Data & Knowledge Engineering 72 (2012) 285–303

Contents lists available at SciVerse ScienceDirect

Data & Knowledge Engineering

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

SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices☆

Mijung Kim*, K. Selçuk Candan

Arizona State University, 699 S. Mill Avenue, Tempe, AZ 85281, USA

ARTICLE INFO

Article history:
Received 6 January 2011
Received in revised form 8 November 2011
Accepted 8 November 2011
Available online 21 November 2011

Keywords:
Clustering
Mining methods and algorithms
Graph partitioning
Vertex-cut

ABSTRACT

Graphs are used for modeling a large spectrum of data from the web, to social connections between individuals, to concept maps and ontologies. As the number and complexities of graph based applications increase, rendering these graphs more compact, easier to understand, and navigate through are becoming crucial tasks. One approach to graph simplification is to partition the graph into smaller parts, so that instead of the whole graph, the partitions and their inter-connections need to be considered. Common approaches to graph partitioning involve identifying sets of edges (or edge-cuts) or vertices (or vertex-cuts) whose removal partitions the graph into the target number of disconnected components. While edge-cuts result in partitions that are vertex disjoint, in vertex-cuts the data vertices can serve as bridges between the resulting data partitions; consequently, vertex-cut based approaches are especially suitable when the vertices on the vertex-cut will be replicated on all relevant partitions. A significant challenge in vertex-cut based partitioning, however, is ensuring the balance of the resulting partitions while simultaneously minimizing the number of vertices that are cut (and thus replicated). In this paper, we propose a SBV-Cut algorithm which identifies a set of *balance vertices* that can be used to *effectively* and *efficiently* bisect a directed graph. The graph can then be further partitioned by a recursive application of structurally-balanced cuts to obtain a hierarchical partitioning of the graph. Experiments show that SBV-Cut provides better vertex-cut based expansion and modularity scores than its competitors and works several orders more efficiently than *constraint-minimization* based approaches.

1. Introduction

Today, graphs and networks are used for modeling a large spectrum of data from the web, to social connections between individuals, to concept maps and ontologies.

Example 1.1. StrandMaps

The National Science Digital Library (NSDL) science literacy maps (or StrandMaps) are acyclic, directed graphs, where each vertex (known as educational benchmark) corresponds to a science concept and the edges denote the learning orders (i.e., pre-requisite relationships) between these concepts [1]. NSDL StrandMaps serve the purpose of navigation help and guidance within the NSDL's educational resources.

As the number and complexities of graph structured data increase, making them easier to understand and navigate through are becoming critical tasks. One common approach for simplifying a graph is to partition it into multiple pieces. In a navigation

☆ This work was supported by NSF grants #1043583 — “MiNC: NSDL Middleware for Network- and Context-aware Recommendations” and #1116394 — “RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis”

* Corresponding author. Tel.: +1 480 965 3190; fax: +1 480 965 2751.
E-mail addresses: mijung.kim.1@asu.edu (M. Kim), candan@asu.edu (K.S. Candan).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.11.004

application, for example, the system can present the user the partitions one at a time and the user can move among these partitions using the graph edges that *connect* them.

Clustering has many applications, from general data clustering [47] and document clustering [14,28] to data collection for wireless sensor networks [27] and stock price movement prediction [37] and graph partitioning also has applications in many domains from general purpose clustering of data where objects can be represented as vertices and their dissimilarities can be represented as edges to social network analysis. Common approaches include identifying sets of edges (or *edge-cuts*) or vertices (or *vertex-cuts*) whose removal partitions the graph into the target number of disconnected components:

- *Edge-cut partitioning.* There are many edge-cut based algorithms for partitioning a given graph, including spectral graph partitioning [15,25,29,38,46] and minimum edge-cut based algorithms [7,12,19,20,26,30–32,48]. A common property of almost all these algorithms is that they select (based on different criteria) a set of edges to be removed – or *cut* – from the graph in such a way that the resulting graph is partitioned into multiple connected components. Intuitively, each connected component is an output partition and the edges in the *cut set* that are used to connect pairs of partitions are *bridges* between the corresponding connected components. As shown in Fig. 1(a), an edge that has been cut defines an *exit point* from one partition and an *entry point* to another partition.
- *Vertex-cut partitioning*. While edge-cuts result in partitions that are vertex disjoint, in vertex-cuts [18,9] the data vertices can serve as bridges between the resulting data partitions (Fig. 1(b)); the cut passes through the vertices of the graph (as opposed to the edges as in edge-cut based partitioning) and each vertex in the cut set serve as the *exit*- and the *entry-point*s of the respective partitions.

One fundamental difference between the two is that, as shown in Fig. 2, (while an edge can be cut only one way — thus serving as a bridge between only two partitions), a vertex can be cut in multiple ways and serve as a bridge among more than two partitions. Due to this and other differences (see Section 2.2), while edge-cut and vertex-cut algorithms show some similarities at the surface, the two problems are known to have different characteristics and difficulties [18].

Since a vertex on a minimum vertex-cut is likely to be on many paths (which are cut into two when the vertex is removed), one advantage of the minimum vertex-cut over the minimum edge-cut approach is that vertex-cuts can help identify those vertices of the graph that are well connected with the rest and use those to partition the graph (see Fig. 3(a)). This is especially useful when we use these vertices as bridges between multiple partitions.

1.1. Finding balanced vertex-cuts

A given graph can be cut in different ways and there are different criteria for defining good vertex-cuts: a minimum vertex-cut would partition a given graph into two by cutting the minimum number of vertices [10]. Given a vertex distinguished as a source vertex and another distinguished as a target vertex, a source-target (or s-t) minimum vertex-cut, on the other hand, would look for a minimum sized cut which places the source and target in different partitions.

While minimum vertex-cut and minimum s-t vertex-cut may be applicable in certain application domains, one disadvantage of these is that (as shown in Fig. 3(b)) they can result in significantly unbalanced partitions. Therefore in many applications, an additional "*balance*" criterion is imposed when defining good vertex-cuts [18,9]. The balanced minimum vertex-cut problem is also known as the *vertex separator* problem and is known to be NP-hard [18]. Existing approximation algorithms are able to achieve an approximation ratio of $O\left(\sqrt{log\ opt}\right)$, where opt is the size of an optimal separator, relying on a (semi-definite) quadratic program formulation of the problem, which can be solved in polynomial time [18].

1.2. Contributions of this paper

While approximation algorithms, such as [18], that rely on explicit constraint programming are useful in judging how close we can hope to get to the optimal partitioning solution in polynomial time, they can still be too costly in practice. In this paper, we present a novel graph partitioning heuristic, SBV-Cut, that provides *structurally-balanced* s-t vertex-cuts of a given graph.

More specifically, we define the concept of *balance vertices* and show how to locate and use these balance vertices to obtain a balanced vertex-cut of the graph: Let us be given a directed graph, $G=(V,E)$; we call a vertex, $v \in V$, a *balance vertex* of G if the vertex (a) is similarly distant from the sources and sinks and (b) is similarly connected to them (we will provide a more formal

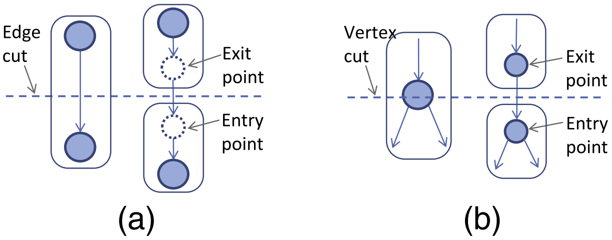


Fig. 1. (a) In an edge-cut based partition, edges serve bridges between partitions; (b) in a vertex-cut based partition, however, the vertices that are cut serve as bridges.

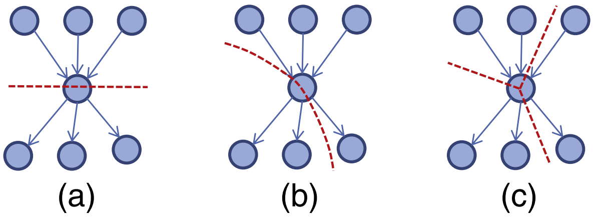


Fig. 2. A vertex can be cut in different ways, including multi-way cuts as in (c).

definition in Section 3). Relying on the observation that a vertex-cut passing through the *balance vertices* will split the input graph into two structurally-balanced partitions, the SBV-Cut algorithm first identifies a set of balance vertices that can be used as a vertex-cut. The graph is then hierarchically partitioned by recursive application of structurally-balanced cuts.

The organization of the paper is as follows:

- We first formulate the problem and introduce quality measures, *expansion* and *modularity*, to assess vertex-cut based graph partitioning solutions (Section 2).
- Next, we introduce the concept of *balance vertices* of a graph and describe how to locate a *structurally balanced vertex-cut* of a graph (Section 3).
- We then present a *vertex-cut based graph partitioning* algorithm called SBV-Cut that leverages these structurally-balancing vertex-cuts (Section 4).
- We run extensive experiments over a wide variety of graphs. The results, reported in Section 5, show that the proposed vertex-cut based partitioning algorithm provides significantly better vertex-cut based *expansion* and *modularity* scores than its competitors and works several orders more efficiently than *constraint-minimization* based approaches.

We conclude the paper in Section 6.

1.3. Related work

We review common graph clustering/partitioning approaches.

1.3.1. Edge-cuts

1.3.1.1. Minimum cut based algorithms. Minimum cut (or maximum flow) techniques [22] are commonly used for partitioning graphs. Flake et al. [21], for example, use a maximum-flow based focused crawler to identify web communities. Flake et al. [20] proposed a minimum cut tree based algorithm where an artificial sink is connected to all vertices in a graph and the minimum cut tree is calculated to find graph clustering using the minimum cut algorithm.

