Linear-time Hierarchical Community Detection

Ryan A. Rossi
Adobe Research

Nesreen K. Ahmed
Intel Labs

Eunyee Koh
Adobe Research

Sungchul Kim
Adobe Research

ABSTRACT

Community detection in graphs has many important and fundamental applications including in distributed systems, compression, image segmentation, divide-and-conquer graph algorithms such as nested dissection, document and word clustering, circuit design, among many others. Finding these densely connected regions of graphs remains an important and challenging problem. Most work has focused on scaling up existing methods to handle large graphs. These methods often partition the graph into two or more communities. In this work, we focus on the problem of *hierarchical community detection* (*i.e.*, finding a hierarchy of dense community structures going from the lowest granularity to the largest) and describe an approach that runs in linear time with respect to the number of edges and thus fast and efficient for large-scale networks. The experiments demonstrate the effectiveness of the approach quantitatively. Finally, we show an application of it for visualizing large networks with hundreds of thousands of nodes/links.

KEYWORDS

Community detection, hierarchical communities, linear-time algorithms, label propagation, graph clustering, graph mining

1 INTRODUCTION

Communities of a graph are sets of nodes that are densely connected and close to one another in the graph [7]. Communities are important for understanding complex systems modeled as graphs [7, 16]. In our modern age of big data, it has become increasingly important to study and understand complex systems that arise from large data of diversely interconnected entities such as biological networks [1], social networks [9], citation networks [8], among many others. Community detection in graphs has been one of the most fundamental tools for analyzing and understanding the components of complex networks and has been used for many real-world applications. It has been used extensively in applications to distributed systems [10, 19, 20], compression [3, 15], image segmentation [6, 17], document and word clustering [5], among others.

Communities are sets of vertices $C_1, \ldots, C_k$ such that each set C_k has with more connections inside the set than outside [7]. While there are many different methods for finding communities [7, 16], it is generally agreed that a community $C_k \subseteq V$ is "good" if the induced subgraph is dense (*e.g.*, many edges between the vertices in C_k) and there are relatively few edges from C_k to other vertices $\bar{C}_k = V \setminus C_k$ [16]. Let $E(C_k)$ denote the set of edges between vertices in C_k (internal edges) and $E(C_k, \bar{C}_k)$ be the set of all edges between C_k and $\bar{C}_k$ (external edges). Another desired property of a community C_k is that vertices in C_k are all close to one another, *i.e.*, the distance between any two vertices $v, w \in C_k$ denoted as $\texttt{dist}(v, w)$ is as small as possible (small proximity, distance). Community detection aims to cut a graph into two or more sparsely interconnected dense subgraphs [7]. Semantically, these subgraphs may represent a tightly-knit group of friends, a household or organization, web pages of the same general topic, or a group of researchers that frequently publish together. In this work, we address the following problem:

Definition 1 (Hierarchical Community Detection). *Given an (un)directed graph $G = (V, E)$, the problem of hierarchical community detection is to find*

(i) a hierarchy of communities denoted as $\mathbb{H} = \{\mathcal{C}^1, \ldots, \mathcal{C}^L\}$ where $\mathcal{C}^t = \{C_1^t, \ldots, C_k^t\}$ are the communities at level t in the hierarchy

