Title
Fast determination of structurally cohesive subgroups in large networks

Journal
Journal of Computational Science, 17(Pt 1)

ISSN
1877-7503

Authors
Sinkovits, Robert S
Moody, James
Oztan, B Tolga
et al.

Publication Date
2016-11-01

DOI
10.1016/j.jocs.2016.10.005

Peer reviewed

HHS Public Access

Author manuscript

J Comput Sci. Author manuscript; available in PMC 2017 November 01.

Published in final edited form as:

J Comput Sci. 2016 November ; 17(Pt 1): 62–72. doi:10.1016/j.jocs.2016.10.005.

Fast determination of structurally cohesive subgroups in large networks

Robert S. Sinkovits[a,*], **James Moody**[b,c], **B. Tolga Oztan**[d], and **Douglas R. White**[d]

[a]San Diego Supercomputer Center, University of California, San Diego, United States

[b]Department of Sociology, Duke University, United States

[c]King Abdulaziz University, Saudi Arabia

[d]Department of Anthropology and Institute of Mathematical Behavioral Science, University of California, Irvine, United States

Abstract

Structurally cohesive subgroups are a powerful and mathematically rigorous way to characterize network robustness. Their strength lies in the ability to detect strong connections among vertices that not only have no neighbors in common, but that may be distantly separated in the graph. Unfortunately, identifying cohesive subgroups is a computationally intensive problem, which has limited empirical assessments of cohesion to relatively small graphs of at most a few thousand vertices. We describe here an approach that exploits the properties of cliques, *k*-cores and vertex separators to iteratively reduce the complexity of the graph to the point where standard algorithms can be used to complete the analysis. As a proof of principle, we apply our method to the cohesion analysis of a 29,462-vertex biconnected component extracted from a 128,151-vertex co-authorship data set.

Keywords

Graph algorithm; Social network; Cohesive subgroup; Vertex separator

1. Introduction

A graph $G = (V, E)$ is a set of vertices $V(G)$ together with a set of edges $E(G)$ that connect pairs of vertices. Furthermore, the graph is simple if it contains no loops (e.g. edges that connect a vertex to itself), each pair of vertices is joined by at most one edge and the edges are undirected. In all discussions that follow, we restrict ourselves to simple graphs and note that graphs containing loops or multiple edges (multigraphs) can be converted to simple graphs without loss of generality.

Before describing k-components, which are the focus of this work, we first introduce several other concepts that we will use repeatedly. A clique is a subset of vertices from a graph with the property that all of the members are directly connected. A maximal clique has the

*Corresponding author. sinkovit@sdsc.edu (R.S. Sinkovits).

additional property that it is not wholly contained within a larger clique. Throughout the remainder of this discussion, we use clique to refer to maximal cliques. A *k*-core of a graph is an induced subgraph with the property that all of its vertices have at least degree *k*.

A structurally *k*-cohesive subgroup, also known as a *k*-component, is defined as a subset of vertices within a graph with the property that they remain connected subject to the removal of any $k-1$ vertices. Equivalently, Menger's theorem implies that any pair of vertices within the *k*-cohesive subgroup is joined by at least *k* vertex disjoint paths [1,2], which have the property that they only have the start and end vertices in common. Whereas community detection merely separates subgroups into two based on their relative densities, *k*-cohesive groups form a nested structure. They are stacked successively from 1 to *k* with non-increasing size and may share vertices with one another, a far more orderly complex structure [3]. An undirected connected graph is trivially 1-cohesive. By definition, the vertices within a biconnected component, or bicomponent, form a 2-cohesive subgroup that is necessarily contained within a 1-cohesive component.

While *k*-cohesion is a more powerful and rigorous concept than other measures of robustness, its primary drawback is that identifying cohesive subgroups is computationally intensive, there by making it impractical to use for the analysis of larger graphs. For example, recent work in this area addressed the development of heuristics for finding *approximate* solutions to the identification of *k*-components that is an order of magnitude faster than algorithms available at the time [4]. An efficient implementation of this algorithm in turn depends on fast approximations for finding the number of vertex disjoint paths between vertex pairs – the shortest path is calculated between a pair of vertices and all vertices lying on this path are labeled as having been visited. A new shortest path excluding previously visited vertices is calculated and the process is repeated until there are no more paths involving only unvisited vertices [5]. Although this is certainly an important advance, the inexactness of the results can limit their usefulness. False positives, vertices that are mistakenly included in the *k*-component, can be identified and removed, albeit at additional computational expense. False negatives, on the other hand, are impossible to detect without being able to compare to the exact results.

Here we describe an *exact* solution to the *k*-components problem that is considerably faster than the algorithm described above. The efficiency of our method is due to the application of graph reduction techniques that do not alter the final solution. At those stages where vertex disjoint path calculations are required, only exact algorithms are used. As a result, we find that the application of our method to a graph that is thirteen times larger than the biggest problem tackled so far using the approximate method (128,151 vs.9767 vertices) could be completed in less than one-sixth of the time.

We developed an iterative approach to the problem that gradually reduces the size and complexity of the original graph. Our method builds on existing algorithms or concepts for finding cliques [6], articulation points, bicomponents [7] and *k*-cores [8,9],together with approximate and exhaustive searches for higher order vertex separators, sets of vertices whose removal results in a disjoint graph. Finally, we resort to the standard algorithms based on the solution to the maximum-flow problem [10] for identifying *k*-components only after

the problem has been reduced to a manageable size. The idea of searching for *k*-components within *k*-cores had been suggested by Seidman [11]. This was then formalized by White and Harary [12], relying on Whitney's theorem [13], to address the hierarchical nesting of *k*-components within *k*-edge-components, which are further contained within *k*-cores. We note that several of these steps, such as the reduction to bicomponents and *k*-cores, have been implemented previously to significantly improve the speed of the *k*-component search [4], but to the best of our knowledge the use of cliques and the implementation of new algorithms for finding either subsets or complete sets of vertex separators is novel.

The primary application of *k*-components is to social networks. These are generally composed of clusters of vertices that are more connected to each other than to the rest of the graph. The clustered nature of social networks has proved relevant for questions as widely varying as young adolescent delinquency [14], political polarization [15], and national economic performance [16], to name a few (for a general review, see: [17]). This structural het-erogeneity has sparked a rapidly growing literature on cohesive subgroups and modularity or community detection [18]. However, most techniques for identifying group structure turn on simple volume differences, rather than the *patterns* of relations. In the common view [19], a community is a collection of vertices that share some higher-than-expected fraction of ties within the group than between – the "modularity" of the network. But volume-based community detection techniques miss the key structural feature of resilience to disconnection by vertex removal. Since the sociological roots of a community are its supra-individual character – that a group can remain even in the face of individuals leaving – com-munity detection techniques focusing on that crucial aspect and its twin property of multiple intraconnecting paths should be more common. We have previously identified vertex intraconnectivity and the groups implied by the nesting of vertex-intraconnected sets as a key foundation for such groups [3,12]. We use *intra* to pre-fix connectivity as a reminder that the Menger theorem [2] proves that the limit of any maximal set of vertex pairs each with *k* or more independent paths between them also defines the boundaries of a *k*-cohesive group that cannot be separated by removal of fewer than *k* vertices.

In the following section, we describe the techniques that we use to first reduce the graph. We then discuss efficient approaches to finding both all 2- and 3-vertex separators and approximate techniques for finding a large fraction of the separators. We conclude with a description of the complete method, its application to a previously intractable problem and future improvements.

2. Graph reductions

Two inexpensive and easy-to-apply techniques can be used to identify and remove vertices that either cannot belong to a *k*-component or are members of cliques at the fringes of the graph. When combined with the algorithm for finding bicomponents and applied in an iterative manner to identify a kernel from which no more vertices can be deleted, they form a powerful tool for graph reduction.

2.1. Exploiting the properties of cliques

Real-world graphs generally have cliques of varying sizes and many of these will contain both vertices that have neighbors residing outside of the clique and vertices that only have neighbors within the clique. We refer to the former as “worldly” vertices and the latter as “isolated” vertices (Fig. 1).

Consider a clique of size n_t with n_w worldly vertices and n_i isolated vertices ($n_t = n_w + n_i$). If n_w is less than or equal to k, the isolated vertices cannot belong to a larger k-component containing vertices outside of the clique. The reason this is true is that all paths from the isolated vertices to vertices that lie outside of the clique must pass through the worldly vertices, thereby setting an upper limit on the number of vertex disjoint paths to n_w. In other words, the worldly vertices serve as a choke point or bottleneck for paths originating at the isolated vertices. We can remove these isolated vertices from the graph and record that the clique forms a $(n_t - 1)$-component of the graph.

It should be noted that the above procedure is simply identifying a subset of the $(k - 1)$-separators within the graph. Hence, their application in this way is consistent with established techniques for splitting the graph using complete sets of vertex separators. Even though it cannot be used to find all of the separators, it can bed one extremely quickly for sparse graphs due to the efficiency of the clique-finding algorithm and can substantially reduce the size and complexity of the graph before further processing. Tomita et al. [20] point out that in the worst case generating all maximal cliques scales as $O(3^{n/3})$, which can limit the applicability to dense graphs. We do not believe that this will be an issue for social networks, which tend to be sparse, but this should be kept in mind when extending to other types of graphs or networks. Finally, we note also that in general applying separators to split the original graph into smaller, higher-order components can introduce new separators, but this does not occur when using the cliques in this way.

2.2. Exploiting properties of *k*-cores

The requirements for a k-core are less stringent that those for a k-component; all k-components are k-cores, but the converse is not true. As a concrete example, consider a graph where all vertices have degree equal to at least three, meeting the requirement for a 3-core, but that contains a 2-vertex separator.

Computationally, k-cores can be identified very easily by iteratively removing all vertices of degree less than k until only k-cores remain. The bookkeeping associated with the k-coring process is even simpler than that used in the clique analysis described earlier since the deleted vertices do not have to be tracked as possibly belonging to a distinct k-component.