1.3.1.2. Spectral partitioning. Spectral clustering is an alternative approach, where the input graph is partitioned according to its top singular vectors [29]. There are several spectral clustering algorithms. One variant uses the second eigenvector of the Laplacian matrix of the graph for the approximation of the optimal ratio cut partition [25]. According to the spectral graph theory, a generalized eigenvalue problem can be formulated for the minimization of normalized cut and the normalized cut can be used

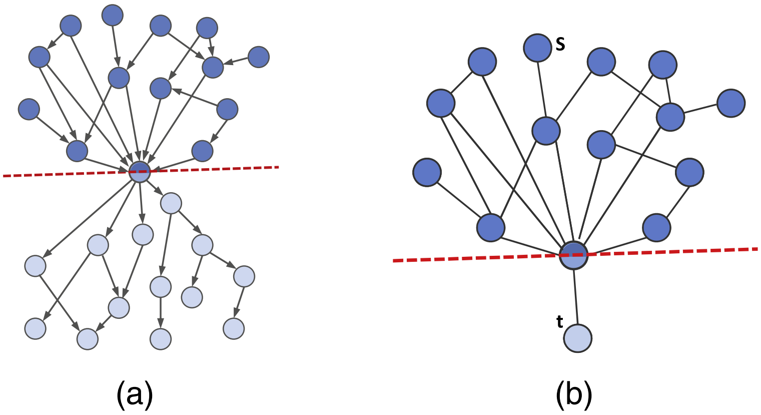


Fig. 3. (a) Minimum vertex-cuts can help partition the graph on vertices that are well connected to the rest; (b) on the other hand, minimum vertex-cuts may also result in very unbalanced partitions.

for graph partitioning [46]. In this paper, we compare our algorithm to spectral clustering algorithm presented in [15]: the algorithm considers the commonly used normalized spectral clustering [42] using an approximation technique in such a way that the algorithm first computes a dense nearest-neighbors matrix of the graph and then identifies a sparse matrix which approximates this dense matrix to apply spectral clustering.

1.3.1.3. Edge-cut based quality criteria. In general, most partitioning algorithms target small inter-partition cuts. In addition, if the resulting partitions have large intra-cluster cuts (i.e., are difficult to further partition), this is taken as a strong evidence of the fact that the located partitions are good. Expansion (or conductance), defined based on these observations, is one of the widely-used criteria [20]. Leskovec et al. [36] introduced network community profile plot to detect communities according to the conductance measure. In [41], an alternative quality function known as modularity to find the best divisions for a network is proposed. Kannan et al. [29] proposed a bi-criteria measure of quality of a clustering based on expansion-like criteria given by two parameters: (a) minimum conductance of the clusters and (b) the ratio of the weight of inter-cluster edges to the total weight of all edges. Shi et al. [46] proposed a global criterion, the normalized cut.

1.3.1.4. Multilevel edge-cut algorithms. Since the graph partitioning problem is NP-complete, there have been many heuristic algorithms studied. A popular example of such algorithms is Kernighan-Lin (KL) algorithm [32], which was improved by Fiduccia–Mattheyses (FM) algorithm [19].

Another class of such heuristics is multilevel algorithm [7,12,26,30,31,48]. Multilevel algorithm is a graph partitioning algorithm that collapses vertices and edges into a smaller graph, partitions it, and then refines the partition during an uncoarsening phase. This local refinement step is performed by different strategies based on KL and the variants of KL [26]. KL refines the partition repeatedly until it reaches at a local minimum which can be far from the best solution. This localized nature of KL can be optimized by incorporating existing search techniques. Such techniques include genetic algorithm [12], simulated annealing [33], tabu search [45], and hybrid heuristics such as Baños et al. [7] that proposed a hybrid of simulated annealing and tabu search. In [48], the local refinement is enhanced with load-balancing for achieving higher quality solutions.

The *k*-way partitioning can be generalized by recursive bisection. `Chaco` [26] and `METIS` [30] provided multilevel KL which is a generalization of the graph bisection algorithm of KL-type. There are also *k*-way partitioning algorithms which refine *k* partitions directly [31,48]. Cong et al. [16] proposed an approach to improve the quality of multiway partitioning solutions by starting with a multiway partition and applying 2-way FM to pairs of blocks. Moraglio et al. [39] also proposed a method for multiway graph partitioning based on a labeling-independent metric.

While a key difference between our algorithm and these algorithms is that we propose a vertex-cut based graph partitioning algorithm as opposed to these algorithms are edge-cut based, it is still possible to compare the quality of the resulting partitions (see Section 5).

In this paper, we compare `SBV-Cut` against `METIS`, `Chaco`, and `MLrMSATS` [7]. `METIS` and `Chaco` are fast. In terms of quality, however, our algorithm outperforms all of the three algorithms (see Section 5).

1.3.2. Vertex-cuts

Common applications of vertex-cut problems include avoiding bottlenecks in communication networks. For example, a small balanced vertex separator [18,9] can be used to balance the workload, while minimizing communication.

Since the vertex-cut problem is in its most general form NP-hard, various approximation algorithms and heuristics have been developed to tackle the problem [18]. Biha et al. [9] provide an exact solution to the vertex separator problem: it represents the underlying problem in terms of constraints and solves the resulting mixed-integer programming. Let $\beta(n)$ be a target positive integer, such that $max\{|A|,|B|\}\leq\beta(n)$ for *A* and *B* that are the resulting two partitions. Biha et al. [9] show that the problem becomes polynomially solvable only when $\beta(n)=n-k$, where *n* is the number of vertices of the graph, for some positive constant *k*. Biha et al. [9] also denote the maximum number of node-disjoint paths between *u* and *v* as α_{uv}.

In addition to $\beta(n)$, the problem specification in [9] also takes as input an α_{min} parameter, which is the lower bound of the cardinality of any separator. In the rest of this paper, we refer to α_{min} and $\beta(n)$ as simply α and β respectively; we also refer to this version of the vertex separator problem as (α,β)-optimal vertex-cut problem.

As we show experimentally in Section 5.3, the input values of α and β have significant impact on the efficiency and effectiveness of the algorithm. Unfortunately, setting these parameters is not trivial; thus one of our goals is to develop a *parameter-free* algorithm. A second disadvantage of the (α,β)-optimal algorithm presented by Biha et al. [9] is that finding a solution can be very time consuming, and thus unpractical, for large data sets. We discuss this in Section 5.3 in detail.

2. Problem formulation

We formulate the problem of vertex-cut based graph partitioning. Before discussing vertex-cuts, however, we first provide the background on common edge-cut based graph partitioning techniques as some of the concepts within the context of edge-cuts will also apply (when suitably adapted) in the context of vertex-cuts.

2.1. Background: edge-cuts and quality

An edge-cut simply is a set of edges whose removal partitions the graph into two.

Definition 2.1. Edge-Cut

Let $G=(V,E)$ be a graph. Let $E' \subseteq E$ be a set of edges such that $G'=(V,E\backslash E')$ is disconnected.

Intuitively, given an edge-cut E', the resulting disconnected components serve as graph partitions and the edges in E' serve as bridges between these partitions. Edge-cut based graph partitioning algorithms search for edge-cuts such that

- given a minimality criterion, the edge-cut E' is minimal and/or
- the resulting graph partitions (or clusters), $C_1, C_2, \ldots, C_m$, satisfy a given optimality criterion (such as cluster diameter, cluster homogeneity and compactness, cluster separation, and cluster integrity).

Some clustering criteria, such as *expansion* [29,20] and *modularity* [41], combine both of the above.

2.1.1. Edge-cut based expansion

Let the edge-cut E' be such that the input graph G is partitioned into two clusters, C_1 and C_2 and let $edge\_cut(C_1,C_2)=|E'|$ denote the number of edges in the edge-cut that separates the vertices in C_1 from the vertices in C_2. The expansion corresponding to the edge-cut, E' is defined as follows [29,20]:

$$expansion_{ecut}(C_1,C_2) = \frac{edge\_cut(C_1,C_2)}{min\{|C_1|,|C_2|\}} \tag{1}$$

where $|C_1|$ and $|C_2|$ denote the number of vertices in the clusters, C_1 and C_2 respectively. Intuitively, the lower the expansion, the smaller is the number of edges needed to separate into the two clusters, relative to the sizes of the clusters. More generally, given an edge-cut E' which partitions $G=(V,E)$ into m clusters $C_1, C_2, \ldots, C_m$, the edge-cut (and the resulting clustering) is thought to be good if

$$expansion_{ecut}\left(E^{'}\right) = \underset{C_i \in \{C_1,C_2,\ldots,C_m\}}{\Theta} expansion_{ecut}(C_i, G-C_i), \tag{2}$$

where Θ is either *average* or *maximum*, is small.

2.1.2. Edge-cut based modularity

The modularity of an edge-cut [41], instead, is defined as

$$modularity_{ecut}\left(E^{'}\right) = \sum_{1\le i\le m}\left(\frac{|E_{i,i}|}{|E|} - \left(\sum_{j\ne i}\frac{\left|E_{i,j}\right|}{|E|}\right)^2\right), \tag{3}$$

where $E_{i,j} \subseteq E'$ is the set of edges in the cut that are used to connect vertices in cluster C_i to those in cluster C_j and $E_{i,i}$ is the set of edges within C_i. Intuitively, the higher the modularity is, the denser is each cluster and the smaller is the fraction of edges that connect different clusters.

2.2. Vertex-cuts and cluster quality

We focus on vertex-cuts for graph partitioning.

Definition 2.2. Vertex-cut

Let $G=(V,E)$ be a graph. Let $V' \subseteq V$ be a set of vertices and E' be the set of edges incident to the vertices in V', such that $G'=(V\backslash V', E\backslash E')$ is disconnected. The set V' of vertices is referred to as a vertex-cut of G.

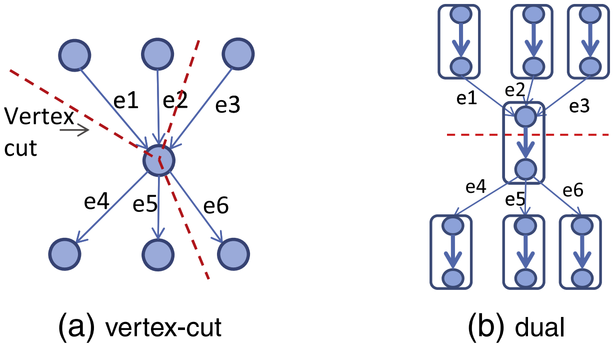


Fig. 4. Dual graph option #1, created by replacing a vertex with two vertices and edge. Note that while a vertex can be cut in multiple ways, this dual graph created can be cut only in one way.

Note that, unlike the case of edge-cuts (where the edges in the edge-cut are removed to obtain the clusters), in vertex-cut base partitioning, the vertices in V' and the corresponding edges in E' are included in the resulting clusters. More specifically, if C_i is a resulting connected component (disconnected from the rest of the graph), C_i is augmented with

- the edges in E' incident to the vertices of C_i and
- the vertices in V' neighboring the vertices of C_i through those edges.

As a consequence, as shown in Fig. 3, the resulting clusters are vertex (and possibly edge) overlapping.

Definition 2.3. s-t vertex cut

Let $G=(V,E)$ be a (connected) directed graph, with a set $S\subseteq V$ of source vertices and a set $T\subseteq V$ of sink vertices. A vertex-cut V' is referred to as a S-T vertex-cut if vertices in S are not reachable from the vertices in T and vice versa after the vertices in the vertex-cut have been removed from G.