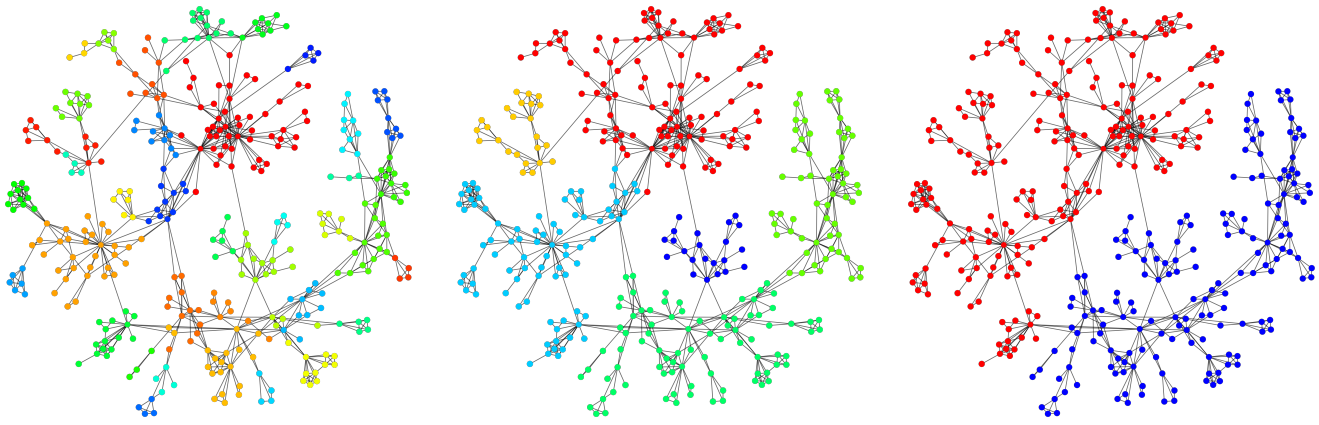
$$V_t = \bigcup_k C_k^t \quad \text{and} \quad |\mathcal{C}^1| < \cdots < |\mathcal{C}^t| < \cdots < |\mathcal{C}^L| \tag{1}$$

(ii) a hierarchy of community (super) graphs $G_1, \ldots, G_t, \ldots, G_L$ where $G_t = (V_t, E_t)$ succinctly captures the relationships between the communities (nodes in G_t) at a lower $t-1$ level in the hierarchy. The hierarchy of community (super) graphs indicate how the functional units (communities) of the graph interact at each level and how they combine to form larger communities.

While there have been a lot of work on community detection [7, 16], most research (i) does not address the hierarchical community detection problem (Definition 1) *or* are (ii) inefficient for large networks with a worst-case time (and space) complexity that is *not* linear in the number of edges. In this work, we describe an approach called HLP that addresses both these limitations. In particular, HLP solves the hierarchical community detection problem by detecting a hierarchy of communities (going from the lowest to highest granularity) along with a hierarchy of community (super) graphs that reveal the higher-order organization and components at each level and how these components interact with one another to form larger components at a higher-level in the hierarchy. Most importantly, HLP is fast and efficient for large networks with a worst-case time complexity that is linear in the number of edges whereas the space complexity of HLP is linear in the number of nodes.

2 APPROACH

This section describes our fast linear-time approach for revealing hierarchical communities in large graphs. Given G, the algorithm outputs a hierarchy of communities $\mathbb{H} = \{\mathcal{C}^1, \ldots, \mathcal{C}^L\}$ where L is the number of layers (*i.e.*, levels in the community hierarchy $\mathbb{H}$). A summary of the approach is shown in Algorithm 1. There are



(a) $\mathcal{C}^1$ (40 communities) (b) $\mathcal{C}^2$ (6 communities) (c) $\mathcal{C}^3$ (2 communities)

Figure 1: Network science co-authorship network. HLP summarizes the higher-order organization of the network at multiple granularities as shown in 1(a)-1(c). Node color encodes community assignment. See text for discussion.

two general steps: Label Propagation (Section 2.1) and Super Graph Construction (Section 2.2).

2.1 Label Propagation

Note $\Gamma(v_i) = \{j \in V \mid (i,j) \in E\}$ is the set of neighbors of node i. The first step performs label propagation. In particular, the approach begins with each node belonging to its own community. For each node $v_i \in V$ (or edge), we assign it to the community $C_k \in \mathcal{C}$ that has the maximum number of neighbors $\Gamma(v_i)$ in it. More formally,

$$\underset{C_k \in \mathcal{C}}{\operatorname{argmax}} \sum_{v_j \in \Gamma(v_i)} \mathbb{I}\left[\, v_j \in C_k \,\right] \tag{2}$$

