J. Parallel Distrib. Comput. 76 (2015) 66–80

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Multi-threaded modularity based graph clustering using the multilevel paradigm

Dominique LaSalle *, George Karypis

Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, USA

HIGHLIGHTS

- We present efficient serial and parallel algorithms for modularity maximization.
- We show that these algorithms run in $O(m + n)$ time and $O(m + n)$ space.
- These algorithms produce clusterings of high modularity.
- These algorithms are 2.7–44 times faster than current methods.
- These algorithms clustered a graph with 3.3 billion edges in under 90 s.

ARTICLE INFO

Article history:
Received 24 March 2014
Received in revised form
29 July 2014
Accepted 22 September 2014
Available online 28 September 2014

Keywords:
Graph clustering
Multilevel paradigm
Multi-threading
Shared-memory parallel

ABSTRACT

Graphs are an important tool for modeling data in many diverse domains. Recent increase in sensor technology and deployment, the adoption of online services, and the scale of VLSI circuits has caused the size of these graphs to skyrocket. Finding clusters of highly connected vertices within these graphs is a critical part of their analysis.

In this paper we apply the multilevel paradigm to the modularity graph clustering problem. We improve upon the state of the art by introducing new efficient methods for coarsening graphs, creating initial clusterings, and performing local refinement on the resulting clusterings. We establish that for a graph with n vertices and m edges, these algorithms have an $O(m+n)$ runtime complexity and an $O(m+n)$ space complexity, and show that in practice they are extremely fast. We present shared-memory parallel formulations of these algorithms to take full advantage of modern architectures, which we show have a parallel runtime of $O(m/p+n/p+k)$, where p is the number of threads and k is the number of clusters. Finally, we present the product of this research, the clustering tool *Nerstrand*.[1] In serial mode, *Nerstrand* runs in a fraction of the time of current methods and produces results of equal quality. When run in parallel mode, *Nerstrand* exhibits significant speedup with less than one percent degradation of clustering quality. *Nerstrand* works well on large graphs, clustering a graph with over 105 million vertices and 3.3 billion edges in 90 s.

1. Introduction

Graphs are an important tool for representing data in many diverse domains. Graph clustering is a technique for analyzing the structure of a graph by identifying groups of highly connected vertices. Discovering this structure is an important task in social network, biological network, and web analysis. In recent years, the scale of these graphs has increased to millions of vertices and billions of edges, making this discovery increasingly difficult and costly.

Modularity [21] is one of the most widely used metrics for determining the quality of non-overlapping graph clusterings, especially in the network analysis community. The problem of finding a clustering with maximal modularity is NP-Complete [6]. As a result many polynomial time heuristic algorithms have been developed [19,9,20,3,33,8]. Among these algorithms, approaches resembling the multilevel paradigm as used in graph partitioning have been shown to produce high quality clustering solutions and scale to large graphs [5,22,26].

However, most of these approaches adhere closely to the agglomerative method of merging pairs of clusters iteratively. This can lead to skewed cluster sizes as well as require excessive

* Corresponding author.
E-mail addresses: lasalle@cs.umn.edu (D. LaSalle), karypis@cs.umn.edu (G. Karypis).

1 The *Nerstrand* software is available at http://cs.umn.edu/~lasalle/nerstrand.

http://dx.doi.org/10.1016/j.jpdc.2014.09.012
0743-7315/© 2014 Elsevier Inc. All rights reserved.

amounts of computation time. While methods for prioritizing cluster merges have been proposed to reduce skewed cluster sizes, these approaches are inherently serial. The use of post-clustering refinement has not been present in most of these approaches.

In this paper we present multilevel algorithms for generating high quality modularity-based graph clusterings. The contributions of our work are:

- A method for efficiently contracting a graph for the modularity objective.
- A robust method for generating clusterings of a contracted graph.
- A modified version of boundary refinement for the modularity objective.
- Shared-memory parallel formulations of these algorithms.

We show that for a graph with n vertices and m edges, these algorithms have $O(m+n)$ time and $O(m+n)$ space complexities. We show that the shared-memory parallel versions of these algorithms have a parallel time complexity of $O(m/p+n/p+k)$ where p is the number of threads and k is the number of clusters. To validate our contributions, we compare our implementation of these algorithms, *Nerstrand*, against the serial clustering tool *Louvain* [5] and the parallel clustering tools *community-el* [26] and *NetworKit* [28], and show that *Nerstrand* produces clusterings of equal or greater modularity and is 4.5–27.2 times faster than the methods that generate clusterings with competitive modularity. The parallel version of *Nerstrand* is scalable and extremely fast, clustering a graph with over 105 million vertices and 3.3 billion edges in 90 s using 16 cores.

This paper is organized into the following sections. In Section 2 we define the notation used throughout this paper. In Section 3 we give an overview of current graph clustering methods for maximizing modularity. In Section 4 we give an overview of the multilevel paradigm, its use in the graph partitioning problem, and more recently in the graph clustering problem. Descriptions of the serial algorithms we developed are presented in Section 5, and descriptions of their parallel counter parts are presented in Section 6. In Section 7 we describe our experimental setup. This is followed by the results of our experiments in Sections 8 and 9, in which we evaluate the quality and speed of the presented algorithms. Finally in Section 10, we review the findings of this paper.