The use of k-cores alone does not solve the problem of finding the k-components, but it can result in substantially simpler graphs and make the remaining problem more tractable.

2.3. Combined graph reductions

The full power of these two techniques becomes apparent when used iteratively and in combination with an algorithm for finding bicomponents. Given a k-component, our immediate goal is to generate a reduced graph that still contains the embedded $(k + 1)$-

components. After identifying cliques with n_k worldly vertices and removing the isolated vertices, the resultant graph may contain new cliques with this property and multiple cycles of isolated vertex removal may be necessary (Fig. 2).

The graph obtained from the clique analysis may in turn not be a k-core and vertices of degree less than k can be removed. The process of k-coring can lead to the introduction of new articulation points, which can then be removed by decomposing the graph into its bicomponents (Fig. 3).

The smaller bicomponents can be processed separately and are usually modest enough in size that they can be analyzed using standard cohesion algorithms. The new dominant bicomponent may contain cliques with n_k worldly vertices and the process of isolated node removal, k-coring and decomposition into bicomponents can be repeated until no further progress is possible (Fig. 4).

It should be kept in mind that the new graph obtained from this iterative reduction procedure is only guaranteed to be a bicomponent, regardless of the degree of cohesion of the original graph, since new lower-order vertex separators may be introduced. This is not an issue when searching for tricomponents starting from an initial bicomponent since the reduced graph is also a bicomponent. When searching for higher-order components, we find in practice that it is relatively easy to recover the largest k-component during the search for the $(k + 1)$-components due to the drastically smaller sizes of the reduced graphs.

3. Efficient identification of vertex separators

Our approach to finding the k-components depends critically on the identification of vertex separators. Once a $(k-1)$-vertex separator has been found, it can be used to partition the graph into two or more candidate k-components. In this section we describe our method for finding the 2- and 3-vertex separators. Although our primary case study (see Section 5.1) did not require that we find 4-vertex separators in the largest 4-component, one of the techniques described here (Section 3.3.2) was modified and adapted to higher order separators in our other two case studies.

3.1. Restricting search for vertex separators

We described earlier how we could use cliques to identify and remove vertices that could not be members of the dominant k-component. This was essentially a shortcut for rapidly finding a subset of the vertex separators where the number of worldly vertices in a clique was equal to the size of the separator being sought.

We can extend this idea to exclude vertices from consideration in the search for vertex separators. The isolated vertices within a clique, regardless of the number of worldly vertices or size of the clique, cannot be members of a separator since the removal of all or any of these vertices does not change the number of connected components. In addition, these isolated vertices do not contribute to the number of vertex disjoint paths between any pairs of vertices outside of the clique. This becomes obvious when we consider that any path

through an isolated vertex would have to first pass through one of the worldly vertices (Fig. 5).

Before we begin the search for the vertex separators, we first construct a new graph in which all isolated vertices have been removed. In our case study, we find that this generally leads to a 50% reduction in the time required for these searches.

3.2. Identification of 2-vertex separators

We employ two methods to find the 2-vertex separators. The first is a fast approximate algorithm that identifies a large fraction of the separators, namely those in which the two members are joined by an edge. The pairs of endpoints are then tested to determine whether or not they form a separator. The more comprehensive search takes advantage of the highly efficient algorithms for finding the articulation points of a graph. Rather than explicitly testing all pairs of vertices to determine whether or not they form a 2-vertex separator, we remove each vertex in turn and locate the articulation points in the resultant graph (Fig. 6). While this could potentially be more expensive than testing all pairs of vertices since Tarjan's algorithm for articulation points has a worst case scaling of O(|V| + |E|) [21], we found in practice that this approach was much faster.

We are fully aware that other algorithms for finding 2-vertex separators and tricomponents exist that are more efficient than the approach described above [22–27]. We feel though that the ease of implementation, coupled with the fact that deploying this simple algorithm did not hamper our ability to rapidly solve a previously intractable problem (see Section 5), justified its use. In addition, the idea of dividing separators into those that are easy to find (members joined by edge) versus those that are hard to find (members not joined by edge) could be applied to higher order separators to drastically reduce the time to solution.

3.3. Identification of 3-vertex separators

Once the size of the graph, or more specifically the largest tricomponent, grows to be sufficiently large, a brute force search for the 3-vertex separators becomes impractical. To avoid this unfavorable $O(|V|^3)$ scaling behavior, we divide our search into two phases where we separately search for the easy- and hard-to-find separators (Fig. 7).

3.3.1. Identification of easy-to-find 3-vertex separators—In our case study, we noted that for the vast majority of the 3-vertex separators, at least two of the vertices within the separator are neighbors within the graph. To find these separators, we iterate over the edges of the graph, remove the endpoints of the edge and then identify the articulation points in the resultant graph (Fig. 8). In the worst case, this search will scale as O(|E| (|V| + |E|)).

3.3.2. Identification of hard-to-find 3-vertex separators—The remaining 3-vertex separators, for which none of the members are neighbors, are more difficult to identify. For this particular class of separators, we employ a different strategy (Fig. 9).

Consider a vertex v with N neighbors $\{n_1, n_2, \cdots, n_N\}$. For any pair of neighbors $\{n_i, n_j\}$, precisely one of the vertex disjoint paths between the pair will contain v. If the number of vertex disjoint paths between n_i and n_j is exactly three, then v must be a member of a 3-

vertex separator. This algorithm does not directly tell us the membership of the 3-vertex separators, but rather finds the set of all vertices that collectively belong to any 3-vertex separator. If the size of this set is small, then an exhaustive search over all combinations of three vertices from this list is trivial.

We can make the algorithm more efficient by noting that not all pairs of vertices neighboring the candidate vertex v need to be tested. For example, if two of the neighbors are themselves joined by an edge, then we only need to calculate the number of vertex disjoint paths between one member of the pair and the other neighbors (Fig. 10).

4. Finding *k*-components

In this section, we describe the iterative procedure that we use to identify the 3-, 4- and 5 components. Although other algorithms exist that are specifically designed for this task [22–25], our iterative approach is highly efficient for finding the 4- and 5- components and can serve as a template for identifying higher-order *k*-components.

4.1. Finding tricomponents

Beginning with the largest bicomponent, we reduce the graph using a slightly modified version of the process described earlier that includes just the reduction to *k*-cores and bicomponents. The clique processing is omitted since it provides little additional benefit, but at significant computational expense due to the large number of cliques that only contain three members (triangles). Once the graph is reduced to a kernel, we iteratively find and apply the easy-to-find 2-vertex separators, followed by *k*-coring and reduction to bicomponents. Finally, the process is repeated using a search for all 2-vertex separators (Fig. 11).

4.2. Finding higher order *k*-components

The search for the 4-components begins with the same steps that are used during the search for the tricomponents. Starting with the largest tricomponent that was found earlier, we first reduce the graph by repeated application of the clique processing ($n_W = 3$), *k*-coring ($k = 4$) and reduction to the largest bicomponent. Since the resulting graph at this point is not guaranteed to be a tricomponent, we still need to repeatedly identify and apply all 2-separators to obtain a new tricomponent that is generally much smaller than the original graph.

Our strategy at this point is similar to the search for the largest tricomponent in that we first take advantage of the easy-to-find 3-vertex separators (at least two members joined by an edge) before extending the search to include all separators. To further accelerate the search for 4-components, we again reduce to *k*-cores, largest bicomponent and largest tricomponent after applying the 3-vertex separators. (Fig. 12).

The search for higher order components ($k > 4$) uses the same general strategy of reducing the graph using clique processing ($n_W = k - 1$), *k*-coring and reduction to the largest bicomponent. The resultant graph is then processed further to reduce to a more tractable ($k - 1$)-component. In the primary test problem used in this study, an explicit search for 4-

separators was not necessary since the graph obtained by applying the earlier steps was sufficiently small (|V| = 33) that existing algorithms could be used.

5. Case studies

5.1. Sociology co-authorship data set

To test our new approach, we chose Moody's co-authorship dataset [28] for our primary case study. This collection spans 36-years of publications in sociology journals from 1963 to 1999 and encompasses 128,151 authors. In a one-mode network representation of the data, the vertices correspond to authors and the edges indicate joint authorship. The graph is unweighted (i.e. the edges are not weighted to reflect the number of papers co-authored) and undirected. This problem is challenging enough to be intractable using earlier methods. Given that the data was obtained from a naturally emerging social network, it is expected to be representative of at least one class of real problems. Finally, this particular data set was selected since the original analysis left a number of important questions unanswered. Graph reduction and cohesive subgroup algorithms are implemented in R, with calls to the igraph software package [29].

The 128,151 vertices belong to 20,182 disjoint clusters, with the distribution dominated by a single large connected component of 68,285 vertices. All other connected components were at least three orders of magnitude smaller, ranging in size from 2 to 48 vertices. Within the largest connected graph, we identified 25,823 biconnected components, including one of size 29,462 plus other biconnected components ranging in size from 2 to 36. This large biconnected component is the starting point for our analysis, as was the case for Moody.

5.1.1. Identifying tricomponents

We applied the procedure described in Section 4 to identify the major tricomponent within the bicomponent. A summary of the progress made during the graph reduction steps and the application of the 2-vertex separators is given in the first two columns of Table 1.

The reduction procedure decreased the number of vertices in the graph by one-third, while applying the easy 2-vertex separators, k-coring and largest bicomponent identification further reduced the graph to 40% of its original size. The search for the largest tricomponent was completed with three more iterations using all 2-vertexseparators, plus a final iteration to confirm that no more separators remained in the graph.

While our primary focus is on the single dominant tricomponent, we also kept track of the smaller k-components that were trimmed from the graph. In this analysis, we did not consider nested components (e.g. 4-components that may reside within these small tricomponents) and limited ourselves to the lowest level of cohesion. Given their small size relative to the 10,177-vertex tricomponent, we did not feel that exploring the nested structures provided additional insights. Completing the analysis would be trivial using standard cohesion algorithms.

