

## Graph clustering

The *graph* concept is extremely general, and allows a rough categorization in *complete weighted* graphs, *weighted structured* graphs, and *simple* graphs. These three categories can be viewed as constituting the two ends and the middle of a spectrum of possibilities. For convenience I shall assume that weights, if present, are nonnegative. The clustering problem is one of the few problems which makes sense for all three categories. Until now it has mainly been studied in the setting of weighted complete graphs, which are often also metric — in which case it is really better to use the label metric space. I adhere to the custom that *graphs* — without further qualification — are simple graphs, which is in line with common usage. In this chapter I discuss the clustering problem in the setting of simple graphs, which is in a sense the natural habitat of the *MCL* algorithm, and the resemblances and differences with clustering in the setting of complete weighted graphs. To this end, a brief sketch is given of different types of problems in graph theory.

The research area *combinatorial optimization* involves graphs from all three categories, but individual problems (e.g. the Travelling Salesman Problem, the clique problem) are generally attached to a specific category. Clustering is a sibling of the optimization problem known as *graph partitioning* (Section 3.2), which is a well-defined problem in terms of cost functions. The field of combinatorial optimization appears in the course of a brief non-exhaustive survey of different branches in graph theory (Section 3.1), which is interesting because it naturally brings along the issue of tractability. The chapter concludes with a discussion of the implications of the differences between the vector model and the graph model for clustering.

### 3.1 Graphs, structure, and optimization

The concept of a *simple graph* is tightly associated with the study of its structural properties. Examples are the study of regular objects such as *distance regular graphs*, the study of *graph spectra* and *graph invariants*, and the theory of *randomly generated graphs*. The latter concept is introduced in some more detail, because it is used in defining a benchmark model for graph clustering in Chapter 12. A generic model via which a simple graph on  $N$  edges is randomly generated is to independently realize each edge with some fixed probability  $p$  (a parameter in this model). Typical questions in this branch of random graph theory concern the behaviour of quantities such as the number of components, the girth, the chromatic number, the maximum clique size, and the diameter, as  $n$  goes to infinity. One direction is to hold  $p$  fixed and look at the behaviour of the quantity under consideration; another is to find bounds for  $p$  as a function of  $n$

such that almost every random graph generated by the model has a prescribed property (e.g. diameter  $k$ ,  $k$  fixed).

In the theory of nonnegative matrices, structural properties of simple graphs are taken into account, especially regarding the 0/1 structure of matrix powers. Concepts such as cyclicity and primitivity are needed in order to overcome the complications arising from this 0/1 structure. A prime application in the theory of Markov matrices is the classification of states in terms of their graph-theoretical properties, and the complete understanding of limit matrices in terms of the analytic and structural (with respect to the underlying graph) properties of the generating matrix.

A different perspective is found in the retrieval or recognition of combinatorial notions such as above. In this line of research, algorithms are sought to find the diameter of a graph, the largest clique, a maximum matching, whether a second graph is embeddable in the first, or whether a simple graph has a Hamiltonian circuit or not. Some of these problems are doable; they are solvable in polynomial time, like the diameter and the matching problems. Other problems are extremely hard, like the clique problem and the problem of telling whether a given graph has a Hamiltonian cycle or not. For these problems, no algorithm is known that works in polynomial time, and the existence of such an algorithm would imply that a very large class of hard problems admits polynomial time<sup>1</sup> algorithms for their solution. Other problems in this class are (the recognition version of) the Travelling Salesman Problem, many network flow and scheduling problems, the satisfiability problem for logic formulas, *and* (a combinatorial version of) the clustering problem. Technically, these problems are all in both the class known as *NP*, and the subclass of *NP*-complete problems. Roughly speaking, the class *NP* consists of problems for which it can be verified in polynomial time that a solution to the problem is indeed what it claims to be. The *NP*-complete problems are a close-knit party from a complexity point of view, in the sense that *if any NP-complete problem admits a polynomial-time solution, then so do all NP-complete problems* — see [134] for an extensive presentation of this subject. The subset of problems in *NP* for which a polynomial algorithm exists is known as *P*. The abbreviations *NP* and *P* stand for *nondeterministic polynomial* and *polynomial* respectively. Historically, the class *NP* was first introduced in terms of non-deterministic Turing machines. The study of this type of problems is generally listed under the denominator of *combinatorial optimization*.

