VGP
A software package for one-pass Vertex-cut balanced Graph Partitioning

Fabio Petroni*

2015

Contents

*http://www.fabiopetroni.com

1 Introduction

This software package provide solutions to partition (maintaining load balance among partitions) an unweighted undirected graph preforming a single pass over the data and using a vertex-cut approach. The problem of optimally partitioning a graph to minimize vertex-cuts while maintaining load balance is a fundamental problem in parallel and distributed applications as input placement significantly affects the efficiency of algorithm execution. A vertex-cut partitioning scheme results in partitions that are edge disjoint, while a vertex can be cut in multiple ways and span several partitions.

If you use the program please cite [3].

2 VGP stand-alone program

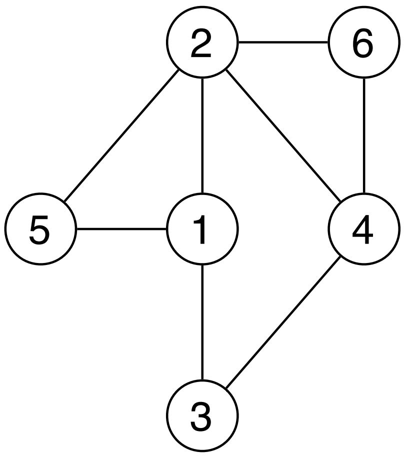
2.1 Input file format

The input of the program is the undirected unweighted graph to be partitioned. A graph $G = (V, E)$ with n vertices (i.e. $|V| = n$) and m edges (i.e. $|E| = m$) is stored in a plain text file and supplied to the program as one command line parameter. Concretely, the graph is represented as an edge-list, one line per edge (i.e. the file contains m lines); each edge is represented by indicating the two connected vertices. Figure 1 shows an example of storage format for an unweighted undirected graph. Notice that the default separator between two vertices is tab (i.e. "̂), but is possible to define a different separator character as one optional command line parameter.

2.2 Output files format

The output of the program is a partition of the graph. It is possible to store the result by defining the *prefix* of the output files as one optional command line parameter. Concretely, the program store three files: *prefix.info*, *prefix.edges*, *prefix.vertices*.

prefix.info This file contains some information for the computation. In particular the parameter given as input to the program and some statistics for the final partition, such as *replication factor*, *load relative standard deviation*



Graph file:

```
1 2
1 3
1 5
2 4
2 5
2 6
3 4
4 6
```

Figure 1: Example of storage format for an unweighted undirected graph.

and *maximum partition size* (for a more in-depth discussion of these metrics, see [3]).

prefix.edges The files contains the partitions where each edge is stored. A vertex-cut partitioning scheme results in partitions that are edge disjoint, therefore each edge is placed in exactly one partition. The output file contains one line per edge with the id of the partition where the algorithm placed it.

prefix.vertices The file contains the partitions where each vertex is replicated. In fact, a vertex can be cut in multiple ways and span (i.e. be replicated in) several partitions. The output file contains one line per vertex withe the ids of the partitions where the algorithm created a replica for it.

2.3 Program

To run the project from the command line, go to the dist folder and type the following:

```
java -jar ”VGP.jar” ...
```

Usage:

```
VGP graphfile nparts [options]
```

Parameters:

graphfile: the name of the file that stores the graph to be partitioned.

nparts: the number of parts that the graph will be partitioned into. Maximum value 256.

Options:

-algorithm string

specifies the algorithm to be used (hdrf greedy hashing grid pds dbh). Default hdrf.

-lambda double

specifies the lambda parameter for hdrf. Default 1.

-threads integer

specifies the number of threads used by the application. Default all available processors.

-output string

specifies the prefix for the name of the files where the output will be stored (files: *prefix.info*, *prefix.edges*, *prefix.vertices*).

3 Algorithms

Consider a graph $G = (V, E)$, where $V = (v_1, \cdots, v_n)$ is the set of vertices and $E = (e_1, \cdots, e_m)$ the set of edges. We define a partition of edges $P = (p_1, .., p_k)$ to be a family of pairwise disjoint sets of edges (i.e. $p_i, p_j \subseteq E$, $p_i \cap p_j = \emptyset$ for every $i \neq j$). Let $A(v) \subseteq P$ be the set of partitions each vertex $v \in V$ is replicated. The size $|p|$ of each partition $p \in P$ is usually defined as its edge cardinality.

3.1 hashing

The *hashing* technique (pseudo-)randomly assigns each edge to a partition: for each input edge $e \in E$, $A(e) = h(e) \bmod |P|$ is the identifier of the target partition.

3.2 dbh [4]

The *Degree-Based Hashing* (*DBH*) algorithm, a variation of the *hashing* heuristic that explicitly considers the degree of the vertices for the placement decision. Concretely, when processing edge $e \in E$ connecting vertices $v_i, v_j \in V$ with degrees d_i and d_j, *DBH* defines the hash function $h(e)$ as follows:

$$h(e) = \begin{cases} h(v_i), & \text{if } d_i < d_j \\ h(v_j), & \text{otherwise} \end{cases}$$

Then, it operates as the *hashing* algorithm.