In total, 2911 k-components ($3 \leq k \leq 18$) ranging in size from 4 to 34 vertices were identified. More than half of these only contained four vertices and the vast majority (97%)

contained fewer than ten vertices. The small, high-cohesion components mostly arose from joint authorship within a research group and included a 19-member clique.

5.1.2. Identifying 4-components—Starting with the largest tricomponent, we first apply the graph reduction steps before beginning the search for 3-vertex separators. We also include the additional step of identifying and applying the 2-vertex separators since the iterative application of the clique processing, *k*-coring and largest bicomponent extraction can result in a new graph that is not necessarily a tricomponent. These steps only take several minutes to complete and result in a new tricomponent that is one-third the size of the starting tricomponent.

We next search for and apply the 3-vertex separators to finish the identification of the 4-component. We begin by identifying and applying the easy-to-find 3-vertex separators (see Section 3). This involves iterating over the edges in the graph and determining the articulation points that result after removing both end points of the edge. We then continue with searches for all possible 3-vertex separators. To help accelerate convergence, we reduce the graph to 4-cores after each round and then recover the largest tricomponent (as noted earlier, *k*-coring can introduce new lower-order vertex separators) The results are shown in middle two columns of Table 3.

Although our focus continues to be on the largest 4-component, we again kept track of the smaller *k*-components that were found along the way. In total, 876 *k*-components ($4 \leq k \leq 17$) ranging in size from 5 to 36 vertices were found. Nearly half of these contained five vertices and most (90%) contained fewer than ten vertices. As we had seen during the search for the tricomponents, the high-cohesion components were mostly formed from cliques or near-cliques.

5.1.3. Identifying higher order (*k* > 4) components—Starting with the largest 4-component, we first apply the graph reduction steps. Since the reduced graph may contain new lower-order separators, we then find and apply the 2-vertex separators to recover the largest tricomponent, followed by the identification and application of the 3-vertex separators to recover the largest 4-component. The final result was a 33-vertex graph that could easily be handled using standard cohesion algorithms and an explicit search for 4-vertex separators was not necessary (see last two columns of Table 1).

An analysis of the results shows that there is not a single dominant 5-component, but rather a near continuum of 86 small 5-components ranging in size from 6 to 33 vertices. Note that the final graph listed in Table 1 is not a 5-component. Rather the largest 5-component was split off from the main graph at an earlier stage of the processing. We also found 61 higher-order ($k > 5$) components including cliques of up to 14 vertices and a 6-component containing 19 vertices. No additional processing was performed at this point since all remaining components are sufficiently small that the computational effort is trivial.

5.2. Collaboration networks

To demonstrate that our method could be generalized to other problems, we tested it on two other data sets that are made avail-able online by Batagelj and Mrvar [30]: Collaboration

network in computational geometry [31,32] and Erdös collaboration network [30]. While these are both considerably smaller than the sociology dataset, they present a different set of challenges. Specifically, the 5-components for these graphs are much larger and required extra processing to explicitly find the 4-vertex separators using an extension of the technique described in Section 3.3.2.

5.3. Timings

The run times for the application of our method to the three datasets are given in Table 2. Since the geometry and Erdös networks, together with their corresponding bicomponents, are much smaller than the sociology data set, the times required to identify the largest tricomponents are considerably lower. The 4-components for these data sets were also more resistant to our reduction techniques and the time required to find the largest 5-components turned out to be much longer. This places an extra burden on the standard maximum-flow based algorithms to find the higher order components: 1221 and 1359 s for processing the geometry and Erdös5-components, respectively. Nonetheless, being able to quickly reduce the original graphs to kernels of a few hundred vertices makes the cohesion analysis of these networks tractable.

6. Discussion

The primary advantage of our approach to identifying higher order *k*-components is that it can be applied to larger problems that were not previously solvable in a reasonable amount of time. In particular, we demonstrated that the sociology co-authorship graph described earlier could be fully decomposed into nested *k*-components in just over 14 min (Table 2). While we did not attempt to solve the entire problem from scratch using the standard maximum-flow based algorithm, applying it just to the 1850-vertex graph that contained no easy-to-find 3-vertex separators required nearly 24 h to complete. Using the igraph implementation of Kanevsky's general purpose algorithm for finding all minimum size vertex separators [33] in this same graph took just over three hours to identify the few remaining 3-vertex separators. Given that multiple iterations of vertex identification are necessary, simply dropping Kanevsky's algorithm into our framework would have resulted in an estimated time to solution for finding the largest 4-component starting from the largest tricomponent of more than 30 h.

Since the application of preexisting exact algorithms to a problem of this size are not practical, we compared the performance of our approach to the approximate algorithm referenced earlier [4] and as implemented in the NetworkX Python package [34] (Table 3).

In addition to providing an exact result, the method described here is significantly faster than the approximate algorithm. We were unable to complete the analysis when using the approximate algorithm and starting from the largest bicomponent, but estimate a run time of nearly nine days assuming quadratic scaling with graph size. Even if this assumption is incorrect (i.e. scaling is better than quadratic), the time to solution for the approximate method applied to the largest tricomponent is already 100× longer than for the exact method. While the absolute performance of software certainly depends on the implementation details

of the algorithms, we do not believe that implementation differences alone between igraph and NetworkX can account for the orders of magnitude variance in run time.

Of course, we are also interested in the differences between the results generated by the exact and approximate algorithms. For example, we find that the approximate algorithm does not recognize what we identified as the largest 4-component (independently confirmed using the NetworkX *node connectivity* function) as in fact being 4-connected. Instead, it predicts that this 1832-vertex graph contains a 1642-vertex 3-component.

Nonetheless, we feel that both the exact and approximate algorithms can play an important role in the analysis of large graphs. Our exact algorithm may not be able to handle substantially larger graphs and approximate methods will still be needed. While the current implementation of the approximate algorithm is slower than our exact algorithm, it can likely be modified to run considerably faster by incorporating some of the additional graph reduction operations that we introduced, particularly those that exploit the properties of cliques.

Given that our approach is a collection of established and novel techniques rather than a single algorithm, it is difficult to assess the overall computational complexity. The efficiency will depend strongly on properties of the graph such as the fraction of vertices that can be classified as isolated at each level of cohesion. For our sociology case study, the bottleneck was locating all of the hard-to-find 3-vertex separators for which no members of the set are neighbors with each other, but for the other two networks finding the 4-vertex separators accounted for a significant fraction of the time.

Since the preexisting algorithms that we leverage for finding articulation points, performing vertex disjoint path calculations or identifying cliques scale at least linearly with the number of edges or vertices, all graph reductions lead to better performance. For example, in Section 3.1 we described how we could create an auxiliary graph with fewer vertices and edges than the original graph. This reduces both the number of vertices that need to be considered in the search for separators and the cost of an individual vertex disjoint path calculation, which scales as $O(|V||E|)$.

None of the individual steps described in this paper are conceptually difficult to understand or implement and, with the possible exception of the algorithms for finding the two classes of 3-separators (Section 3.3), are fairly intuitive. Nonetheless, this is the first instance of which we are aware that they have been applied to the problem of finding k-components.

Another strength of our approach is its modularity. Earlier in the development of our software, we employed a brute force search for the 3-vertex separators that was analogous to our method for finding 2-vertex separators – remove pairs of vertices and identify the articulation points in the resulting graph. While we were able to eventually find all of the k-components in this way, each cycle of 3-vertex separator identification took nearly two hours. Replacing with the more sophisticated search for tricomponents (Section 3.3) reduced the run time by two orders of magnitude. In a similar manner, researchers can independently develop their own algorithms for finding vertex separators and integrate into our framework. For example, replacing what we admit to be an inefficient algorithm for finding 2-vertex

separators with a better method can certainly reduce the time to solution. Others may simply want to restrict themselves to using our pre-processing steps to make the problem more tractable before applying alternative methods for finishing the problem.

Our focus throughout this work was on the development of efficient serial algorithms. It would certainly be straightforward to parallelize the more time-consuming steps, especially the searches for the higher order separators. We find it likely that the time to solution for the co-authorship problem highlighted here could be reduced by another factor of 3–5, with even greater savings for larger graphs. Another potential source of performance improvement would be to re-implement our software using a compiled language. We chose the R language for this work since it is widely used in the social sciences and enabled very rapid prototyping of new ideas. Finally, we would like to consider the application of *k*-edge-components for graph reduction in future versions of our software. Recent advances in the development of fast algorithms in this area [35,36] make them an appealing addition to our toolkit.

As a consequence of its modularity, our framework also makes it possible to incrementally approach the final result. While we were ultimately able to complete the case studies described above in a modest amount of time, larger and more complex problems will certainly benefit from the ability to save intermediate results along the way. This makes it easier to both test individual components of the software and to restart in the event of a hardware crash.

In summary, this work describes three novel contributions that make it much easier to identify cohesive subgroups in large graphs:

1. The use of a new graph reduction technique based on cliques. This is in addition to the *k*-core and bicomponent based techniques that had been used earlier [4].
2. An exploitation of the property of cliques to reduce the set of vertices that need to be considered during a search for minimum size vertex separators.
3. An efficient approach to identifying the easy- and hard-to-find 3-vertex separators within a 4-component.

Software used for the construction of the structurally cohesive subgroups is available at https://github.com/sinkovit/k-components.

Acknowledgments

RSS was partially supported by NSF grants OCI #0910847 Gordon: A Data Intensive Supercomputer and ACI #1053575 XSEDE: eXtreme Science and Engineering Discovery Environment (XSEDE) through the ECSS program. JM was partially supported by NIH grant NICHD HD075712 Models and Tools for Dynamic Health-Relevant Diffusion over Complex Networks.