Note that, in general, vertex-cut problems are not convertible to edge-cut problems, and vice versa [18]: To see why, consider Figs. 4 and 5: one intuitive way to try to convert the vertex-cut problem to an edge-cut problem is to create a dual graph, where each vertex in the original graph is replaced with two vertices (one accounting for the incoming edges and the other the outgoing ones) and a special edge, and allow only those edge-cuts that include these special edges. As shown in Fig. 4, however, such a solution cannot account for multi-way cuts of vertices, where a vertex is included in more than two partitions.

Alternatively, one could attempt to convert this problem to an edge-cut problem by considering the dual-graph, where each edge in the input graph is represented by a vertex and each vertex in the input graph is represented by a set of edges; as shown in Fig. 5, a vertex-cut in the original graph will correspond to an edge-cut in the transformed graph. However, as the figure shows, a small vertex-cut (in this example with *only 1* vertex) in the original graph may correspond to a large edge-cut (which includes *9* edges) in the dual graph. Since edge-cut based partitioning algorithms (such as [30]) try to minimize the number of edges that are cut, this means that they cannot be used to identify vertex-cut based partitions.

Thus, edge- and vertex-cut problems require different algorithms as well as different partition quality criteria. As we discussed earlier, the size of the vertex-cut is not the only possible quality criterion for vertex-cuts: we also need to consider the sizes of the resulting partitions. There are various definitions of cluster quality applicable in the case of vertex-cuts (including the *cost–benefit ratio* and *sparsity* proposed by Feige et al. [18]). In this paper, we adapt the definitions of expansion and modularity, since their interpretations are well understood in the clustering literature [20,29,36,41]. Unlike most existing measures, such as sparsity [18], which consider only vertex or only edge distributions, our definitions capture both vertex and edge characteristics.

2.2.1. Vertex-cut based expansion

One way to adapt the definition of the expansion to vertex-cuts is to replace the $edge\_cut(C_i,C_j)$ with $vertex\_cut(C_i,C_j)$; i.e., the number of vertices in the vertex-cut through which the two clusters, C_i and C_j are split:

$$expansion_{ncut1}\left(C_i,C_j\right)=\frac{vertex\_cut\left(C_i,C_j\right)}{min\{|C_i|,|C_j|\}}. \tag{4}$$

Note that the above definition does not account for edge distributions in the resulting clusters. An alternative definition which directly accounts for the edge distribution is

$$expansion_{ncut2}\left(C_i,C_j\right)=\frac{vertex\_cut\left(C_i,C_j\right)}{min\{|C_i.E|,|C_j.E|\}}, \tag{5}$$

where $|C_i.E|$ denotes the number of edges in the cluster, C_i. In this case, intuitively, the size of the vertex-cut is normalized relative to the sizes of the clusters in terms of their numbers of edges.

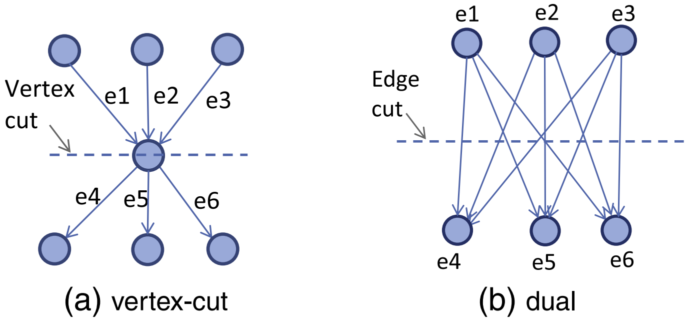


Fig. 5. Dual graph alternative #2: a small vertex-cut in the original may correspond to a large edge-cut in the dual graph.

As before, given a vertex-cut V' which partitions $G=(V,E)$ into m clusters $C_1, C_2, \ldots, C_m$, the vertex-cut is thought to be good if

$$expansion_{ncut}\left(E'\right) = \underset{C_i \in \{C_1, C_2, \ldots, C_m\}}{\Theta} expansion_{ncut}(C_i, G - C_i), \tag{6}$$

where Θ is either *average* or *maximum*, is small.

2.2.2. Vertex-cut based modularity

Similar to the definition of vertex-cut based expansion, vertex modularity can also be defined in two ways, according to whether the number of vertices or the number of edges within a cluster is counted for the first term of the formula. The first definition of vertex-cut based modularity is

$$modularity_{ncut1} = \sum_{1 \le i \le m} \left(\frac{|V_{i,i}|}{|V|} - \left(\sum_{j \ne i} \frac{|V_{i,j}|}{|V|} \right)^2 \right), \tag{7}$$

where $|V_{i,j}|$ is the number of vertices in the graph that exist commonly between clusters C_i and C_j and $|V_{i,i}|$ is the number of vertices in cluster C_i. In the second definition, the first term is modified to consider the edges in the clusters:

$$modularity_{ncut2} = \sum_{1 \le i \le m} \left(\frac{|E_{i,i}|}{|E|} - \left(\sum_{j \ne i} \frac{|V_{i,j}|}{|V|} \right)^2 \right), \tag{8}$$

whereas before $|E_{i,i}|$ denotes the number of edges in C_i. The higher the modularity is, the better is the partitioning.

3. Balance scores of the vertices

Let $G=(V,E)$ be a (connected) directed graph, with a set $S \subseteq V$ of source vertices and a set $T \subseteq V$ of sink vertices. We call a vertex $v \in V$ a *balance vertex* if v is similarly likely to be reached when a random walker proceeds forward from the source vertices in S or backward from the sinks T. Intuitively, v is a vertex where the graph is balanced on both sides in terms of distances to the extremities and connectivity. Given two vertices that are similarly distant from the extremities of the graph, the vertex that is more densely connected to the rest of the graph is said to be the more *dominant* balance vertex. Therefore, we associate a *balance dominance score* to each vertex in the graph by analyzing distances from sources and sinks as well as connectivity within the graph.

To identify these balance scores, in this paper we propose a random walk-based algorithm. Note that, unlike more traditional random-walk based algorithms, such as PageRank [11] and topic distillation [34], the transition probabilities are not only effected by the local properties of the graph, but also its global properties: in particular, since a vertex with a higher balance dominance score should be similarly distant from the sources in S and the sinks in T, the transition probabilities should be *biased* such that the random walker is more likely to move toward vertices that are similarly distant from both sources and the sinks. In this sense, the proposed approach is akin to the approach presented by Candan et al. [13] for identifying how related a given web page is to a set of target pages. Since, in this paper, the bias is needed to represent the connectivity of the vertices to the sources and sinks, we first associate a *minmax-ratio* value that captures the source-and sink-connectivity to each vertex of the graph.

3.1. Minmax-ratios of the vertices

To compute the transition probabilities, we first associate a *minmax-ratio* score to each vertex of the graph:

$$minmax(v) = \frac{max\{asd(S,v), asd(v,T)\}}{min\{asd(S,v), asd(v,T)\}}, \tag{9}$$

where $asd(S,v)$ is the average shortest-path distance from the source vertices in S to v and $asd(v,T)$ is the average shortest-path distance from v to the sink vertices in T. Note that when $minmax(v)=1$, v is equally distant from both S and T along the shortest paths. As the $minmax(v)$ increases, v gets closer to the sources or to the sinks (Fig. 6).

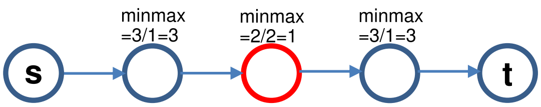


Fig. 6. *Minmax-ratio* scores associated to the vertices of a sample graph; the vertex marked in red, which is equi-distant from s and t has a *minmax-ratio* score of 1, whereas other vertices have higher *minmax-ratio* scores.

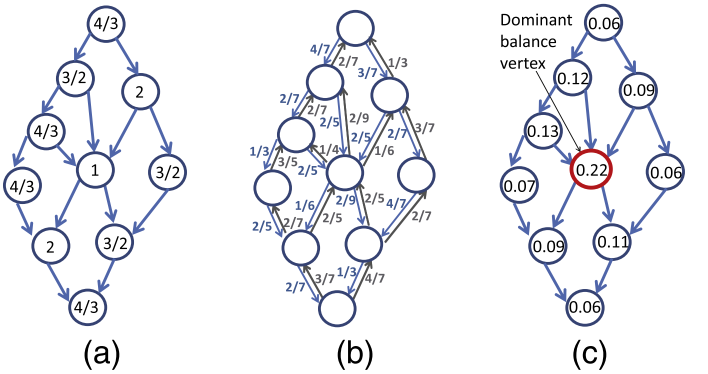


Fig. 7. (a) *Minmax-ratio* values and (b) the corresponding transition probabilities; (c) the balance scores of the vertices in the graph are computed (and the dominant balance vertex identified) using the stationary distribution of the random walk.

3.2. Transition probabilities

Given a graph $G = (V,E)$ and the minmax-ratio values of the vertices of G, we then create a transition matrix $M = (m_{ij})$, where the entry m_{ij} represents the transition probability from vertex v_i to v_j during a random walk. Let $neighbors(v_i)$ be the set of neighbors of v_i (on the undirected version of the graph). Let $v_j \in neighbors(v_i)$ be a neighbor of v_i. Since the transition probability from v_i to all its neighbors needs to add up to 1.0, the *minmax-ratio* score biased transition probability m_{ij} can be computed by solving the following:

$$m_{ij} \times \left(\sum_{v_h \in neighbors(v_i)} \frac{minmax\left(v_j\right)}{minmax(v_h)} \right) = 1.0. \tag{10}$$

Note that since we are trying to locate a point, where both forward and backward traversals are balanced, if there is an edge from v_i to v_j in E, then both m_{ij} and m_{ji} are non-zero. Fig. 7(a) and (b) provide an example.

3.3. Obtaining the balance scores

Once the transition matrix M is computed, the next step is to find the vector $\vec{x} = (x_i)$ that solves the problem $M\,\vec{x} = \vec{x}$ subject to the constraint $\sum_i x_i = 1.0$. In other words, we are looking for the eigenvector of M with the eigenvalue equal to 1; intuitively, each value in x_i describes the stationary probability associated to the vertex $v_i \in V$ (i.e., the portion of the time the random walk spends on vertex v_i). We call the vertex v_i with the highest value x_i, the *dominant balance vertex*. Fig. 7(c) provides an example.