where for any predicate p the indicator function $\mathbb{I}[p] = 1$ iff p holds and 0 otherwise. Hence, $\mathbb{I}\left[\, v_j \in C_k \,\right] = 1$ iff $v_j \in C_k$, and 0 otherwise. In other words, every node $v_i \in V$ is assigned the label that appears the most frequent in the 1-hop neighborhood of the node Eq. 2 can be easily replaced/modified to take into account other important aspects. The algorithm converges when an iteration results in no further changes (*i.e.*, no new assignments are made) or if the max number of iterations is reached which can be interactively tuned by the user. Upon each iteration, we compute a random permutation and use this ordering to assign nodes (or edges) to communities. To further speedup the approach, we leverage the number of previous iterations that the community assignment of a node (or edge) remained unchanged (*i.e.*, the community of v_i remained stable over the last t iterations). In particular, let δ denote a hyperparameter that controls the number of previous iterations that the community assignment of a node or edge must remain unchanged before it is declared as final. Thus, each iteration of the approach can be defined over the set S of graph elements (nodes/edges) that are still active, *i.e.*, $T_i < \delta$ where T_i denotes the number of subsequent iterations that v_i has remained unchanged (*w.r.t.* community assignment). Fast and efficient localized updates are performed when new nodes/edges arrive.

Algorithm 1 Hierarchical Community Detection (HLP)

Input: a graph $G = (V, E)$
Output: hierarchical communities $\mathbb{H} = \{\mathcal{C}^1, \ldots, \mathcal{C}^L\}$

Set $G_0 \leftarrow G$ to be the initial graph and $t \leftarrow 1$
repeat
  $\mathcal{C}^t \leftarrow$ LabelProp(G_{t-1})
  $G_t = (V_t, E_t) \leftarrow$ CreateSuperGraph($G_{t-1}, \mathcal{C}^t$) via Eq. 3
  $t \leftarrow t + 1$
until $|V_t| < 2$ ▷ Stop when no nodes to combine

Algorithm 2 Create Super Graph

Input: a graph $G_{t-1} = (V_{t-1}, E_{t-1})$, communities $\mathcal{C}^t$ from G_{t-1}
Output: community (super) graph $G_t = (V_t, E_t)$ for layer t

$V_t \leftarrow \mathcal{C}^{t-1}$ where $\mathcal{C}^{t-1} = \{C_1, \ldots, C_k\}$ ▷ Super node set
$E_t \leftarrow \emptyset$ ▷ Super edge set
Let $\mathbf{c}$ be the community assignment vector where $c_i = k$ if $v_i \in C_k$
parallel for $i \in V_{t-1}$ **do**
  for $j \in \Gamma_i$ **do** ▷ Neighbor of vertex i
    if $c_i \neq c_j$ **and** $(c_i, c_j) \notin E_t$ **then**
      $E_t \leftarrow E_t \cup (c_i, c_j)$
end parallel

2.2 Super Graph Construction

Given a graph G_{t-1} and $\mathcal{C}^t = \{C_1^t, \ldots, C_k^t\}$, Algorithm 2 computes the community (super) graph $G_t = (V_t, E_t)$ for layer t in the community hierarchy where $V_t \leftarrow \mathcal{C}^t$ and thus the number of nodes in G_t is $n_t = |\mathcal{C}^t|$, *i.e.*, the number of communities detected in the previous graph G_{t-1} (or level in the community hierarchy). Similarly, an edge $(i,j) \in E_t$ iff there is an edge between C_i^t and C_j^t in G_{t-1}, *i.e.*, a link exists between a node $r \in V_{t-1}$ assigned to community C_i^t and another node $s \in V_{t-1}$ assigned to community C_j^t. More

formally,

$$E_t = \left\{(i,j) \,:\, r \in C_i^t,\, s \in C_j^t \,\wedge\, (r,s) \in E_{t-1} \,\wedge\, i \neq j\right\} \quad (3)$$

Property 1. *Let $E_t(C_i, C_j)$ be the set of edges between C_i and C_j (cut set), then the number of edges $|E_{t+1}|$ in the next level $t+1$ is:*

$$|E_{t+1}| = \sum_{C_i \in \boldsymbol{\mathcal{C}}^t} \sum_{C_j \in \boldsymbol{\mathcal{C}}^t} |E_t(C_i, C_j)| \quad s.t.\ \ i < j \quad (4)$$

Note $|E_t(C_i, C_j)|$ does not include multi-edges.

