information Retrieval, SIGIR '08. ACM, New York (2008)

The Influence of Technology on Social Network Analysis
and Mining

Özyer, T.; Rokne, J.; Wagner, G.; Reuser, A.H.P. (Eds.)

2013, XXIII, 643 p., Hardcover

ISBN: 978-3-7091-1345-5