4. SBV-Cut for identifying structurally balanced vertex-cuts

We discuss how to use balance scores of the vertices of a given graph for partitioning it into structurally balanced vertex-cuts. Let $G = (V,E)$ be a graph and $v_i \in V$ be the dominant balance vertex. Intuitively, all the vertices between the source vertices in S and v_i must be on one side of the partition and the vertices between v_i and the sink vertices in T must be on the other side. This,

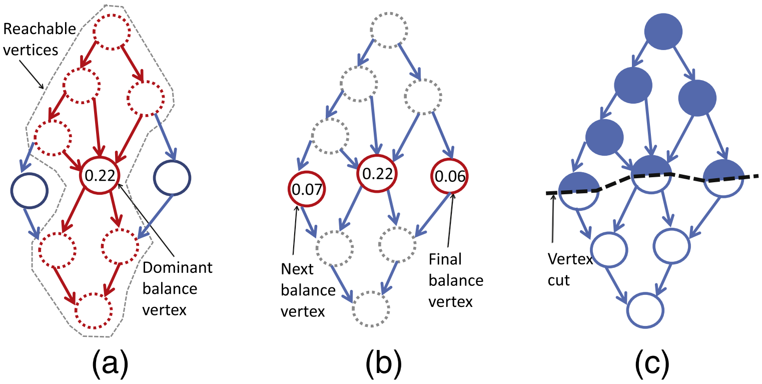


Fig. 8. Outline of the optimistic `SBV-Cut` algorithm: (a) locate the dominant balance vertex, (b) repeatedly identify remaining balance vertices that are not covered by the ones already identified, and (c) partition the graph by including these balance vertices in the vertex-cut.

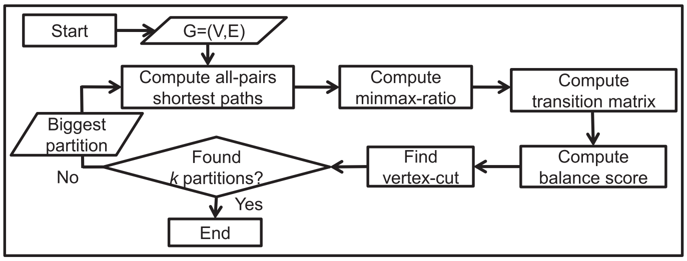


Fig. 9. Flow chart of the optimistic `SBV-Cut` algorithm for acyclic graphs.

however, does not tell us where to place those vertices that are not between the source, the sink, and the dominant balance vertex. Thus, we need to further extend this process.

In the rest of this section, we consider three strategies, *optimistic*, *pessimistic*, and *balance-recomputed* `SBV-Cut` (also referred to as *recomputed*) for bi-partitioning graphs. We first discuss how to optimistically partition acyclic graphs. We will then extend optimistic `SBV-Cut` to cyclic graphs in Section 4.2. In Section 4.3, then, we discuss pessimistic and balance-recomputed strategies.

4.1. Optimistic SBV-Cut on acyclic graphs

The outline of the optimistic `SBV-Cut` algorithm on acyclic graphs is as follows: given an input acyclic directed graph $G=(V,E)$ with the set, S, of source vertices and the set, T, of sink vertices, the algorithm first finds the dominant balance vertex $v_i \in V$. Optimistic `SBV-Cut` then *partially* splits the graph into two as follows (Figs. 8 and 11):

- all the vertices that can reach v_i are considered on one side of the partition,
- all the vertices that are reachable from v_i are considered on the other side of the partition, and
- v_i is included in the set, VC, of vertex-cut.

Note that, source and sink vertices of the graph cannot act as bridges between two partitions; therefore, we do not select them for partitioning the graph in the above step. Once the current dominant balance vertex is selected and the reachable vertices are partitioned into two sets, this leaves those vertices that can neither reach v_i nor are reachable from v_i. Optimistic `SBV-Cut` repeats the above process (until no vertices are left) by selecting the most dominant balance vertex among the remaining vertices and extending the two sides of the partition and the vertex-cut appropriately. *This strategy is optimistic in that it grows the partitions quickly including all reachable vertices in the partitions. The balance scores are computed once at the beginning and never revised.*

As we will see in the Example 4.1 below, once the set of the vertices in the graph has been partitioned into two using the above process, there can remain some edges that cross between the two resulting partitions. Unless eliminated, these edges will become edge-cuts and the result will be a vertex-cut/edge-cut *hybrid* partitioning. In this paper, we do not consider this alternative; instead, we avoid edge-cuts entirely by moving one of the end vertices of each partition-crossing edge into the vertex-cut set, VC. Among the two candidate end-vertices of the crossing edge, the one that is reachable from the balance vertices is included in the vertex-cut. Once there remains no balance vertex to be considered, to complete the bi-partitioning process, each source or sink vertex that has been ignored earlier in the process is attached to the partition to which it is connected by an edge.

In case the target is more than two partitions, the above steps are repeated recursively (each time on the larger remaining partition) to obtain a hierarchical partitioning of the input graph. The flow chart and the pseudo-code of optimistic `SBV-Cut` algorithm are shown in Figs. 9 and 10.

Example 4.1. StrandMap partitioning

Fig. 12 shows how optimistic `SBV-Cut` achieves its partitioning. As Fig. 12(a) shows, the algorithm first partitions the input StrandMap by considering the most dominant balance vertices. The resulting partitioning, however, leaves some edges crossing the two partitions and some sources and sinks unattached to any partition. In a (fast) post-processing step, the algorithm corrects these as shown in Fig. 12(b).

```
Optimistic SBV-Cut (input: acyclic graph $G = (V, E)$; target number of partitions $k$)
Step 1: Compute all-pairs shortest paths
Step 2: Compute minmax-ratio by Equation (9)
Step 3: Compute transition matrix $M$ by Equation (10)
Step 4: Compute balance score through eigen-decomposition of $M$
Step 5: Find a vertex-cut (see Figure 11)
Step 6: Repeat Steps 1 to 5 for the biggest partition found so far until $k$ partitions are found
```

Fig. 10. Pseudo-code of the optimistic `SBV-Cut` algorithm for acyclic graphs.

```
FindVertexCut of optimistic SBV-Cut
(input: acyclic graph $G = (V, E)$; output: a vertex-cut $VC$)
$V_{unreachable} \leftarrow V$
while $V_{unreachable}$ is not empty do {see Figure 8}
    $v_{balance} \leftarrow v$ s.t. $v \in V_{unreachable}$ and $balance\_score(v) = max(balance\_score(V_{unreachable}))$
    $V_{reachable} \leftarrow DFS(v_{balance})$ {$DFS(v)$ is a set of vertices found through Depth First Search from a vertex $v$}
    $V_{unreachable} \leftarrow V_{unreachable} \backslash V_{reachable}$
    $VC \leftarrow VC \cup v_{balance}$
end while
for $v \in VC$ do
    $VC_{reachable} \leftarrow VC_{reachable} \cup DFS(v)$
end for
for $v_{reachable} \in VC_{reachable}$ and $v_{reached} \in V \backslash VC_{reachable}$
  s.t. $v_{reachable} = neighbors(v_{reached})$ and $v_{reachable} \notin VC$ do
    $VC \leftarrow VC \cup v_{reachable}$
end for
```

Fig. 11. Pseudo-code for identifying a vertex-cut of the optimistic SBV-Cut algorithm for acyclic graphs.

4.2. Optimistic SBV-Cut on cyclic graphs

Not all graphs are acyclic. We will extend the basic optimistic SBV-Cut algorithm presented above to handle cyclic graphs.

4.2.1. Cyclic graph partitioning strategy #1

A straight forward extension of the above algorithm to handle graphs with cycles is as follows: given a directed graph $G=(V, E)$ with the set, S, of source vertices and the set, T, of sink vertices, the extended algorithm first finds the dominant balance vertex $v_i \in V$. Optimistic SBV-Cut then *partially* splits the graph into two as follows:

- all the vertices that can reach v_i and not reachable from v_i are considered on one side of the partition,
- all the vertices that are reachable from v_i but cannot reach v_i are considered on the other side of the partition, and
- the vertices that can reach v_i and are also reachable from v_i are included in the set, VC, of vertex-cut.

This leaves those vertices that can neither reach v_i nor are reachable from v_i. Optimistic SBV-Cut repeats the above process (until no vertices are left) by selecting the most dominant balance vertex among the remaining vertices and extending the two sides of the partition and the vertex-cut appropriately. Edge-cuts and unattached sink/source vertices are avoided similarly to the base algorithm.

Note that if a dominant balance vertex is involved in a cycle, the extended optimistic SBV-Cut algorithm described above handles this by including all the vertices that can reach the dominant balance vertex and can also be reached from it in the vertex-cut. However, it is easy to construct graphs where this approach will obviously be disadvantageous. Consider for example a graph where all the vertices in the graph are included in one big cycle; in this case, the optimistic SBV-Cut algorithm will not be able to partition this graph into two.

4.2.2. Cyclic graph partitioning strategy #2

Alternatively, we can extend the optimistic SBV-Cut as follows: given the dominant balance vertex $v_i \in V$,

- all the vertices that can reach v_i and not reachable from v_i without a cycle are considered on one side of the partition,

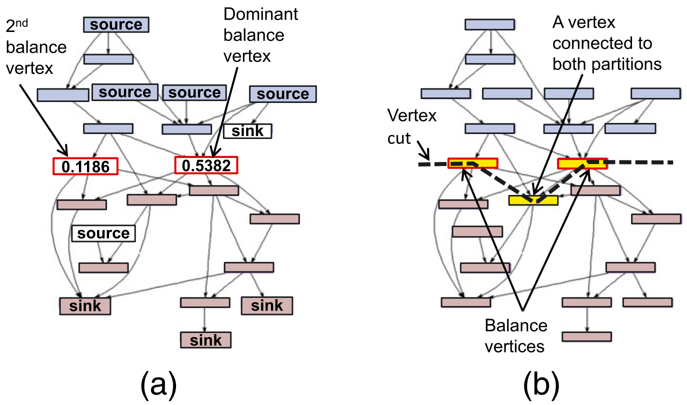


Fig. 12. Example application of optimistic SBV-Cut algorithm on a sample NSDL StrandMap (Map#SMS-MAP-1325, available at http://strandmaps.nsdl.org): (a) the initial partitioning of the graph using dominant balance vertices and (b) post-processing (avoidance of edge-cuts and any unattached source/sink vertices).

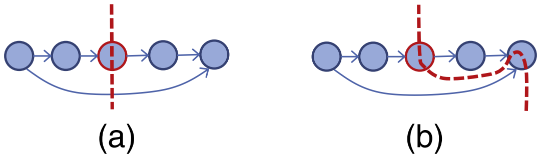


Fig. 13. (a) A vertex-cut splitting a backward edge and (b) an extended vertex-cut including a vertex with a backward bridge.

- all the vertices that are reachable from v_i but cannot reach v_i without a cycle are considered on the other side of the partition, and
- v_i is included in the set, *VC*, of vertex-cut.

This leaves those vertices that can neither reach v_i nor are reachable from v_i. As before, optimistic SBV-Cut repeats this process (until no vertices are left) by selecting the most dominant balance vertex among the remaining vertices and extending the two sides of the partition and the vertex-cut appropriately. Note that in the process some cycles are broken, with one part of the cycle remaining in one partition and the other part in the other partition. These result in backward edge-cuts (Fig. 13). All edge-cuts and unattached sink/source vertices are eliminated as in the base algorithm.