Algorithm 2 returns $G_t = (V_t, E_t)$ for layer t in the hierarchy. The approach terminates when $|V_t| < 2$ as shown in Algorithm 1. Hence, HLP terminates when there are no nodes remaining to combine.

Property 2. *Let $|E(G_t)|$ and $|V(G_t)|$ be the number of edges and nodes in G_t and $G_0 \leftarrow G$, then*

$$|E(G_0)| \geq \cdots \geq |E(G_L)| \quad \textit{and} \quad |V(G_0)| \geq \cdots \geq |V(G_L)| \quad (5)$$

Property 2 has a number of important and useful implications that are leveraged in Section 3.

3 ANALYSIS

This section shows the worst-case time and space complexity of the proposed approach. Let L denote the number of layers (hierarchies) and let T denote the maximum number of iterations at any given layer. Both L and T are small. Further, let $N = |V|$ denote the number of nodes and let $M = |E|$ denote the number of edges in G.

3.1 Time Complexity

Lemma 1. *The worst-case time complexity of* hierarchical label propagation *is*

$$O(LTM) = O(M) \quad (6)$$

where L and T are small constants. Therefore, the time complexity is linear in the number of edges M in the graph.

Supergraph construction: The worst-case time complexity of Algorithm 2 is $O(|E_{t-1}|)$. This is bounded above by the number of edges denoted as $|E|$ in the input graph G.

3.2 Space Complexity

Lemma 2. *The space complexity of hierarchical label propagation is*

$$O(NL) \quad (7)$$

where L is a small constant. Therefore, the space complexity is linear in the number of nodes in G.

Lemma 2 assumes the node community assignments at each layer are stored and given as output to the user. However, this information can be significantly compressed by storing only the community assignments at the first layer, and then storing only how these communities are merged at each subsequent layer.

Supergraph construction: The worst-case space complexity of Algorithm 2 is $O(|E_{t-1}|)$. Similar to time complexity, this is bounded above by the number of edges denoted as $|E|$ in the input graph G.

4 EXPERIMENTS

The experiments in this section are designed to investigate the quality of the communities revealed by HLP and the utility of the hierarchical communities for a visualization application. For comparison, we use a wide variety of graphs from different application domains including social networks (soc), biological/protein networks (bio), infrastructure networks (inf), web graphs (web), road networks (road), and collaboration networks (ca). Due to space constraints, network statistics were removed but can be accessed online at http://networkrepository.com along with the data [14].