### 3.2 Graph partitioning and clustering

In graph partitioning well-defined problems are studied which closely resemble the problem of finding good clusterings, in the settings of both simple and weighted graphs respectively. Restricting attention first to the bi-partitioning of a graph, let  $(S, S^c)$  denote a partition of the vertex set  $V$  of a graph  $(V, E)$  and let  $\Sigma(S, S^c)$  be the total sum of weights of edges between  $S$  and  $S^c$ , which is the cost associated with  $(S, S^c)$ . The maximum cut problem is to find a partition  $(S, S^c)$  which maximizes  $\Sigma(S, S^c)$ , and the minimum quotient cut problem is to find a partition which minimizes  $\Sigma(S, S^c) / \min(|S|, |S^c|)$ . The

---

<sup>1</sup>Polynomial in the size of the problem instance; for a graph this is typically the number of vertices.

recognition versions of these two problems (i.e. given a number  $L$ , is there a partition with a cost not exceeding  $L$ ?) are *NP*-complete; the optimization problems themselves are *NP*-hard [62], which means that there is essentially only one way known to prove that a given solution is optimal; list all other solutions with their associated cost or yield. In the standard graph partitioning problem, the sizes of the partition elements are prescribed, and one is asked to minimize the total weight of the edges connecting nodes in distinct subsets in the partition. Thus sizes  $m_i > 0$ ,  $i = 1, \dots, k$ , satisfying  $k < n$  and  $\sum m_i = n$  are specified, where  $n$  is the size of  $V$ , and a partition of  $V$  in subsets  $S_i$  of size  $m_i$  is asked which minimizes the given cost function. The fact that this problem is *NP*-hard (even for simple graphs and  $k = 2$ ) implies that it is inadvisable to seek exact (best) solutions. Instead, the general policy is to devise good heuristics and approximation algorithms, and to find bounds on the optimal solution. The role of a cost/yield function is then mainly useful for comparison of different strategies with respect to common benchmarks. The clustering problem in the setting of simple graphs and weighted structured graphs is a kind of mixture of the standard graph partitioning problem with the minimum quotient cut problem, where the fact that the partition may freely vary is an enormously complicating factor.

**3.2.1 Graph partitioning applications.** The first large application area of graph partitioning (GP) is found in the setting of sparse matrix multiplication, parallel computation, and the numerical solving of PDE's (partial differential equations). Parallel computation of *sparse matrix multiplication* requires the distribution of rows to different processors. Minimizing the amount of communication between processors is a graph partitioning problem. The computation of so called *fill-reducing orderings of sparse matrices* for solving large systems of linear equations can also be formulated as a GP-problem [74]. PDE solvers act on a mesh or grid, and parallel computation of such a solver requires a *partitioning of the mesh*, again in order to minimize communication between processors [53].

The second large application area of graph partitioning is found in 'VLSI CAD', or Very Large Scale Integration (in) Computer Aided Design. In the survey article [8] Alpert and Kahng list several subfields and application areas (citations below are from [8]):

- *Design packaging*, which is the partitioning of logic arrays into clustering. "This problem (...) is still the canonical partitioning application; it arises not only in chip floorplanning and placement, but also at all other levels of the system design."
- Partitioning in order to improve mappings from HDL<sup>2</sup> descriptions to gate- or cell-level netlists.
- Estimation of wiring requirements and system performance in high-level synthesis and floorplanning.
- Partitioning supports "the mapping of complex circuit designs onto hundreds or even thousands of interconnected FPGA's" (Field-Programmable Gate Array).
- A good partition will "minimize the number of inter-block signals that must be multiplexed onto the bus architecture of a hardware simulator or mapped to the global interconnect architecture of a hardware emulator".

---

<sup>2</sup>Hardware Description Language.

Summing up, partitioning is used to model the distribution of tasks in groups, or the division of designs in modules, such that there is a minimal degree of interdependency or connectivity. Other application areas mentioned by Elsner in [53] are hypertext browsing, geographic information services, and mapping of DNA sequences.

**3.2.2 Clustering as a preprocessing step in graph partitioning.** There is an important dichotomy in graph partitioning with respect to the role of clustering. Clustering can be used as a preprocessing step if natural groups are present, in terms of graph connectivity. This is clearly not the case for meshes and grids, but it is the case in the VLSI CAD applications introduced above, where the input is most often described in terms of hypergraphs.