We refer to this second modified algorithm handling cycles as optimistic SBV-Cut_{cycle} and use this as the default algorithm, unless it is specified otherwise.

4.3. Pessimistic and balance-recomputed SBV-Cut algorithms

One potential drawback of the optimistic approach, which aggressively eliminates vertices from consideration, is that the cuts can be highly affected by the initial dominant balance vertex choice. Pessimistic and balance-recomputed SBV-Cut algorithms try to reduce this impact.

4.3.1. Pessimistic SBV-Cut

Instead of removing the vertices on all paths of the selected balance vertices, pessimistic SBV-Cut removes only those vertices that are not on any remaining source-to-sink path (Fig. 14(b)). The algorithm for selecting the vertices to be removed from consideration is shown in Fig. 15. It describes how we remove the vertices on the forward paths of a balance vertex in a DFS (Depth First Search) fashion. Initially the balance vertex is marked as a removed vertex. After that, we traverse all its forward vertices connected with its outgoing edges and mark each one to be removed if all its backward vertices connected with its incoming edges are marked to be removed. Once we mark a vertex as a removed vertex then we repeat this process with its forward vertices. In Fig. 14(b), first we mark the balance vertex (vertex 4) to be removed and we visit one of its forward vertices (say 6) which is also set to be removed since its backward vertex (vertex 4) is already marked to be removed. Next we consider vertex 7. The vertex 7 will not be removed since its backward vertex (vertex 5) is not marked to be removed. The removal of vertices on the backward paths of a balance vertex is similar. Consequently, the impact of the dominant balance vertex is reduced: this approach is pessimistic in the sense that at each step only vertices that structurally depend on the dominant balance vertex are removed and included in the resulting partitions.

4.3.2. Balance-recomputed SBV-Cut

In both optimistic and pessimistic approaches, the balance scores are computed at the very beginning and are re-used throughout the process. The balance scores computed based on the whole graph, however, may not represent the connectivities remaining in the later stages of the algorithm. Therefore, in balance-recomputed SBV-Cut, the balance scores of the remaining vertices are re-computed at each iteration.

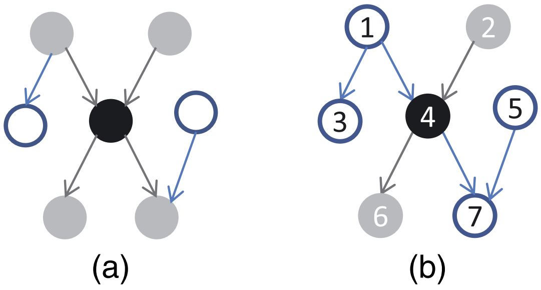


Fig. 14. (a) Optimistic SBV-Cut vs. (b) pessimistic SBV-Cut; vertices in gray denote those that have been removed to be included in partitions due the balance vertex in black. In pessimistic SBV-Cut (b), 1 and 7 cannot be removed since they are on other source-to-sink paths that do not pass on the balance vertex.

```
RemoveForwardPaths of pessimistic SBV-Cut
(input: acyclic graph $G = (V, E)$; a balance vertex $v_{balance}$)
Let $removed$ denote a table of vertices on the paths from $v_{balance}$ to store whether vertices are removed
    (true) or not (false)
Set $removed[v_{balance}]$ = true {balance vertex is removed initially}
for each $v \in V$ such that $(v_{balance}, v) \in E$ do
    $FindForwardVerticestoRemove(V, E, v, removed)$
end for
FindForwardVerticestoRemove($V$,$E$,$v$,$removed$)
if $removed[v]$ = true then
    return
end if
for each $w \in V$ such that $(w, v) \in E$ do
    if $removed[w]$ = false then
        return
    end if
end for
Set $removed[v]$ = true
for each $x \in V$ such that $(v, x) \in E$ do
    $FindForwardVerticestoRemove(V, E, x, removed)$
end for
```

Fig. 15. Pseudo-code of removing forward paths of the pessimistic SBV-Cut algorithm for acyclic graphs.

4.4. Computational complexity

4.4.1. Optimistic strategy

Given a graph $G=(V,E)$, the SBV-Cut algorithm takes $O(|V|)$ time to find sources and sinks. We next need to obtain the shortest path distances between all vertices and all sources and sinks to help compute the *minmax-ratio* values for the vertices of the graph. Given that the sources and sinks can be large and the graph is sparse, instead of computing these distances on a per source/sink basis, we use an all-pairs shortest paths algorithm that gives the shortest path between each pair of vertices. Johnson's shortest path algorithm has a time complexity of $O(|V|log(|V|)+|V||E|)$. Computation of transition probabilities in the next step requires $O(max\_degree|V|)$ time, where max_degree is the maximum degree of any vertex in the graph, since for each vertex in the graph, we need to consider all its incoming and outgoing edges to obtain the transition probabilities. The complexity of the eigen-decomposition step depends on the algorithm that is used; in our implementation we leverage Matlab's *eigs* function, which is based on ARPACK and uses an iterative power method to identify eigenvalues. The cost of this algorithm depends on the number of iterations needed for the process to converge on the eigenvalues. Finally, partitioning the vertices around the balance vertices requires DFS (Depth First Search) to identify reachabilities, which requires $O(|V|+|E|)$ time.

4.4.2. Pessimistic strategy

At each iteration, the pessimistic approach needs to identify those vertices for which all source-to-sink paths have been eliminated by the removal of the most recently selected balance vertex. The complexity of the algorithm now increases with a per iteration $O(|E|)$ factor since we traverse from the balance vertex through its forward and backward vertices (i.e., in the worst case, we traverse $O(|E|)$ edges). For the tail of each edge, we do a constant time table look up to decide whether to remove the vertex.

Table 1
Data sets used in the experiments.

(Acyclic, small) data sets	Avg. vertices	Avg. edges
20 StrandMaps from [1]	20.25	38.7

	(Acyclic and cyclic, large) data sets	Vertices	Edges	Cycles	# of cycles	Avg. cycle length
D1	Arxiv GR-QC (General Relativity and Quantum Cosmology) collaboration network [35]	5242	28,980	Acyclic	–	–
D2	Online Dictionary of Library and Information Science [43]	2899	16,376	Cyclic	5672	220.85
D3	Political blogs [6]	1222	16,714	Cyclic	7548	141.76
D4	E-mail network URV [24]	1113	5451	Acyclic	–	–
D5	Roget's Thesaurus, 1879 [44]	994	3640	Cyclic	1106	140.66
D6	*C. elegans* metabolic network [17]	453	2025	Acyclic	–	–
D7	North American Transportation Atlas Data [8]	332	2126	Acyclic	–	–
D8	Neural network [49]	297	2148	Cyclic	711	43.74
D9	Jazz musicians network [23]	198	2742	Acyclic	–	–
D10	Word adjacencies [40]	112	425	Acyclic	–	–

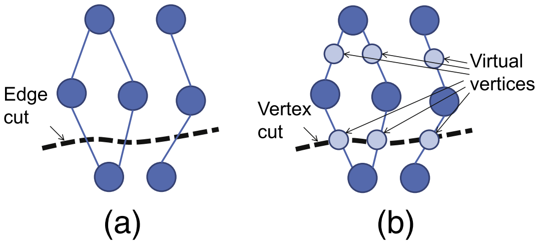


Fig. 16. Converting an edge-cut into a vertex-cut by introducing virtual vertices on each edge.

4.4.3. Balance-recomputation strategy

The balance-recomputed SBV-Cut further recomputes the balance scores of all vertices in each iteration of the process. This requires execution of the three following steps on a per-iteration basis: computation of all-pairs shortest paths ($O(|V|log(|V|)+|V||E|)$), computation of transition probabilities ($O(max\_degree|V|)$), and eigen-decomposition.

5. Evaluation

We evaluate SBV-Cut on graphs of different shapes and sizes and compare the results to (α,β)-optimal *vertex-cut* [9] as well as *edge-cut* based multilevel algorithms (METIS, Chaco, and MLrMSATS) and spectral clustering [15]. The graphs that we use for evaluation include the (acyclic) NSDL Science Literacy Maps [1] considered in Examples 1.1 and 4.1 and larger (some cyclic) data sets listed in Table 1.

For METIS, Chaco, and MLrMSATS, we used authors' own packages [2–4]. We solved the mixed integer programming for (α,β)-optimal vertex-cut [9] using GNU Linear Programming Kit (GLPK) [5]. Other algorithms were implemented using Matlab version 7.9.0.529. All experiments except Chaco and MLrMSATS were run on a Windows XP machine with Intel(R) Core(TM)2 Duo 2.33 GHz CPU and 2 GB memory. For all experiments, the upper bound on the bi-partitioning time was set to 1800 s and cases requiring more time were marked *unsuccessful*.

5.1. Evaluation criteria

We evaluate SBV-Cut and compare it to alternative algorithms based on *expansion* and *modularity* (which take into account the properties of the cuts as well as the resulting clusters), and the execution time.

One major difficulty in comparing the vertex-cut based SBV-Cut to existing edge-cut based graph clustering algorithms, such as METIS, Chaco, and spectral clustering, is that, as discussed in Section 2, vertex-cuts can be evaluated using vertex-cut based expansion and modularity measures ($expansion_{ncut1}$, $expansion_{ncut2}$, $modularity_{ncut1}$, and $modularity_{ncut2}$), whereas edge-cut based algorithms require edge-cut based measures ($expansion_{ecut}$ and $modularity_{ecut}$). We overcome this difficulty by converting edge-cuts to vertex-cuts and vice versa:

- we convert a vertex-cut that SBV-Cut returns into an edge-cut and use edge-cut based measures to compare SBV-Cut to existing edge-cut based algorithms;
- we also convert an edge-cut returned by an existing edge-cut algorithm into a vertex-cut and use vertex-cut based measures to compare existing edge-cut based algorithms to SBV-Cut.

5.1.1. Converting an edge-cut to a vertex-cut

As shown in Fig. 16, an edge-cut can be converted into a vertex-cut simply by introducing *virtual vertices* on every edge of the input graph. After this transformation, the edge-cut of the original graph will correspond to a vertex-cut of the transformed graph.

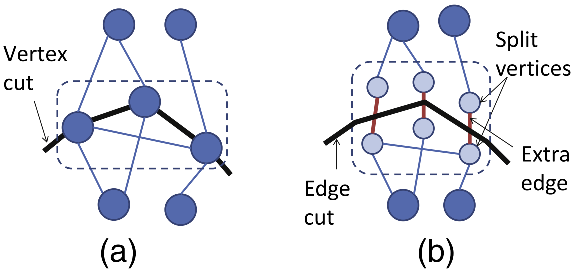


Fig. 17. Converting a vertex-cut into an edge-cut by splitting each vertex on the vertex-cut into multiple vertices and inserting an edge between them.