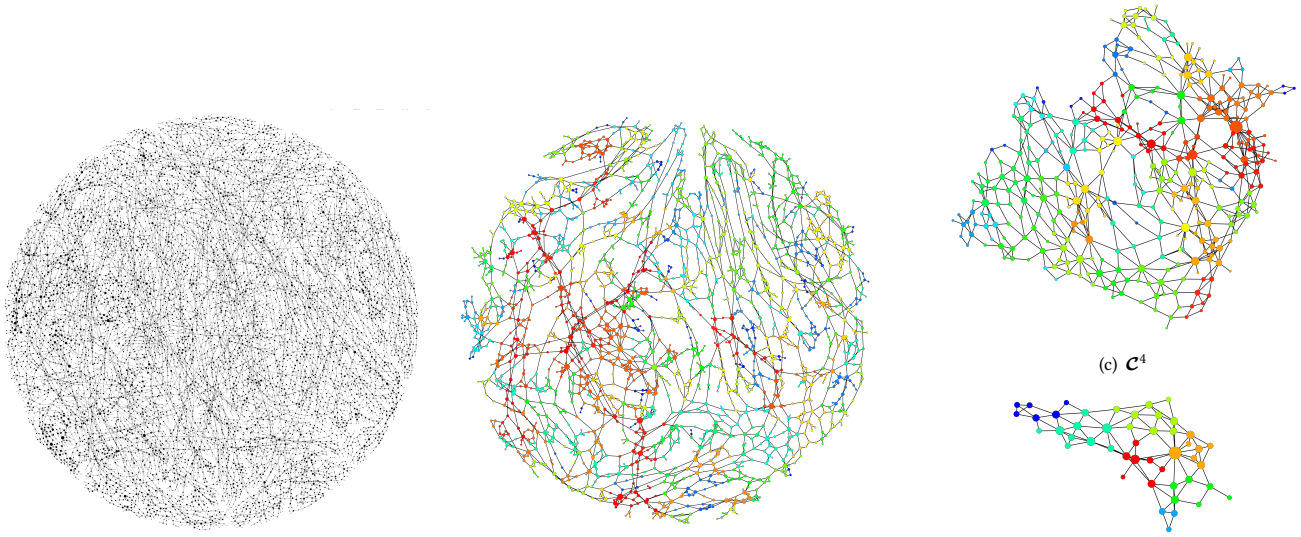
4.1 Comparison

4.1.1 Baseline methods. For fair comparison, we use baselines that are fast with *linear-time* complexity (with the exception of Louvain):

- **Densest Subgraph (DS)** [11]: This method finds an approximation of the densest subgraph in G using degeneracy ordering, and removes this subgraph. This is repeated until all nodes have been assigned.
- **KCore Communities (KCore)** [15, 18]: Many have observed that the largest k-core subgraphs of a real-world network are highly dense subgraphs that often contain the max clique [15]. The KCore baseline simply uses the maximum k-core subgraph as S and $\bar{S} = V \setminus S$.
- **Label Propagation (LP)** [13]: Label propagation takes a labeling of the graph, then for each node, the label is updated according to the label that occurs the most among its neighbors. This is repeated until convergence.
- **Louvain (Louv)** [2]: Louvain performs a greedy optimization of modularity by forming small, locally optimal communities then grouping each community into one node. This two-phase process is repeated until modularity cannot be maximized locally.
- **Spectral Clustering (Spec)** [4]: This baseline uses spectral clustering on the normalized Laplacian of the adjacency matrix to greedily build the sweeping cluster that minimizes conductance.

Table 1: Quantitative evaluation of the methods (modularity). The best result from each graph is bold. Note HLP is the proposed method.

	DS	KCore	LP	Louv	Spec	HLP
soc-yahoo-msg	0.0003	0.0004	0.0479	0.0394	0.0005	**0.0569**
bio-gene	0.0195	0.0217	0.0315	0.0408	-0.0208	**0.0846**
ca-cora	0.0089	0.0304	0.0444	0.0608	0.0164	**0.1026**
soc-terror	0.0888	0.0892	0.0967	0.0967	0.0999	**0.1243**
inf-US-powerGrid	0.0027	0.0027	0.0061	0.0212	0.1127	**0.1242**
web-google	0.0272	0.0275	0.0429	0.0471	0.1010	**0.1122**
ca-CSphd	0.0224	0.0224	0.0234	0.0198	0.0131	**0.1201**
ca-netscience	0.0164	0.0168	0.1063	0.0561	0.1229	**0.1233**
road-luxem.	0.0629	0.0629	0.0077	0.0046	-0.1170	**0.1141**
bio-DD21	0.0865	0.0866	0.0106	0.0202	0.1241	**0.1247**



(a) road-luxembourg ($\mathcal{C}^2$) (b) $\mathcal{C}^3$ (c) $\mathcal{C}^4$ (d) $\mathcal{C}^5$

Figure 2: In this case study, the network data is a road network of luxembourg consisting of 114,600 nodes and 239,332 edges making it impossible to visualize the entire network. There are 9,452 communities in $\mathcal{C}^2$ and therefore impossible to visualize by assigning each community a unique color. (a) Super graph derived after first layer consisting of 9,452 supernodes (communities) with 25,386 superedges (between community edges). (b) consists of 2,023 communities with only 6,588 between community edges whereas (c)-(d) consists of 372 and 48 communities with 1,580 and 214 between community edges, respectively. Nodes are weighted by degree. See text for discussion.

4.1.2 *Quantitative evaluation.*

We quantitatively evaluate the communities using modularity [12]. Modularity is defined as:

$$\mathbb{E}(\mathbf{c}) = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i, c_j) \tag{8}$$

where M is the number of edges, $\mathbf{A}$ is the adjacency matrix with $A_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise; d_i and d_j is the degree of node i and j; c_i and c_j are the community assignments of node i and node j; and δ is an indicator function such that $\delta(c_i, c_j) = 1$ if $c_i = c_j$ and 0 otherwise. We report the best result from any layer/level in the community hierarchy. Results are provided in Table 1. Notably, HLP outperforms all the other baseline methods across all graphs as shown in Table 1. HLP reveals better high quality communities across a wide variety graphs from different application domains (social, biological, infrastructure, among others) as shown in Table 1. Overall, HLP typically achieves at least an order of magnitude improvement over the other baseline methods.