References

1. Harary F. Graph Theory, Addison-Wesley, Reading, MA. 1969
2. Menger K. Zur all gemeinen kurventheorie. Fund. Math. 1927; 10(1):96–115.
3. Moody J, White DR. Structural cohesion and embeddedness: a hierarchical concept of social groups. Am. Sociol. Rev. 2003; (2016):103–127.

4. Torrents J, Ferraro F. Structural cohesion: visualization and heuristics for fast computation. J. Soc. Struct. 2015; 16(8):1–35.
5. White DR, Newman M. Fast approximation algorithms for finding node-independent paths in networks. Santa Fe Inst. Work. Pap. Ser. 2001
6. Eppstein, D., Löffler, M., Strash, D. Lecture Notes in Computer Science. Springer; 2010. Listing all maximal cliques in sparse graphsin near-optimal time; p. 403-414.
7. Hopcroft J, Tarjan R. Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM. 1973; 16(6):372–378.
8. Batagelj V, Zaversnik M. An O (m) algorithm for cores decomposition of networks. 2003 arXiv preprint cs/0310049.
9. Batagelj V, Zaveršnik M. Fast algorithms for determining (generalized) core groups in social networks. Adv. Data Anal. Classif. 2011; 5(2):129–145.
10. Goldberg AV, Tarjan RE. A new approach to the maximum-flow problem. JACM. 1988; 35(4): 921–940.
11. Seidman SB. Network structure and minimum degree. Soc. Networks. 1983; 5(3):269–287.
12. White DR, Harary F. The cohesiveness of blocks in social networks: node connectivity and conditional density. Soc. Methodol. 2001; 31(1):305–359.
13. Whitney H. Congruent graphs and the connectivity of graphs. Am. J. Math. 1932; 54(1):150–168.
14. Kreager DA, Rulison K, Moody J. Delinquency and the structure of adolescent peer groups. Criminology. 2011; 49(1):95–127. [PubMed: 21572969]
15. Mani D, Moody J. Moving beyond stylized economic network models: the hybrid world of the Indian firm ownership network. AJS. 2014; 119(8):1629. [PubMed: 25418990]
16. Moody J, Mucha PJ. Portrait of political party polarization. Network Sci. 2013; 1(01):119–121.
17. Moody J, Coleman J. Clustering and cohesion in networks: concepts and measures. Int. Encyclop. Soc. Behav. Sci. 2014
18. Newman ME. Modularity and community structure in networks. Proc. Natl. Acad. Sci. 2006; 103(23):8577–8582. [PubMed: 16723398]
19. Porter MA, Onnela J-P, Mucha PJ. Communities in networks. Not. AMS. 2009; 56(9):1082–1097.
20. Tomita E, Tanaka A, Takahashi H. The worst-case time complexity for generating all maximal cliques and computational experiments. T eor. Comput. Sci. 2006; 363(1):28–42.
21. Tarjan R. Depth-first search and linear graph algorithms. SIAM J. Comput. 1972; 1(2):146–160.
22. Hopcroft JE, Tarjan RE. Dividing a graph into triconnected components. SIAM J. Comput. 1973; 2(3):135–158.
23. Saifullah, AM., Üngör, A. Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory. Vol. 94. Australian Computer Society, Inc.; 2009. A Simple Algorithm for Triconnectivity of a Multigraph.
24. Hsu, T-s, Ramachandran, V. Foundations of Computer Science, Proceedings 32nd Annual Symposium On. 1991. IEEE; 1991. A linear time algorithm for triconnectivity augmentation.
25. Miller GL, Ramachandran V. A new graph triconnectivity algorithm and its parallelization. Combinatorica. 1992; 12(1):53–76.
26. Gutwenger, C., Mutzel, P. Graph Drawing. Springer; 2000. A linear time implementation of SPQR-trees.
27. Jiang, Z. Masters Thesis. University of Windsor; 2013. An Empirical Study of 3-vertex Connectivity Algorithms.
28. Moody J. The structure of a social science collaboration network: disciplinary cohesion from 1963 to 1999. Am. Sociol. Rev. 2004; 69(2):213–238.
29. Csardi G, Nepusz T. The igraph software package for complex network research. InterJournal Complex Syst. 2006; 1695(5):1–9.
30. Batagelj V, Mrvar A. Pajek datasets. 2006 Available from: http://vlado.fmf.uni-lj.si/pub/networks/data/.
31. Beebe NHF, Nelson HF. Beebe's Bibliographies Page. 2002 Available from:http://www.math.utah.edu/~beebe/bibliographies.html.

32. Jones B. Computational Geometry Database. 2002 Available from: http://jeffe.cs.illinois.edu/compgeom/biblios.html.
33. Kanevsky A. Finding all minimum-size separating vertex sets in a graph. Networks. 1993; 23(6): 533–541.
34. Schult, DA., Swart, P. Exploring network structure, dynamics, and function using NetworkX. Proceedings of the 7th Python in Science Conferences (SciPy 2008); 2008. p. 11-16.
35. Akiba, T., Iwata, Y., Yoshida, Y. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. Proceedings of the 22nd ACM International Conference on Information &Knowledge Management; 2013. p. 909-918.
36. Yuan L, et al. I/O efficient ECC graph decomposition via graph reduction. Proceedings of the VLDB Endowment. 2016; 9(7):516–527.

Biographies

Robert S. Sinkovits obtained his Ph.D. in Physics from the University of Connecticut and his B.S. in Engineering Physics from Lehigh University. He is currently the Director of the Scientific Computing Applications Group at the San Diego Supercomputer Center and a co-PI on the NSF-funded Comet award. His areas of interests are performance optimization, graph algorithms and the development of software for the life sciences (structural biology, flow cytometry, wildlife tracking).

James Moody is the Robert O. Keohane professor of sociology at Duke University. He has published extensively in the field of social networks, methods, and social theory. His work has focused theoretically on the network foundations of social cohesion and diffusion, with a particular emphasis on building tools and methods for understanding dynamic social networks. He has used network models to help understand school racial segregation, adolescent health, disease spread, economic development, and the development of scientific disciplines. His work is funded by NSF, NIH, RWJF and the JSMF.

B. Tolga Oztan holds a PhD in Mathematical Behavioral Sciences from University of California, Irvine with a research focus on comparative cross-cultural studies. His most

recent work studies the connection between cooperation and kinship in pre-industrial societies using statistical methods that deal with non-independent data points and phylogenetic tree analysis. He also utilizes social network analysis methods and game theory simulations to better understand cooperation and kinship behavior.

Douglas R. White, mathematical anthropologist, rediscovered the 1927 Menger theorem of structural cohesion as a predictor (with Moody and Harary) of diverse sociological phenomena, is a theorist of network dynamics, dynamical city-size and world-system trade, a complex dynamics theorist, creator of the Complex Social Science cross-cultural data analysis port (with network autocorrelation), editor of the Wiley companion to cross-cultural research and developer of its Standard sample. Fieldwork includes three Amerindian societies, Ireland, and Austria.

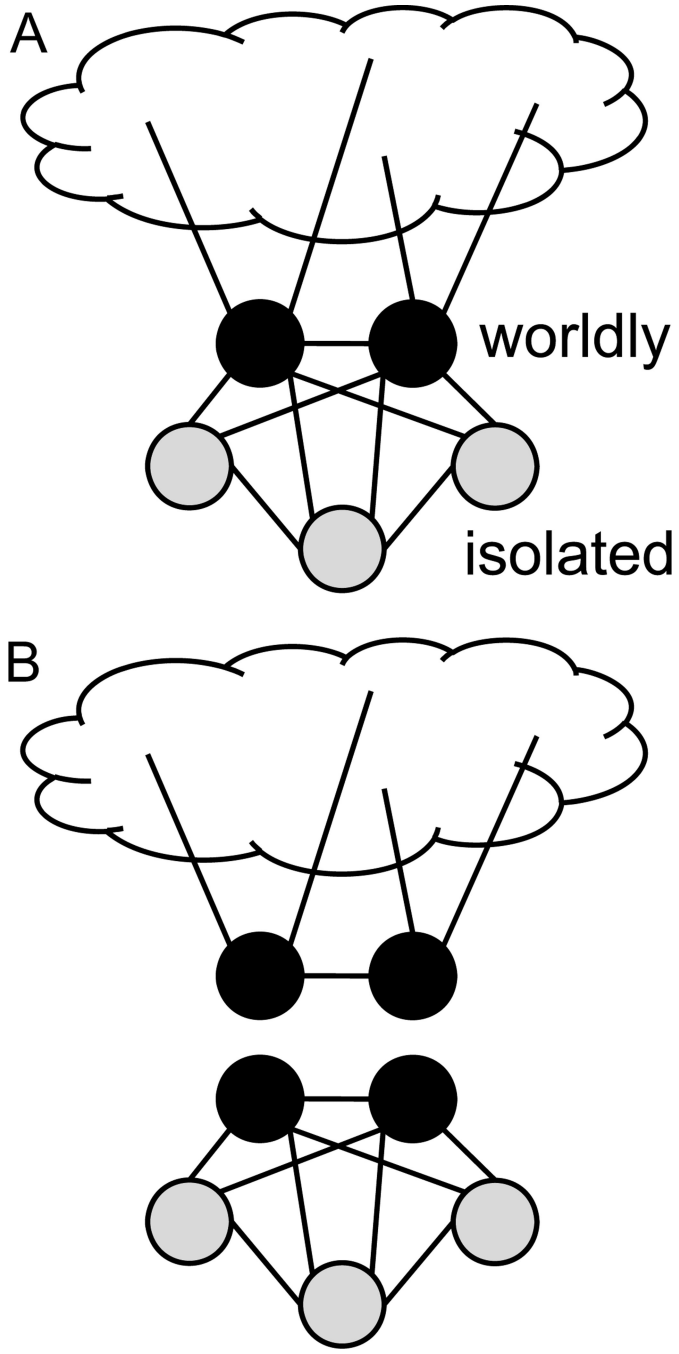

Fig. 1.