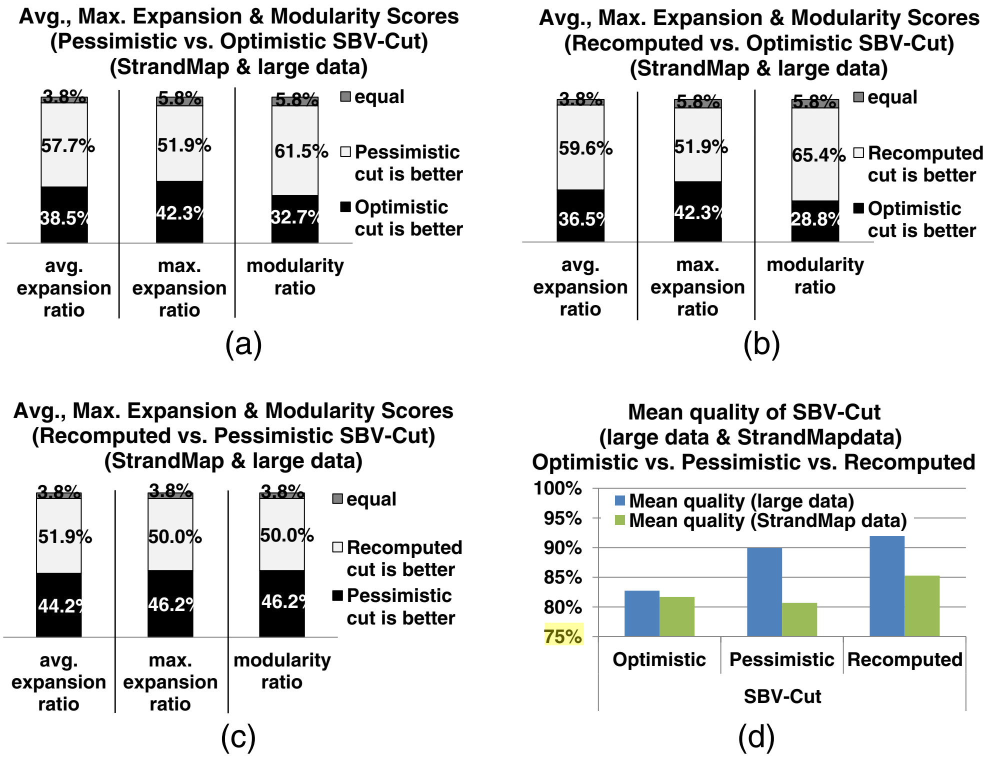


Fig. 18. (a) Pessimistic vs. optimistic SBV-Cut, (b) balance-recomputed vs. optimistic SBV-Cut, (c) balance-recomputed vs. pessimistic SBV-Cut, and (d) mean relative quality scores SBV-Cut strategies.

In the experiments, we refer to the vertex-cut based expansion and modularity values computed on this graph as $expansion^*_{ncut1}$, $expansion^*_{ncut2}$, $modularity^*_{ncut1}$, and $modularity^*_{ncut2}$.

5.1.2. Converting a vertex-cut to an edge-cut

In order to convert vertex-cuts to edge-cuts, we modify the graph such that vertex-cuts correspond to edge-cuts (Fig. 17). More specifically, the original graph is extended such that each vertex shared by more than one partition is represented by multiple vertices, one for each resulting partition. These vertices are then connected to each other with new edges. The edge-cut based expansion and modularity are measured on this *extended graph*. In the experiments, we refer to the edge-cut based expansion and modularity values computed on this graph as $expansion^+_{ecut}$ and $modularity^+_{ecut}$.

Note that, we have various alternative definitions of expansion and modularity measures. Therefore, to simplify the visualization and enable observation of general trends, (*unless otherwise specified*) we use average quality scores, obtained by averaging the score for all relevant data sets and target partition numbers.

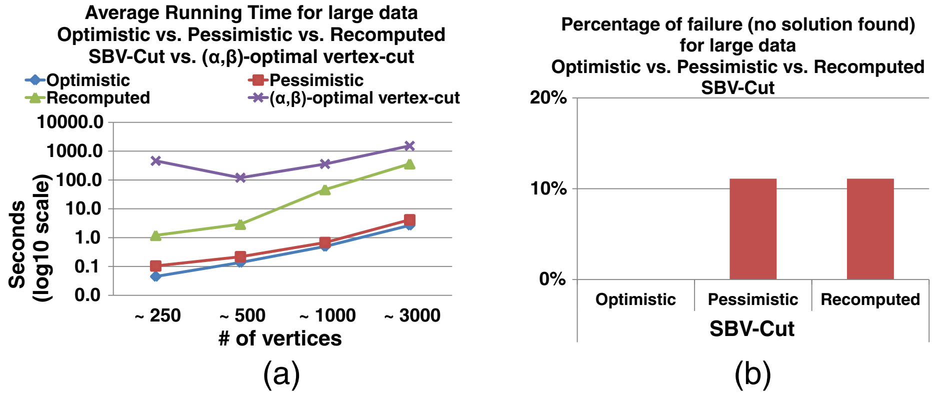


Fig. 19. (a) Running time and (b) percentage of failure (with an 1800 s upper bound imposed).

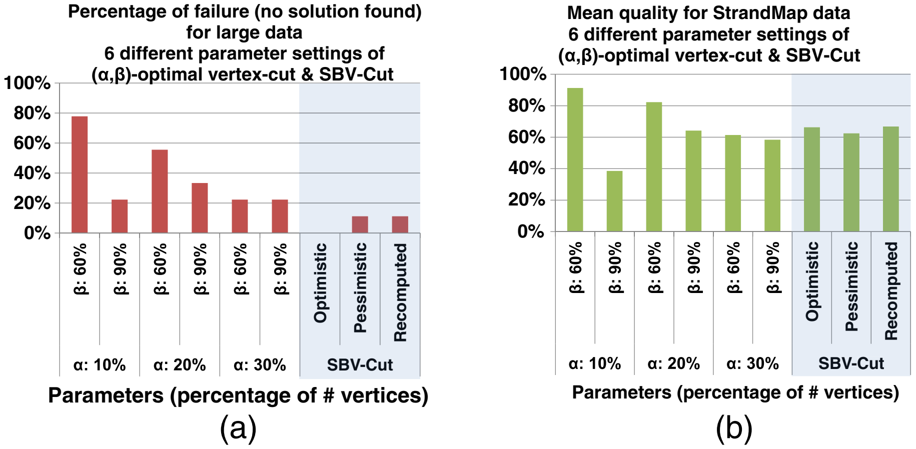


Fig. 20. (a) Failure rates for different parameter settings (large data; 1800 s limit) and (b) mean relative quality scores (StrandMap data) for (α,β)-optimal vertex-cut and three algorithms of SBV-Cut.

5.2. Optimistic vs. pessimistic vs. balance-recomputed SBV-Cut

First, we examine the impacts of different SBV-Cut strategies in Section 4.3. Fig. 18(a) compares the ratio of the cases in which optimistic strategy provides a better expansion or modularity performance than the pessimistic strategy and vice versa. As the figure shows, as one would expect, the pessimistic strategy (which is less aggressive) performs better than the optimistic strategy both in terms of expansion and modularity. Similarly, as shown in Fig. 18(b), the recomputation based strategy also outperforms the optimistic strategy. An interesting result is observed, however, when the performances of pessimistic and recomputation strategies are compared: as shown in Fig. 18(c), while recomputation is better in general, the pessimistic strategy is nevertheless highly competitive.

These results are studied in more detail in Fig. 18(d), which compares the *mean relative quality* scores for each of the three SBV-Cut strategies: here, *X*% means that the partitions based on a given strategy is, on the average, *X*% as good as the best of

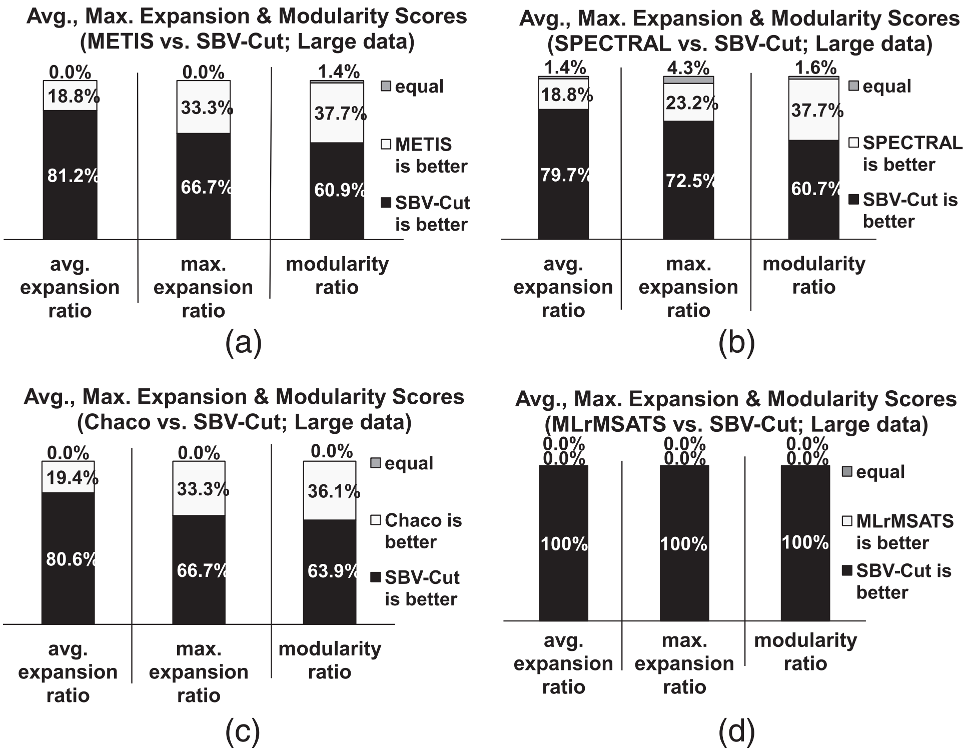


Fig. 21. Percentage of cases where SBV-Cut is better in large graphs: (a) vs. METIS, (b) vs. spectral, (c) vs. Chaco, and (d) vs. MLrMSATS.