Now we investigate the communities found by HLP by overlaying the community assignments on top of the network structure (node-link diagram). The communities given by HLP at different levels in the hierarchy are shown in Figure 1 for the network science co-authorship network. Communities in 1(a) represent small groups of researchers that frequently publish together whereas communities in 1(b) represent different research areas and so on.

4.1.3 *Runtime Performance.*

Figure 2 visualizes the important components (functional modules) of a large road network from luxembourg at multiple scales (layers). Note that using a serial python implementation of the proposed method takes only 10.2 seconds to derive the initial 9,452 communities visualized in Figure 2(a). However, the next layer is orders of magnitude faster taking less than a second (0.611 sec.) and the runtime steadily decreases as a function of the supergraph size (number of supernodes, superedges) and the number of iterations to converge in the preceding layers. Furthermore, the number of iterations until convergence also steadily decreases as the number of layers increases.

4.2 Visualizing Large Networks

One important application of HLP is visualization of large networks. In Figure 2, we use HLP to compute a hierarchy of communities for a large real-world network consisting of 114,600 nodes and 239,332 edges. While it is impractical and often impossible to visualize such a large network, we can use HLP to summarize the graph structure at multiple levels as shown in Figure 2. Instead of visualizing the graph at the level of intersections (nodes in the original road network), we can instead visualize the graph at a higher-level where nodes represent something more meaningful, *e.g.*, instead of intersections, nodes at layer 2 shown in Figure 2(b) might represent neighborhoods and edges represent routes from one neighborhood to another. Thus, HLP uncovers the *hierarchical higher-order organization* of complex networks.

REFERENCES

[1] Uri Alon. 2003. Biological networks: the tinkerer as an engineer. *Science* 301, 5641 (2003), 1866–1867.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *JSTAT* 10 (2008).

[3] Gregory Buehrer and Kumar Chellapilla. 2008. A scalable pattern mining approach to web graph compression with communities. In *WSDM*. 95–106.

[4] Fan RK Chung. 1997. *Spectral graph theory*. AMS.

[5] Inderjit S Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*.

[6] Pedro F Felzenszwalb and Daniel P Huttenlocher. 2004. Efficient graph-based image segmentation. *IJCV* 59, 2 (2004).

[7] Santo Fortunato. 2010. Community detection in graphs. *Phy. Rep.* 3 (2010).

[8] C Lee Giles. 2006. The future of citeseer: citeseer x. In *ECML*. Springer, 2–2.

[9] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *PNAS* 99, 12 (2002), 7821–7826.

[10] Bruce Hendrickson and Robert Leland. 1995. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM SISC* 16, 2 (1995).

[11] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *ICALP*.

[12] M.E.J. Newman. 2001. The structure of scientific collaboration networks. *PNAS* 98, 2 (2001), 404.

[13] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (2007), 036106.

[14] Ryan A. Rossi and Nesreen K. Ahmed. 2016. An Interactive Data Repository with Visual Analytics. *SIGKDD Exp.* (2016). http://networkrepository.com

[15] Ryan A. Rossi, David Gleich, and Assefaw Gebremedhin. 2015. Parallel Maximum Clique Algorithms with Applications to Network Analysis. *SISC* (2015).

[16] Satu Elisa Schaeffer. 2007. Graph clustering. *Comp. sci. rev.* 1, 1 (2007), 27–64.

[17] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* 22, 8 (2000), 888–905.

[18] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2016. CoreScope: Graph Mining Using k-Core Analysis–Patterns, Anomalies and Algorithms. In *ICDM*.

[19] Horst D Simon. 1991. Partitioning of unstructured problems for parallel processing. *Comp. Sys. in Eng.* 2, 2 (1991).

[20] Rafael Van Driessche and Dirk Roose. 1995. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel comp.* (1995).