Graph (A) with a maximal clique of size five containing two “worldly” vertices (black) and three “isolated” vertices (gray). The cloud represents the remainder of the graph and contains all vertices that are not a member of the clique. The worldly vertices in the graph form a 2-vertex separator, which can be used to partition the graph into two cohesive subgroups (B).

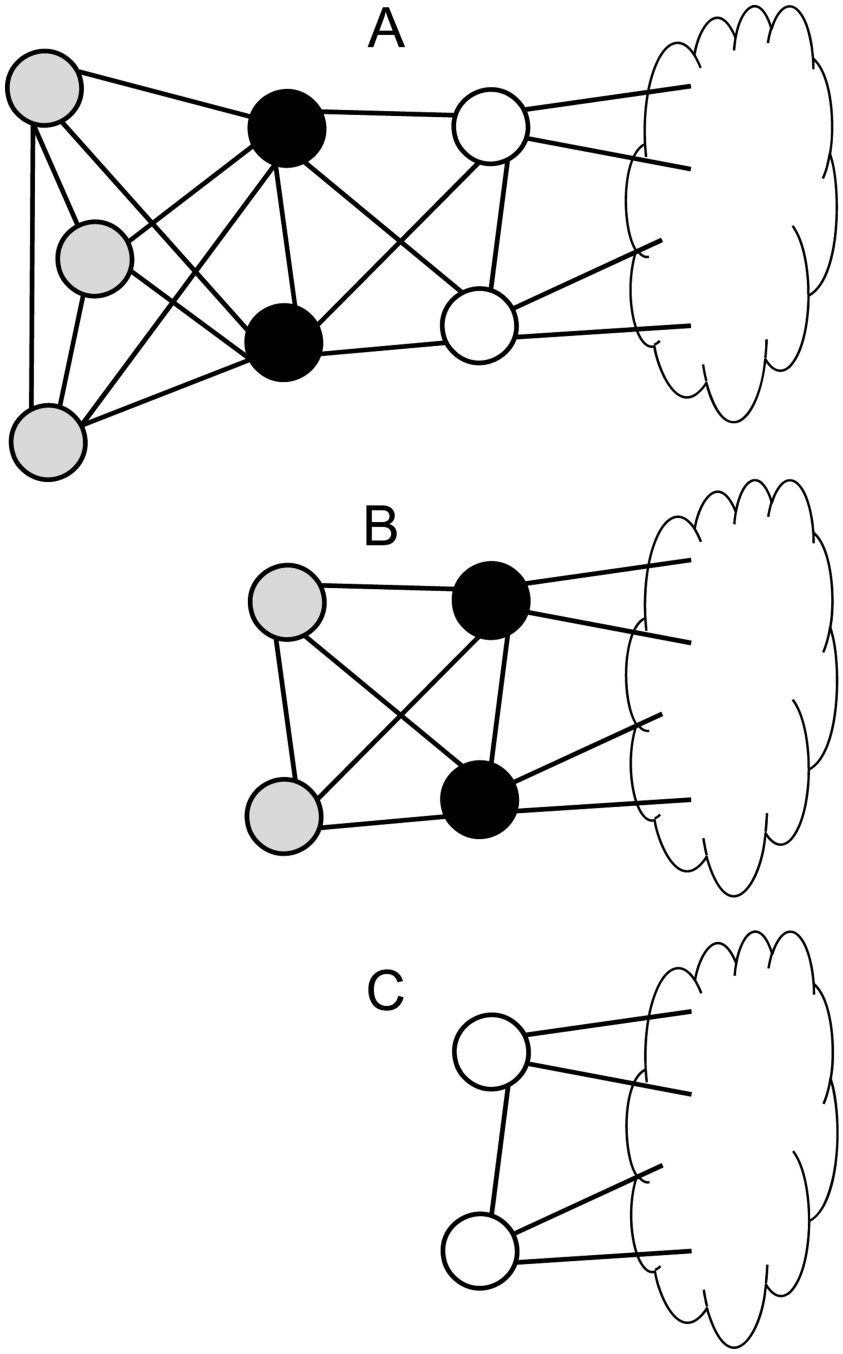

Fig. 2.

Example illustrating how removal of isolated vertices from a clique can change the worldliness of the remaining vertices. Isolated and worldly vertices are shown in gray and black, respectively, for clique at edge of the graph; cloud rep-resents remainder of the graph. After deletion of isolated vertices in graph A, the formerly worldly vertices are now isolated in graph B and become candidates for removal in next round of clique processing, resulting in graph C.

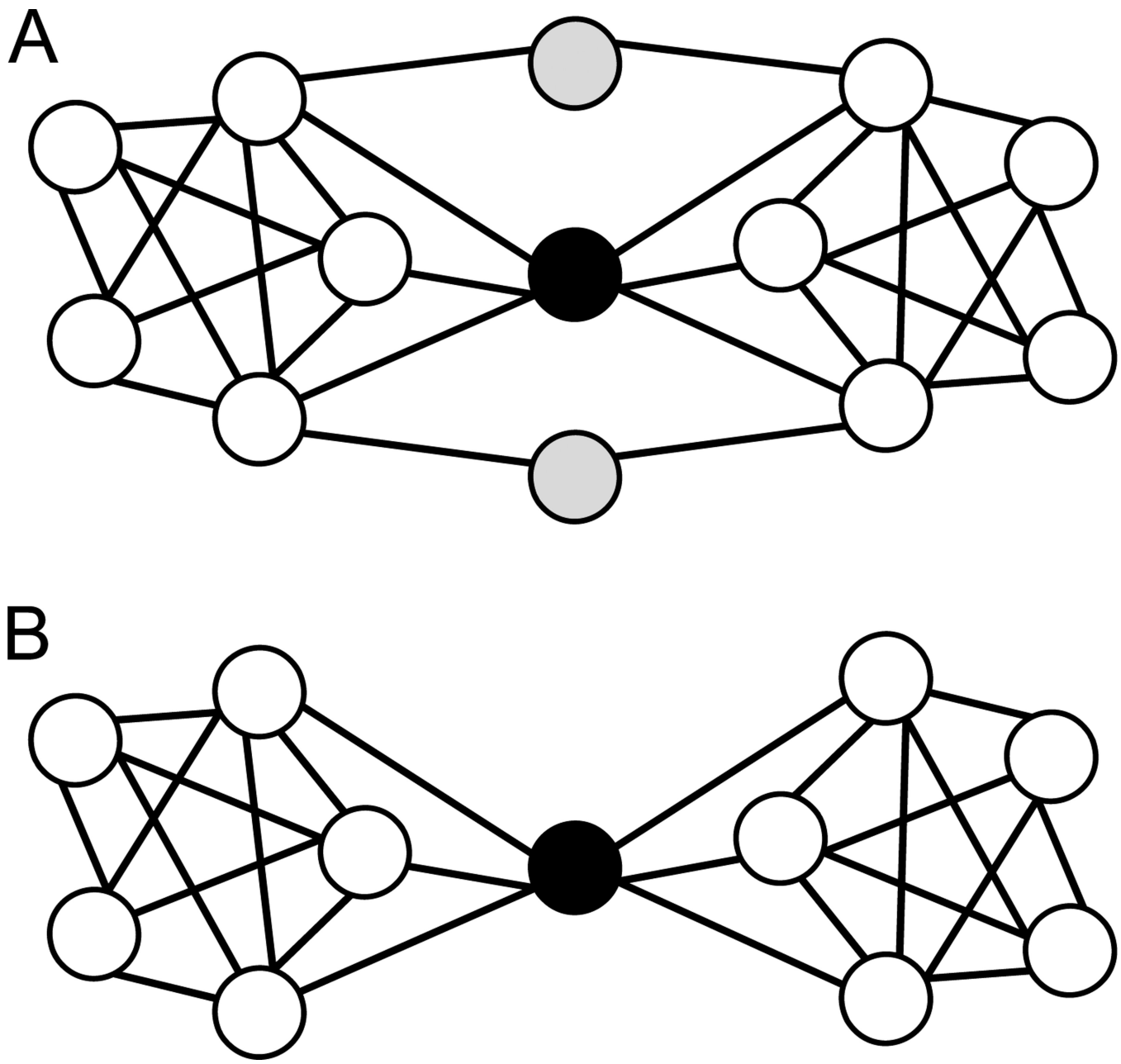

Fig. 3.

Example showing how the process of k-coring can result in a graph with new articulation points. Reducing the bicomponent (A) to a 3-core (B) removes two vertices (gray) in the top graph, with the outcome that the black vertex becomes an articulation point in the new graph.