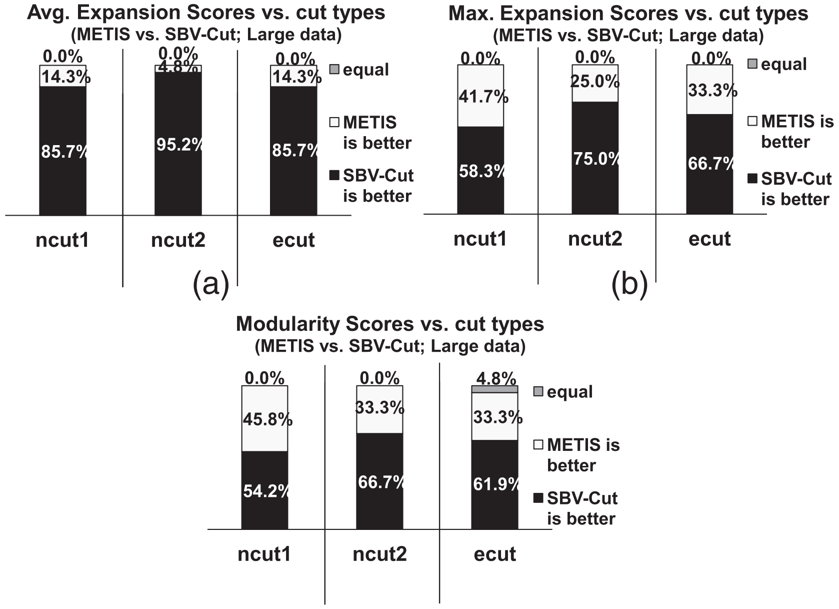


Fig. 22. Impact of cut types, $expansion^*_{ncut1}$, $expansion^*_{ncut2}$ and $expansion^+_{ecut}$ on large graphs.

the three strategies. As shown in this figure, the comparison of mean quality of three approaches overall agrees with the results in Fig. 18(a), (b) and (c) except that for StrandMap data, the pessimistic SBV-Cut is slightly lower mean quality than the optimistic one (81% vs. 82% respectively), which is largely due to a single case where the relative score of the pessimistic SBV-Cut is extremely lower (27%) than the score of the optimistic one (100%).

Fig. 19(a) shows that running time for all three SBV-Cut algorithms (as well as the (α,β)-optimal strategies discussed in the next subsection). As described earlier, we imposed a time limit of 1800 s and marked all runs beyond this as *unsuccessful*. Fig. 19a shows that the running time of the pessimistic strategy is at least an order faster than the recomputation strategy and is close to the optimistic one. Fig. 19(b) shows that, while the optimistic strategy completed under 1800 s for all cases, pessimistic and recomputation based strategies both failed only in 1 case out of 9. When these results are considered along with the quality results in Fig. 18 the pessimistic strategy emerges as the most advantageous of the three.

5.3. (α,β)-optimal Vertex-Cut vs. SBV-Cut

We compare SBV-Cut with the (α,β)-optimal vertex-cut [9]. Unlike our SBV-Cut which is parameter-free, Biha et al. [9] require as input two parameters: the lower bound (α) on the size of the cut and the upper bound (β) on the maximum number of vertices. In Fig. 20, we consider 6 different settings for the (α,β) pair and compare the mean relative quality results to SBV-Cut strategies (in the figure each parameter value is set as a percentage of the total number of vertices).

First of all, as shown in Fig. 20(a), the non-completion rate of (α,β)-optimal approach is extremely high for the large data set, especially for strict α and β parameters. This is also confirmed by Fig. 19(a) which shows that (α,β)-optimal approach can be multiple orders slow that optimistic and pessimistic SBV-Cut strategies. As shown in Fig. 20(b), the qualities of the resulting

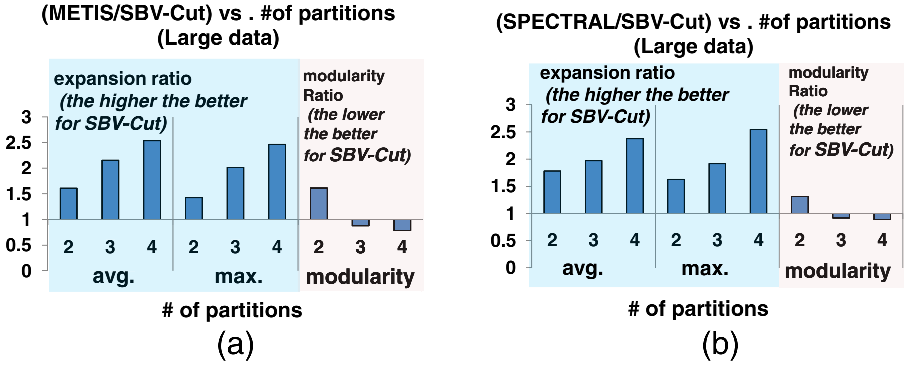


Fig. 23. Expansion and modularity ($\frac{METIS\ \ score}{SBV-Cut\ \ score}$, $\frac{spectral\ score}{SBV-Cut\ \ score}$) ratios for large graphs (expansion ratio > 1 and modularity ratio < 1 indicate that SBV-Cut is better).

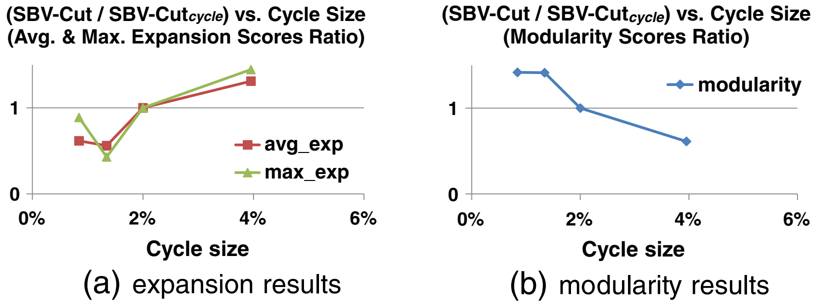


Fig. 24. SBV-Cut vs. SBV-Cut$_{cycle}$ for varying cycle lengths (relative to the # of edges in the graph).

partitions depend on the (α,β) parameter settings; while as expected there exist settings that can beat SBV-Cut, the gains come with multiple orders of increase in the average execution time (as was shown in Figs. 19(a) and 20(a)).

5.4. Edge-cuts vs. SBV-Cut

Here we compare SBV-Cut to four different edge-cut techniques: METIS [30], spectral clustering [15], Chaco [26], and MLrMSATS [7] using large data sets in Table 1. Note that on small data sets (StrandMaps), SBV-Cut performed better against these algorithms in terms of both partition quality and the execution time. We omit these results on small data sets and focus on larger graphs.

5.4.1. Results on large graphs

5.4.1.1. Overview. Fig. 21 compares the expansion and modularity behaviors of SBV-Cut algorithm against those of METIS, spectral clustering, Chaco, and MLrMSATS and shows that SBV-Cut outperforms these four algorithms. Notably, as shown in Fig. 21(d), SBV-Cut is better than MLrMSATS for all the cases. More specifically, in terms of average expansion and modularity scores, the comparisons of SBV-Cut vs. MLrMSATS come to 0.1914 vs. 0.6068 for average expansion and 0.2552 vs. −0.1179 for average modularity.

Next we do further analysis for cut-based scores and relative scores using METIS and spectral clustering.

5.4.1.2. Cut types. Note that results in Fig. 21 include both vertex-cut and edge-cut based scores. Fig. 22 shows the results for different types of cuts. As seen in the figure, for all types of measures (*including* edge-cut based measures), SBV-Cut outperforms METIS and spectral clustering. The difference is especially strong in average expansions.

5.4.1.3. Relative scores. Fig. 23 plots the average $\frac{METIS\_score}{\text{SBVcut score}}$ and $\frac{\text{spectral score}}{\text{SBVcut score}}$ for varying number of target partitions.

As the number of target partitions increases, the scores of SBV-Cut tend to become increasingly better than those of METIS and spectral clustering. Intuitively, in the initial bi-partitioning, some of outlying seed vertices can make the dominant balance vertex relatively unbalanced with respect to the whole graph. In fact, we can see that in terms of relative scores SBV-Cut is especially critical: for example, according to Fig. 23, METIS returns expansion scores up to 2.5× *worse* than SBV-Cut.

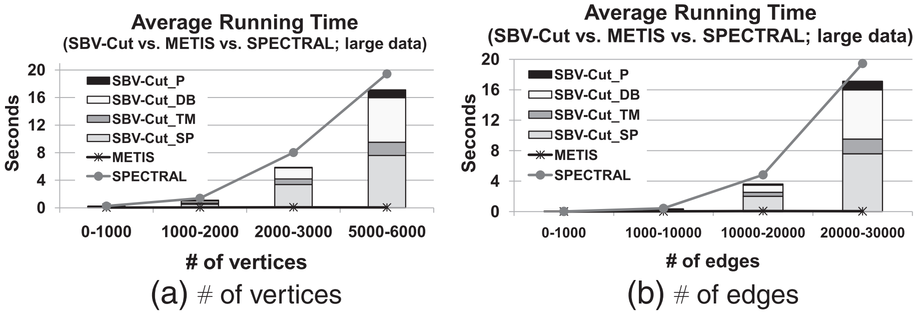


Fig. 25. Average bi-partitioning times of different algorithms for large graphs. The running time of SBV-Cut is divided into four parts, SBV-Cut_P, partitioning based on dominant balance vertices; SBV-Cut_DB, finding dominant balance vertices; SBV-Cut_TM, computation of transition matrix; and SBV-Cut_SP, computation of all-pairs shortest paths.

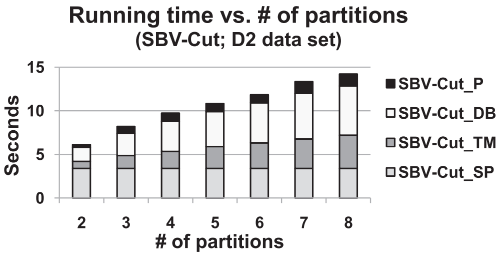


Fig. 26. The running time of SBV-Cut for the D2 data set vs. # of partitions.

5.4.1.4. Cycle size. Some of the graphs in the large graph data set contain cycles. In Section 4.2, we have seen that large cycles can degenerate the SBV-Cut clustering quality and, thus, we proposed a slight modification (SBV-Cut$_{cycle}$) which can improve the quality when graphs contain large cycles. Fig. 24 compares SBV-Cut and SBV-Cut$_{cycle}$ for varying degrees of average cycle length (in terms of the number of edges on a cycle as a function of the number of edges in the whole graph). As this figure shows, when the cycles are short (containing <2% of the edges in the graph), the basic SBV-Cut algorithm performs better than SBV-Cut$_{cycle}$; on the other hand, as the cycles become longer (>2%), the SBV-Cut$_{cycle}$ algorithm provides better scores than the basic SBV-Cut.