2. Definitions and notation

A simple undirected *graph* $G=(V,E)$ consists of a set of vertices V and a set of edges E, where each edge $e=\{v,u\}$ is composed of an unordered pair of vertices (i.e., $v,u\in V$). The number of vertices is denoted by the scalar $n=|V|$, and the number of edges is denoted similarly as $m=|E|$. Each edge $e\in E$ can have a positive weight associated with it that is denoted by $\theta(e)$. If there are no weights associated with the edges, then their weights are assumed to be one.

Given a vertex $v\in V$, its set of adjacent vertices (connected by an edge) is denoted by $\Gamma(v)$ and is referred to as the *neighborhood* of v. For an unweighted graph, $d(v)$ denotes the number of edges incident to v (e.g., $d(v)=|\Gamma(v)|$), and for the case of weighted edges, $d(v)$ denotes the total weight of its incident edges (e.g., $d(v)=\sum_{u\in\Gamma(v)}\theta(\{v,u\})$).

A *clustering* C of G is described by the division of V into k non-empty and disjoint subsets $C=\{C_1,C_2,\ldots,C_k\}$, which are referred to as *clusters*. The sum of vertex degrees within a cluster is denoted as $d(C_i)$ (i.e., $d(C_i)=\sum_{v\in C_i}d(v)$). The internal degree $d_{\text{int}}(C_i)$ of a cluster C_i is the number of edges (or sum of the edge weight) that connect vertices in C_i to other vertices within C_i. The external degree $d_{\text{ext}}(C_i)$ of a cluster C_i is the number of edges (or sum of the edge weight) that connect vertices in C_i to vertices in other clusters. The neighborhood of a cluster V_i, that is all clusters connected to C_i by at least one edge, is denoted by $\Gamma(C_i)$. The number of edges connecting the cluster C_i to C_j is denoted as $d_{C_j}(C_i)$. Since G is an undirected graph, $d_{C_j}(C_i)=d_{C_i}(C_j)$. Similarly, the number of edges (or total edge weight) connecting a vertex v to the cluster C_i is denoted as $d_{C_i}(v)$ (i.e., $d_{C_i}(v)=\sum_{u\in C_i\cap\Gamma(v)}\theta(\{v,u\})$). To aid in the discussion of moving vertices between clusters, we will denote the cluster C_i with the vertex v removed, as $C_i-\{v\}$, and the cluster C_j with the vertex v added as $C_j+\{v\}$.

The metric of graph *modularity*, and the focus of this paper, was introduced by Newman and Girvan [21], and has become ubiquitous in the recent graph clustering/community detection literature. Modularity measures the difference between the expected number of intra-cluster edges and the actual number of intra-cluster edges. Denoted by Q, the modularity of a clustering C is expressed as

$$Q=\frac{1}{d(V)}\left(\sum_{C_i\in C}\left(d_{\text{int}}(C_i)-\frac{d(C_i)^2}{d(V)}\right)\right), \tag{1}$$

where $d(V)$ is the total degree of the entire graph (i.e., $d(V)=\sum_{v\in V}d(v)$). From this, we can see the modularity Q_{C_i} contributed by cluster C_i is

$$Q_{C_i}=\frac{1}{d(V)}\left(d_{\text{int}}(C_i)-\frac{d(C_i)^2}{d(V)}\right). \tag{2}$$

The value of Q ranges from -0.5, where all edges in the graph are inter-cluster edges, and approaches 1.0 if all edges in the graph are intra-cluster edges and there is a large number of clusters. Note that this metric does not use the number of vertices within a cluster, but rather only the edges. Subsequently, vertices of degree zero, can arbitrarily be placed in any cluster without changing the modularity.

3. Modularity based graph clustering

A large number of approaches for maximizing modularity have been developed since it was first proposed [21] a decade ago. Fortunato [13] provides an overview of modularity and methods for its maximization.

The majority of approaches fall into the category of agglomerative clustering. In agglomerative clustering, each vertex is placed in its own cluster, and pairs of clusters are iteratively merged together if it increases the modularity of the clustering. When there exists no pair of clusters whose merging would result in an increase in modularity, the process stops, and the clustering is returned.

The greedy agglomerative method introduced by Clauset et al. [9], is the most well-known of the these approaches, due to its ability to find good clusterings in relatively little time. Its low runtime is the result of exploiting the sparse structure of the graph to limit the number of merges it needs to consider and the number of updates that it needs to perform during agglomeration. The quality of the clusterings it finds is the result of recording the modularity after each merge, and continuing to perform cluster merges until there is only a single cluster, and then reverting to the state with the maximum modularity. The structure used to maintain this state information is a binary tree in which each node represents a cluster, and the children of a node are the clusters which were merged to form the node. They established an upper bound on the complexity of this algorithm of $O(mh\log n)$, where h is the height of the tree recording cluster merges. If this tree is fairly balanced, h will be close to $\log n$.

It was noted that this algorithm tends to discover several super-clusters, composed of most of the vertices in the graph. Wakita and