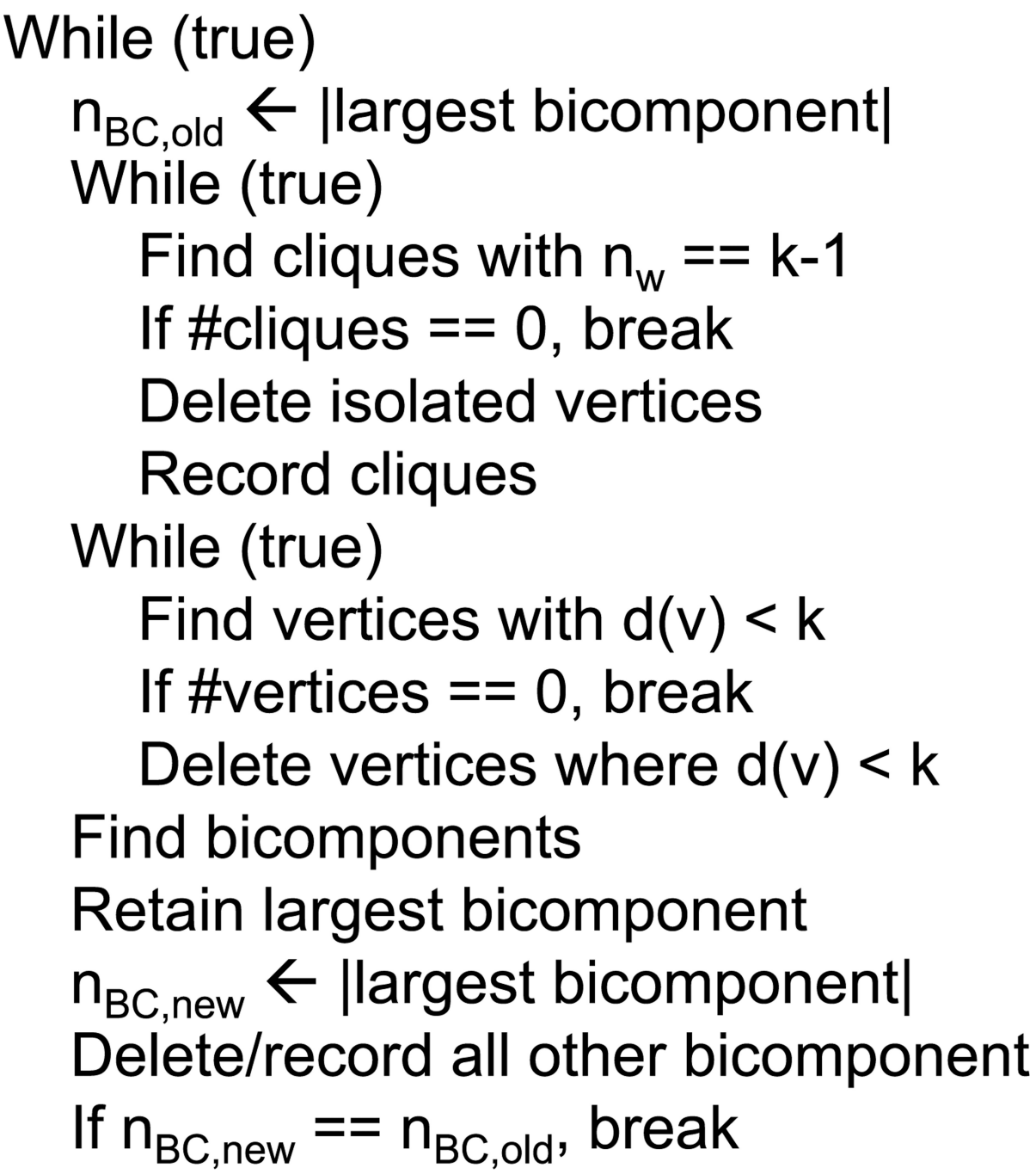

Fig. 4.

Pseudocode for graph simplification procedure. Starting with (k−1)-component, the algorithm generates a new bicomponent that excludes many of the vertices that cannot possibly belong to the dominant k-component.

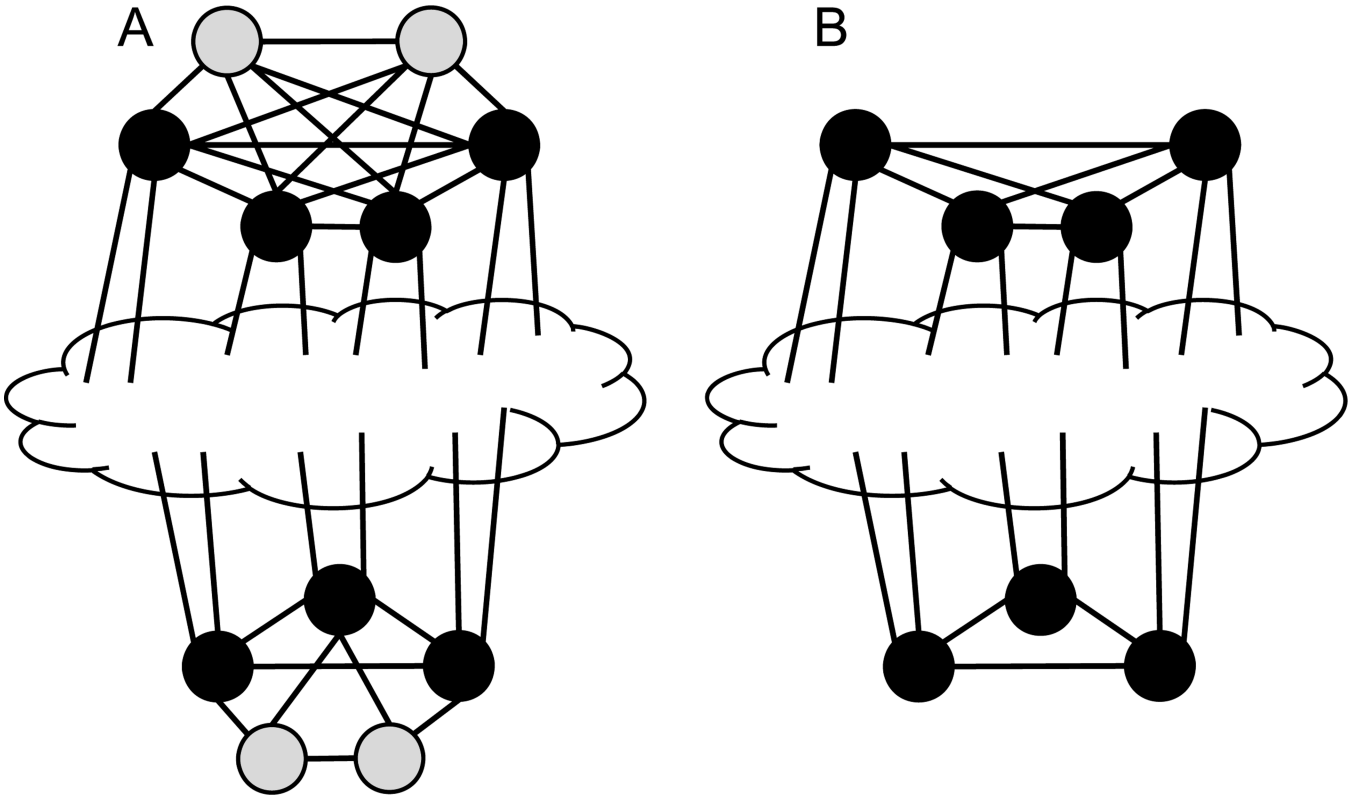

Fig. 5.

Example showing how isolated vertices, regardless of the size or the number of worldly vertices in the clique, can be removed from the graph without altering the number of connected components. As a result, these vertices cannot be members of a vertex separator. In addition, the removal of these isolated vertices does not change the number of vertex disjoint paths between any pair of vertices not belonging to the same clique.

For e in E
  Select end points $\{v_1, v_2\}$ of edge e
  Test $\{v_1, v_2\}$ is 2-vertex separator

For $v_1 \leftarrow V$
  $G' \leftarrow G - v_1$
  Identify articulation points $\{v_{art}\}$ of G'
  For $v_2 \leftarrow \{v_{art}\}$
    Add $\{v_1, v_2\}$ to list of 2-vertex separators

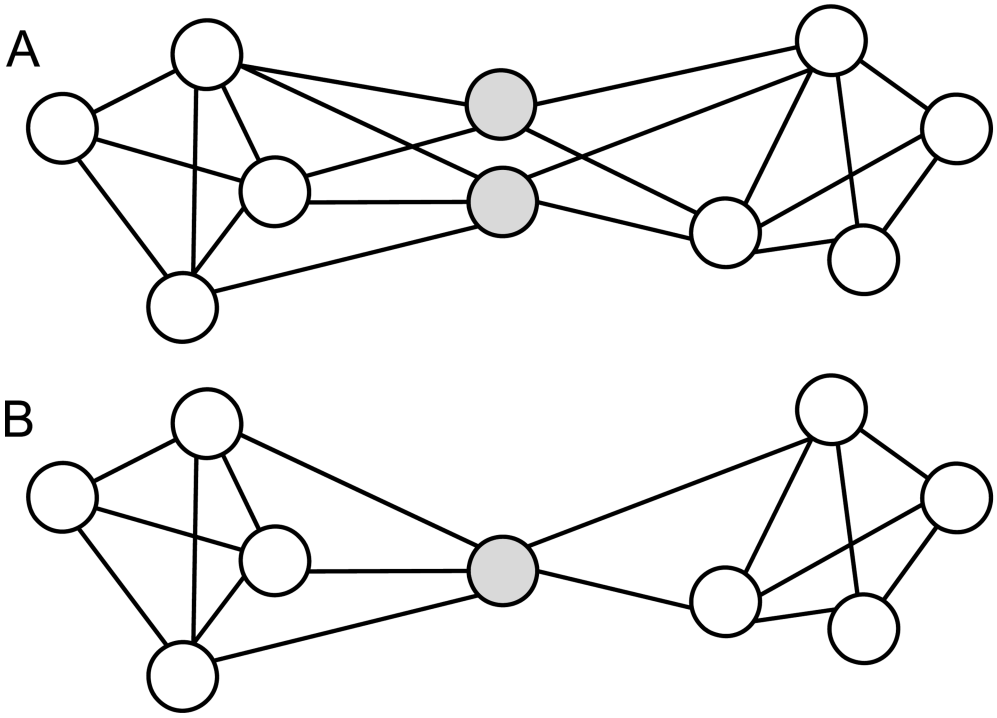

Fig. 6.

Pseudocode illustrating search for 2-vertex separators. First code block finds only those separators where the pair of vertices is joined by an edge; second code block finds all separators. The graphs demonstrate how 2-vertex separators can be identified by removing a test vertex and searching for the articulation points in the new graph. In this example, the lower gray vertex becomes an articulation point after the removal of the upper gray vertex.