5.4.1.5. Execution time. Fig. 25 compares the running times for the large graphs data set. As can be seen here, on larger graphs, METIS significantly outperforms both spectral clustering and SBV-Cut in terms of execution times. However, as discussed above, this execution time advantage comes with a significant penalty in terms of qualities of the resulting partitions (especially as the number of resulting partitions increase). Once again, bi-partitioning times of spectral and SBV-Cut algorithms behave similarly.

Unlike METIS and spectral clustering,[1] SBV-Cut is a hierarchical clustering algorithm; therefore, the execution time increases as the number of target partitions increases. Fig. 26 shows that the execution time of SBV-Cut increases roughly linearly with the number of target partitions (except that the initial all-pairs shortest paths step does not need to be repeated for subsequent partitioning).

In Fig. 25, we compare the execution times of SBV-Cut, METIS, and spectral clustering. We omit Chaco and MLrMSATS since they are implemented for a Linux machine. In the experiments, Chaco is almost as fast as METIS but MLrMSATS runs even slower than spectral clustering which is slower than SBV-Cut.

6. Conclusions

In this paper, we presented a vertex-cut based graph partitioning algorithm, *structural balance vertices* (SBV)-Cut. SBV-Cut searches for vertices where the graph is balanced in terms of distances to the extremities (sources and sinks) as well as its connectivity to the rest and cuts the graph incrementally along these *dominant balance vertices*. Experimental results show that SBV-Cut performs couple of orders more efficiently than and almost as effectively as the existing vertex-cut based algorithms. The results also show that SBV-Cut performs very well in terms of expansion and modularity (especially in terms of vertex-cut based measures) when compared to more traditional edge-cut based clustering algorithms, including multilevel algorithms, METIS, Chaco, and MLrMSATS and spectral clustering.

Our proposed algorithm shares the relatively high computational complexity of partitioning algorithms that take into account the spectral characteristics of the underlying graph and this complexity makes the algorithm slower than METIS and similar heuristics. As a result, web-scale graphs would require suitable heuristics and index structures to pre-compute (approximate) shortest-path distances and the random walk for obtaining balance scores are maintained incrementally as in search engines implementations of PageRank style web page scores. The remainder of the algorithm is highly efficient if these scores can be maintained off-line.

We would like to highlight that there are many applications, where a graph with 10s to 1000s of nodes needs to be partitioned off-line (for summarization, for indexing, or for parallel processing) and since the quality of the resulting partitions will impact the quality and/or efficiency of the applications, the users' trade-off between time and quality tips toward quality. Experimental results show that the proposed algorithm is especially suitable for these scenarios even without pre-computation of distances or balance scores (see Section 5).

For future work, in order to improve the time complexity of our algorithm, we will focus on heuristics for finding dominant balance vertices without using eigen decomposition which can be a bottleneck in running time and detecting *k* balance vertices directly rather than generalizing bisection.

References

[1] http://strandmaps.nsdl.org/.
[2] http://glaros.dtc.umn.edu/gkhome/views/metis.
[3] http://www.sandia.gov/bahendr/chaco.html.
[4] http://www.ace.ual.es/cgil/grafos/Graph-Partitioning.html.

[1] We have also experimented with hierarchical versions of METIS and spectral clustering. Since the resulting clusters are not better in terms of modularity and expansion, we do not report those results in this paper.

[5] http://www.gnu.org/software/glpk/.

[6] L. Adamic, N. Glance, The political blogosphere and the 2004 US Election, Proc. of the WWW2005 Workshop on the Weblogging Ecosystem, 2005.

[7] R. Baños, C. Gil, J. Ortega, F. Montoya, Multilevel heuristic algorithm for graph partitioning, 3rd EvoCOP, LNCS, Essex (UK), 2003, Published by.

[8] V. Batagelj, A. Mrvar, Pajek datasets, http://vlado.fmf.uni-lj.si/pub/networks/data/2006.

[9] M. Biha, M. Meurs, An exact algorithm for solving the vertex separator problem, Journal of Global Optimization (2010) 1–10.

[10] P. Black, Minimum vertex cut, Dict. of Algorithms and Data Structures [online], U.S. NIST, 2004.

[11] S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems 30 (1–7) (1998) 107–117.

[12] T. Bui, B. Moon, Genetic algorithm and graph partitioning, IEEE Transactions on Computers 45 (1996) 841–855.

[13] K. Candan, W. Li, Reasoning for web document associations and its applications in site map construction, Data & Knowledge Engineering 43 (2) (2002).

[14] C.-L. Chen, F.S.C. Tseng, T. Liang, Editorial: An integration of WordNet and fuzzy association rule mining for multi-label document clustering, Data & Knowledge Engineering 69 (11) (2010).

[15] W. Chen, Y. Song, H. Bai, C. Lin, E. Chang, Parallel spectral clustering in distributed systems, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (2010) 3.

[16] J. Cong, S.K. Lim, Multiway partitioning with pairwise movement, ICCAD, 1998, pp. 512–516.

[17] J. Duch, A. Arenas, Community detection in complex networks using extremal optimization, Physical Review E 72 (2005) 027104.

[18] U. Feige, M. Hajiaghayi, J. Lee, Improved approximation algorithms for minimum-weight vertex separators, Proc. of STOC, 2005, pp. 563–572.

[19] C. Fiduccia, R. Mattheyses, A linear time heuristic for improving network partitions, Proc. 19th IEEE DAC, 1982, pp. 175–181.

[20] G. Flake, R. Tarjan, K. Tsioutsiouliklis, Graph clustering and minimum cut trees, Internet Mathematics 1 (4) (2004).

[21] G. Flake, S. Lawrence, C. Giles, Efficient identification of web communities, KDD, 2000, pp. 150–160.

[22] L. Ford Jr., D. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.

[23] P. Gleiser, L. Danon, Jazz musicians network, Advances in Complex Systems 6 (2003) 565.

[24] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, A. Arenas, Self-similar community structure in a network of human interactions, Physical Review E 68 (2003) 065103.

[25] L. Hagen, A. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Transaction on Computer-Aided Design 11 (9) (1992) 1074–1085.

[26] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, Proc. Supercomputing, 1995.

[27] H. Jiang, S. Jin, C. Wang, Prediction or not? An energy-efficient framework for clustering-based data collection in wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 22 (6) (2011) 1064–1071.

[28] A. Kalogeratos, A. Likas, Document clustering using synthetic cluster prototypes, Data & Knowledge Engineering 70 (3) (2011).

[29] R. Kannan, S. Vempala, A. Vetta, On clusterings — good, bad and spectral, FOCS, 2000, pp. 367–377.

[30] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1998) 359–392.

[31] G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs, Journal of Parallel and Distributed Computing 48 (1) (1998) 96–129.

[32] B. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphics, The Bell System Technical Journal (1970) 291–307.

[33] S. Kirkpatrick, C. Gelatt Jr., M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[34] J. Kleinberg, Authoritative sources in a hyperlinked environment, Proc. of SODA, 1998, pp. 668–677.

[35] J. Leskovec, J. Kleinberg, C. Faloutsos, Graph evolution: densification and shrinking diameters, ACM Transactions on Knowledge Discovery from Data 1 (1) (2007).

[36] J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney, Statistical properties of community structure in large social and information networks, Proc. of WWW, 2008.

[37] M.-C. Lin, A.J.T. Lee, R.-T. Kao, K.-T. Chen, Stock price movement prediction using representative prototypes of financial reports, ACM Transactions on Management Information Systems 2 (3) (2008).

[38] J.G. Martin, E.R. Canfield, Ranks and representations for spectral graph bisection, SIAM Journal on Scientific Computing 31 (5) (2009) 3529–3546.

[39] A. Moraglio, Y.-H. Kim, Y. Yoon, B. Moon, Geometric crossovers for multiway graph partitioning, Evolutionary Computation 15 (4) (2007) 445–474.

[40] M. Newman, Finding community structure in networks using the eigenvectors of matrices, Physical Review E 74 (2006) 036104.

[41] M. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69 (2004) 026113.

[42] A. Ng, M. Jordan, Y. Weiss, On Spectral Clustering: Analysis and an Algorithm, NIPS, 2001, pp. 849–856.

[43] J. Reitz, ODLIS: Online Dictionary of Library and Information Science, 2002.

[44] P. Roget, Roget's Thesaurus of English Words and Phrases, 1879.

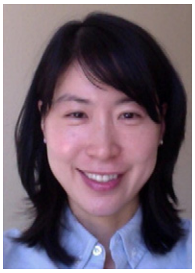
[45] E. Rolland, H. Pirkul, F. Glover, A tabu search for graph partitioning, Annals of Operations Research 63 (1996).

[46] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8) (2000) 888–905.

[47] Y. Song, S. Jin, J. Shen, A unique property of single-link distance and its application in data clustering, Data & Knowledge Engineering 70 (11) (2011) 984–1003.

[48] C. Walshaw, M. Cross, Mesh partitioning. A multilevel balancing and refinement algorithm, SIAM Journal on Scientific Computing 22 (1) (2000) 63–80.

[49] D. Watts, S. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442.



Mijung Kim is a Ph.D. student of Computer Science and Engineering at the School of Computing, Informatics, and Decision Science Engineering at Arizona State University since Jan 2010. Her main research interests include data management, data mining, information and multimedia retrieval, WWW, and database systems.

K. Selcuk Candan is a Professor of Computer Science and Engineering at the School of Computing, Informatics, and Decision Science Engineering at the Arizona State University. He joined the department in August 1997, after receiving his Ph.D. from the Computer Science Department at the University of Maryland at College Park. Prof. Candan's primary research interest is in the area of management of non-traditional, heterogeneous, and imprecise (such as multimedia, web, and scientific) data. His various research projects in this domain are funded by diverse sources, including the National Science Foundation, Department of Defense, Mellon Foundation, and DES/RSA (Rehabilitation Services Administration). He has published over 140 articles and many book chapters. He has also authored 9 patents. Recently, he co-authored a book titled "Data Management for Multimedia Retrieval" for the Cambridge University Press. Prof. Candan served as an editorial board member of one of the most respected database journals, the Very Large Databases (VLDB) journal. He is also in the editorial board of the Journal of Multimedia. He has served in the organization and program committees of various conferences. In 2006, he served as an organization committee member for SIGMOD'06, the flagship database conference of the ACM and one of the best conferences in the area of management of data. In 2008, he served as a PC Chair for another leading, flagship conference of the ACM, this time focusing on multimedia research (MM'08). More recently, he served as a program committee group leader for ACM SIGMOD'10. In 2011, he will serve as a general co-chair for the ACM MM'11 conference. In 2012 he will serve as a general co-chair for ACM SIGMOD'12.