3.3 Constrained partitioning: grid and pds [2]

The *grid* and *PDS* techniques belong to the *constrained partitioning* family of algorithms. The general idea of these solutions is to allow each vertex $v \in V$ to be replicated only in a small subset of partitions $S(v) \subset P$ that is called the *constrained set* of v. The constrained set must guarantees some properties; in particular, for each $v_i, v_j \in V$: (i) $S(v_i) \cap S(v_j) \neq \emptyset$; (ii) $S(v_i) \nsubseteq S(v_j)$ and $S(v_j) \nsubseteq S(v_i)$; (iii) $|S(v_i)| = |S(v_j)|$. To position a new edge e connecting vertices v_i and v_j, it picks a partition from the intersection between $S(v_i)$ and $S(v_j)$ either randomly or by choosing the least loaded one. Different solutions differ in the composition of the vertex constrained sets. The *grid* solution arranges partitions in a $X \times Y$ matrix such that $|P| = XY$. It maps each vertex v to a matrix cell using a hash function h, then $S(v)$ is the set of all the partitions in the corresponding row and column. It this way each constrained sets pair has at least two partitions in their intersection. *PDS* generates constrained sets using *Perfect Difference Sets*. This ensure that each pair of constrained sets has exactly one partition in the intersection. *PDS* can be applied only if $|P| = x^2 + x + 1$, where x is a prime number.

3.4 greedy [1]

When processing edge $e \in E$ connecting vertices $v_i, v_j \in V$, the *greedy* technique follows this simple set of rules:

Case 1: If neither v_i nor v_j have been assigned to a partition, then e is placed in the partition with the smallest size in P.

Case 2: If only one of the two vertices has been already assigned (without loss of generality assume that v_i is the assigned vertex) then e is placed in the partition with the smallest size in $A(v_i)$.

Case 3: If $A(v_i) \cap A(v_j) \neq \emptyset$, then edge e is placed in the partition with the smallest size in $A(v_i) \cap A(v_j)$.

Case 4: If $A(v_i) \neq \emptyset$, $A(v_j) \neq \emptyset$ and $A(v_i) \cap A(v_j) = \emptyset$, then e is placed in the partition with the smallest size in $A(v_i) \cup A(v_j)$ and a new vertex replica is created accordingly.

Symmetry is broken with random choices. An equivalent formulation consists of computing a score $C^{\text{greedy}}(v_i, v_j, p)$ for all partitions $p \in P$, and then assigning e to the partition p^* that maximizes C^{greedy}. The score consists of two elements: (i) a replication term $C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p)$ and (ii) a balance term $C^{\text{greedy}}_{\text{BAL}}(p)$. It is defined as follows:

$$C^{\text{greedy}}(v_i, v_j, p) = C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p) + C^{\text{greedy}}_{\text{BAL}}(p) \tag{1}$$

$$C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p) = f(v_i, p) + f(v_j, p) \tag{2}$$

$$f(v,p) = \begin{cases} 1, & \text{if } p \in A(v) \\ 0, & \text{otherwise} \end{cases}$$

$$C^{\text{greedy}}_{\text{BAL}}(p) = \frac{\text{maxsize} - |p|}{\epsilon + \text{maxsize} - \text{minsize}} \tag{3}$$

where *maxsize* is the maximum partition size, *minsize* is the minimum partition size, and ϵ is a small constant value.

3.5 hdrf [3]

When processing edge $e \in E$ connecting vertices v_i and v_j, the *HDRF* algorithm retrieves their partial degrees and increments them by one. Let $\delta(v_i)$

be the partial degree of v_i and $\delta(v_j)$ be the partial degree of v_j. The degree values are then normalized such that they sum up to one:

$$\theta(v_i) = \frac{\delta(v_i)}{\delta(v_i) + \delta(v_j)} = 1 - \theta(v_j) \tag{4}$$

As for the greedy heuristic, the *HDRF* algorithm computes a score $C^{\text{HDRF}}(v_i, v_j, p)$ for all partitions $p \in P$, and then assigns e to the partition p^* that maximizes C^{HDRF}. The score for each partition $p \in P$ is defined as follows:

$$C^{\text{HDRF}}(v_i, v_j, p) = C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p) + C^{\text{HDRF}}_{\text{BAL}}(p) \tag{5}$$

$$C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p) = g(v_i, p) + g(v_j, p) \tag{6}$$

$$g(v, p) = \begin{cases} 1 + (1 - \theta(v)), & \text{if } p \in A(v) \\ 0, & \text{otherwise} \end{cases}$$

$$C^{\text{HDRF}}_{\text{BAL}}(p) = \lambda \cdot C^{\text{greedy}}_{\text{BAL}}(p) = \lambda \cdot \frac{\text{maxsize} - |p|}{\epsilon + \text{maxsize} - \text{minsize}} \tag{7}$$

The λ parameter allows control of the extent of partition size imbalance in the score computation.

References

[1] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, pages 17–30, 2012.

[2] N. Jain, G. Liao, and T. L. Willke. Graphbuilder: Scalable graph etl framework. In *First International Workshop on Graph Data Management Experiences and Systems*, page 4. ACM, 2013.

[3] F. Petroni, L. Querzoni, G. Iacoboni, K. Daudjee, and S. Kamali. Hdrf: Efficient stream-based partitioning for power-law graphs. In *CIKM*, 2015.

[4] C. Xie, L. Yan, W.-J. Li, and Z. Zhang. Distributed power-law graph computing: Theoretical and empirical analysis. In *Advances in Neural Information Processing Systems*, pages 1673–1681, 2014.