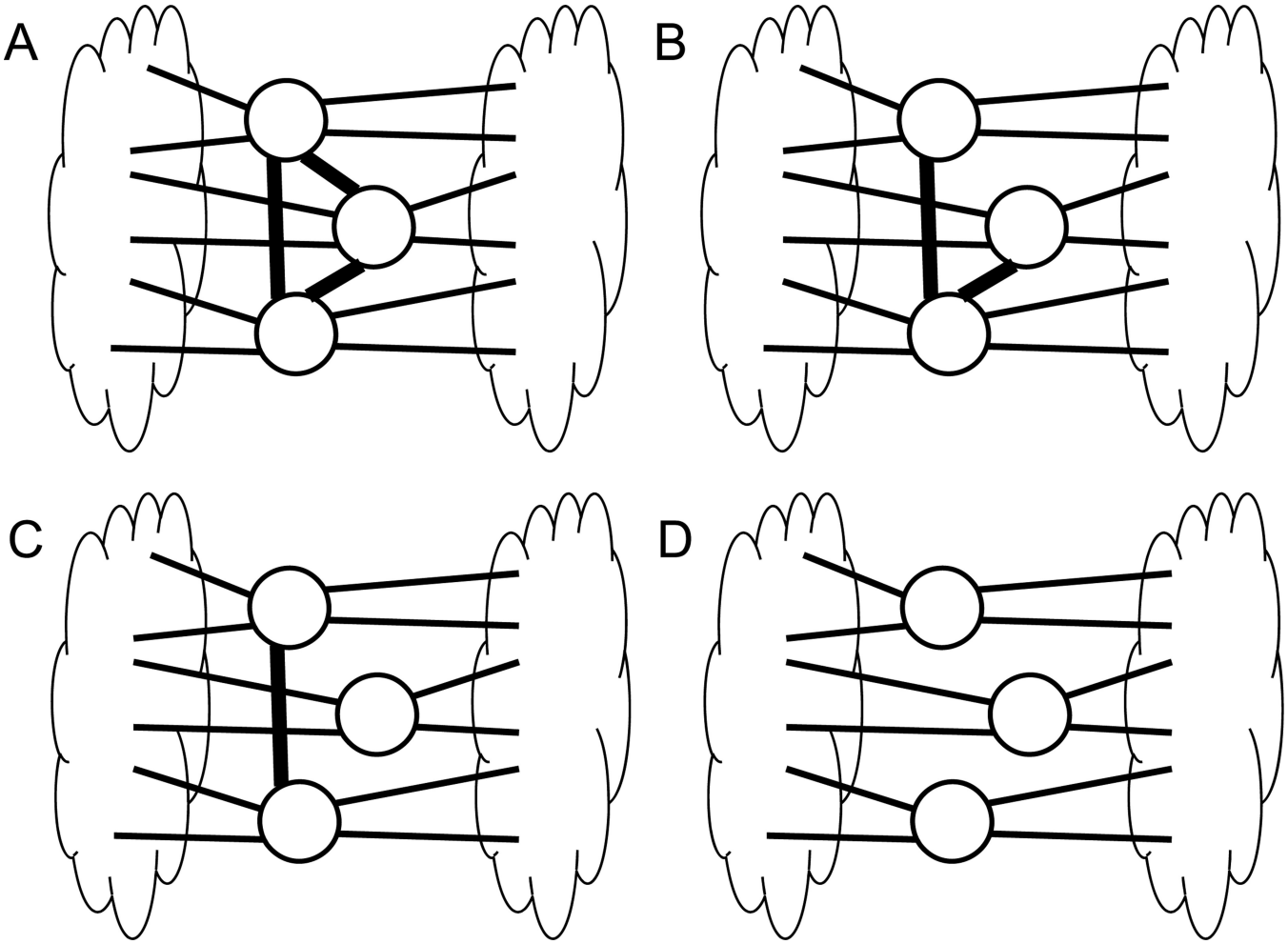

Fig. 7.

3-vertex separators with different level of connectedness. In graphs A-–C, at least two of the three vertices forming the separator are joined by edges (shown with heavier lines for clarity). These separators can be found easily by iterating over edges and looking for articulation points that result from the removal of the edge endpoints. In graph D, none of the three vertices forming the separator are joined by an edge and an alternative search strategy is required.

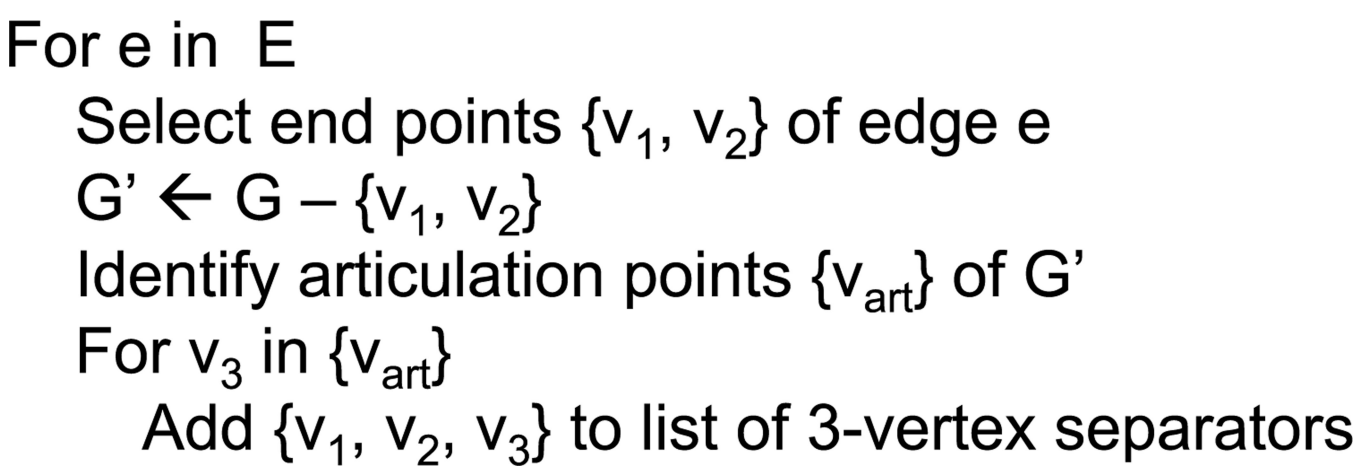

Fig. 8.
Pseudocode for identifying the easy to find 3-vertex separators where at least two of the members are neighbors.

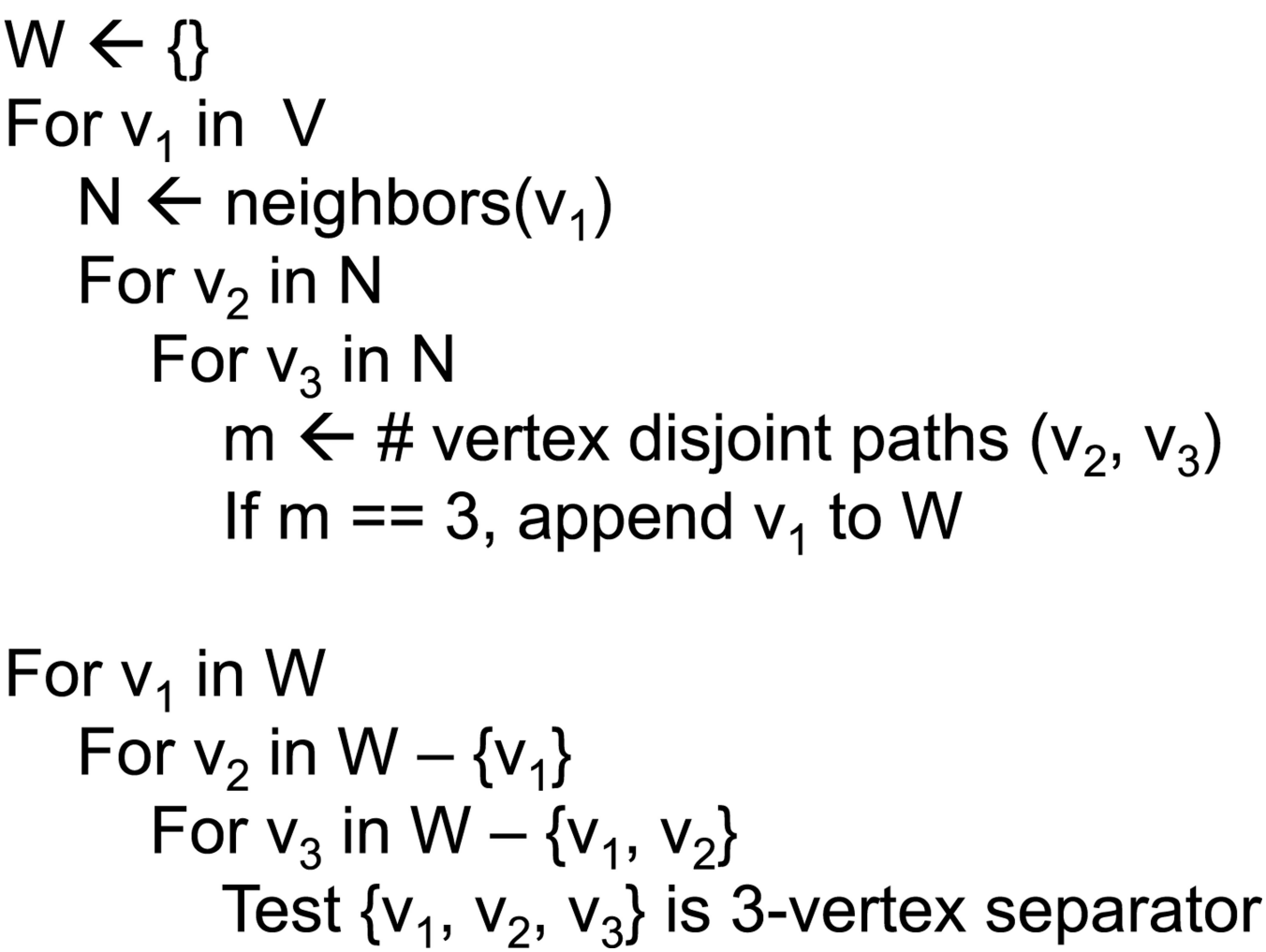

Fig. 9.
Pseudocode for identifying the hard-to-find 3-vertex separators where none of the members are neighbors.

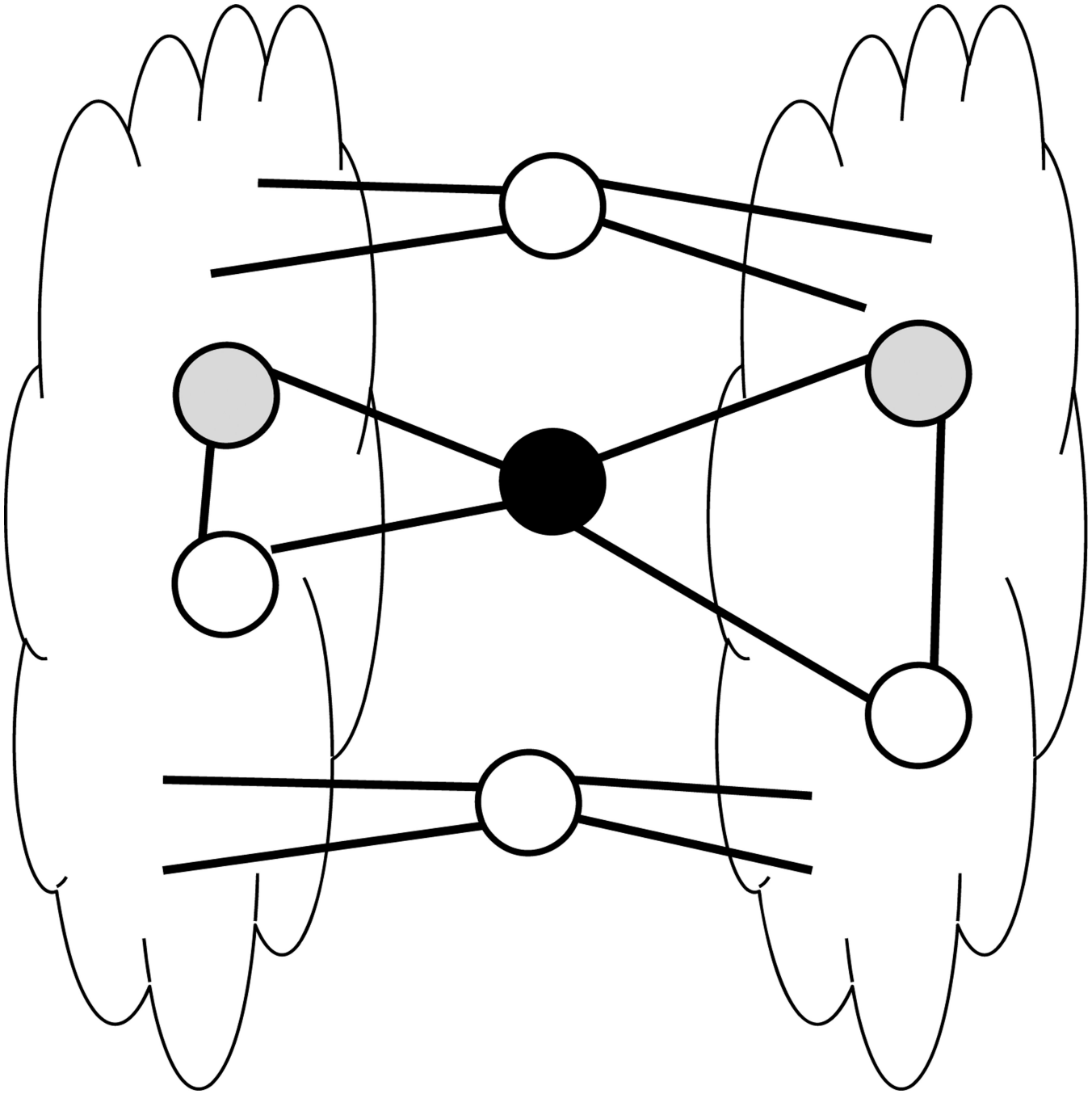

Fig. 10.

3-vertex separator with no edges between the member vertices. In order for a vertex to be a member of a 3-vertex separator, at least one pair of the neighbours must be joined by exactly three vertex disjoint paths. Note that not all pairs of vertices neighboring the candidate vertex (shown in black) need to be tested. If two of the neighbors are themselves neighbors, then we only need to calculate the number of vertex disjoint paths between one member of the pair and the other neighbors. In this case, calculations can be limited to the pair of gray vertices.

```
While (true)
    Find easy 2-separators
    If #2-separators == 0, break
    For {v1, v2} in 2-separators
        Process smaller disjoint graph(s)
        Trim vertices from larger graph
    Reduce to 3-core
    Reduce to largest bicomponent

While (true)
    Find all 2-separators
    If #2-separators == 0, break
    For {v1, v2} in 2-separators
        Process smaller disjoint graph(s)
        Trim vertices from larger graph
    Reduce to 3-core
    Reduce to largest bicomponent
```

Fig. 11.

Pseudocode for identifying the largest tricomponent. This procedure would normally be applied to the simplified graph obtained by k-coring and reduction to largest bicomponent.

```
While (true)
    Find easy 3-separators
    If #3-separators == 0, break
    For {v1, v2, v3} in 3-separators
        Process smaller disjoint graph(s)
        Trim vertices from larger graph
    Reduce to 4-cores
    Reduce to largest bicomponent
    Reduce to largest tricomponent

While (true)
    Find all 3-separators
    If #3-separators == 0, break
    For {v1, v2, v3} in 3-separators
        Process smaller disjoint graph(s)
        Trim vertices from larger graph
    Reduce to 4-cores
    Reduce to largest bicomponent
    Reduce to largest tricomponent
```

Fig. 12.

Pseudocode for identifying the largest 4-component. This procedure would normally be applied to the simplified graph obtained by clique processing, k-coring and reduction to largest tricomponent.

Table 1

Size of graph during search for largest 3-, 4- and 5-components. The columns labeled Step list the graph reduction steps and the corresponding columns labeled |V| contain the size of the reduced graph. Explanation of terms: start = starting graph used for reduction to k-component; k-core/bi = reduction to k-core followed by reduction to largest bicomponent; clique = identification of cliques and removal of isolated vertices; tri = reduction to largest tricomponent; easy k (n), all k (n) = identification of easy-to-find or all k-vertex separators, where n is the number of k-vertex separators. Note that last entry in table (33*) was not a 5-component, but was small enough to be processed using standard techniques.

Finding 3-component		**Finding 4-component**		**Finding 5-component**	
Step	**|V|**	**Step**	**|V|**	**Step**	**|V|**
start	29,462	start	10,177	start	1832
k-core/bi	19,412	clique	7963	clique	1573
k-core/bi	19,242	k-core/bi	4602	k-core/bi	769
k-core/bi	19,239	clique	4285	clique	712
easy 2 (2101)	13,315	k-core/bi	4101	k-core/bi	672
k-core/bi	11,899	clique	4065	clique	671
all 2 (462)	10,597	k-core/bi	4055	all 2 (13)	623
k-core/bi	10,476	all 2 (135)	3571	all 2 (5)	584
all 2 (91)	10,250	all 2 (15)	3541	easy 3 (39)	404
k-core/bi	10,228	all 2 (6)	3525	k-core/bi	376
all 2 (13)	10,184	easy 3 (205)	2926	tri	296
k-core/bi	10,177	k-core/bi	2688	easy 3 (7)	265
		tri	2616	k-core/bi	254
		easy 3 (79)	2448	tri	254
		k-core/bi	2375	easy 3 (5)	233
		tri	2363	k-core/bi	229
		easy 3 (36)	2203	tri	229
		k-core/bi	2153	easy 3 (3)	224
		tri	2148	k-core/bi	224
		easy3 (23)	2084	tri	224
		k-core/bi	2049	easy 3 (3)	188
		tri	2040	k-core/bi	178
		easy3 (7)	2020	tri	33*

J Comput Sci. Author manuscript; available in PMC 2017 November 01.

Finding 3-component		Finding 4-component		Finding 5-component	
Step	**|V|**	**Step**	**|V|**	**Step**	**|V|**
		k-core/bi	2014		
		tri	2008		
		easy 3 (7)	1925		
		k-core/bi	1891		
		tri	1888		
		easy 3 (2)	1880		
		k-core/bi	1879		
		tri	1879		
		easy3 (2)	1876		
		k-core/bi	1863		
		tri	1863		
		easy 3 (3)	1853		
		k-core/bi	1850		
		tri	1850		
		all 3 (3)	1833		
		k-core/bi	1832		

Table 2

Sizes of largest k-components at each level of cohesion for the three data sets described in Sections 5.1 and 5.2. Columns 3, 5 and 7 list the times needed to extract the largest k-components from the (k − 1) components.

	Sociology		Geometry		Erdös	
k	**size**	**t (s)**	**size**	**t (s)**	**size**	**t (s)**
0	128,151		7343		6927	
1	68,285	<1	3621	<1	6927	<1
2	29,462	<1	1901	<1	2145	<1
3	10,177	411	1052	5	1191	9
4	1832	420	620	94	721	272
5	33	18	358	222	469	372
total		**851**		**321**		**653**

Table 3

Comparison of performance for the exact algorithm (described here) and the approximate algorithm (Torrents and Ferraro). Run times correspond to time needed to extract largest k-components from (k − 1)-components (exact) or complete analysis from (k − 1)-component (approx). The starred entry was an estimate based on quadratic scaling with graph size.

k	size	t_{exact} (s)	t_{approx} (s)
2	29,462	<1	<1
3	10,177	411	768,090 *
4	1832	420	91,649
5	33	18	2939

* Indicates that the time was an estimate based on extrapolation.