The currently prevailing methods in hypergraph partitioning for VLSI circuits are so-called *multilevel* approaches where hypergraphs are first transformed into normal graphs by some heuristic. Subsequently, the dimensionality of the graph is reduced by clustering; this is usually referred to as *coarsening*. Each cluster is contracted to a single node and edge weights between the new nodes are computed in terms of the edge weights between the old nodes. The reduced graph is partitioned by (a combination of) dedicated techniques such as spectral or move-based partitioning.

The statement that multi-level methods are prevalent in graph partitioning (for VLSI circuits) was communicated by Charles J. Alpert in private correspondence. Graph partitioning research experiences rapid growth and development, and seems primarily US-based. It is a non-trivial task to analyse and classify this research. One reason for this is the high degree of cohesion. The number of articles with any combination of the phrases *multi-level*, *multi-way*, *spectral*, *hypergraph*, *circuit*, *netlist*, *partitioning*, and *clustering* is huge — witness e.g. [7, 9, 31, 32, 38, 39, 73, 75, 81, 101, 102, 138, 174]. Spectral techniques are firmly established in graph partitioning, but allow different approaches, e.g. different transformation steps, use of either the Laplacian of a graph or its adjacency matrix, varying numbers of eigenvectors used, and different ways of mapping eigenspaces onto partitions. Moreover, spectral techniques are usually part of a larger process such as the multi-level process described here. Each step in the process has its own merits. Thus there is a large number of variables, giving way to an impressive proliferation of research. The spectral theorems which are fundamental to the field are given in Chapter 8, together with simple generalizations towards the class of matrices central to this thesis, diagonally symmetric matrices.

### 3.3 Clustering in weighted complete versus simple graphs

The *MCL* algorithm was designed to meet the challenge of finding cluster structure in simple graphs. In the data model thus far prevailing in cluster analysis, here termed a *vector model* (Section 2.3 in the previous chapter), entities are represented by numerical scores on attributes. The models are obviously totally different, since the vector model induces a (metric) Euclidean space, along with geometric notions such as convexity, density, and centres of gravity (i.e. vector averages), notions not existing in the (simple)

graph model. Additionally, in the vector model all dissimilarities are immediately available, but not so in the (simple) graph model. In fact, there is no notion of similarity between disconnected nodes except for third party support, e.g. the number of paths of higher length connecting two nodes.

Simple graphs are in a certain sense much closer to the space of partitions (equivalently, clusterings) than vector-derived data. The class of simple graphs which are disjoint unions of complete graphs are in a natural 1-1 correspondence with the space of partitions. A disjoint union of simple complete graphs allows only one sensible clustering. This clustering is perfect for the graph, and conversely, there is no other simple graph for which the corresponding partition is better justified<sup>3</sup>. Both partitions and simple graphs are generalized in the notion of a hypergraph. However, there is not a best constellation of points in Euclidean space for a given partition. The only candidate is a constellation where vectors in the same partition element are infinitely close, and where vectors from different partition elements are infinitely far away from each other.

On the other hand, a cluster hierarchy resulting from a hierarchical cluster method (most classic methods are of this type) can be interpreted as a tree. If the input data is Euclidean, then it is natural to identify the tree with a tree metric (which satisfies the so called *ultrametric inequality*), and consider a cluster method 'good' if the tree-metric produced by it is close to the metric corresponding with the input data. Hazewinkel [80] proved that single link clustering is optimal if the (metric) distance between two metrics is taken to be the Lipschitz distance. It is noteworthy that the tree associated with single link clustering can be derived from the minimal spanning tree of a given dissimilarity space [88].

What is the relevance of the differences between the data models for clustering in the respective settings? This is basically answered by considering what it requires to apply a vector method to the (simple) graph model, and vice versa, what it requires to apply a graph method to the vector model. For the former, it is in general an insurmountable obstacle that generalized (dis)similarities are too expensive (in terms of space, and consequently also time) to compute. Dedicated graph methods solve this problem either by randomization (Section 5.4) or by a combination of localizing and pruning generalized similarities (the *MCL* algorithm, Chapter 11).

More can be said about the reverse direction, application of graph methods to vector data. Methods such as single link and complete link clustering are easily formulated in terms of threshold graphs associated with a given dissimilarity space (Chapter 4). This is mostly a notational convenience though, as the methods used for going from graph to clustering are straightforward and geometrically motivated. For intermediate to large threshold levels the corresponding threshold graph simply becomes too large to be represented physically if the dimension of the dissimilarity space is large. Proposals of a combinatorial nature have been made towards clustering of threshold graphs, but these are computationally highly infeasible (Section 4.2).

---

<sup>3</sup>These considerations form the starting point for the formulation of generic performance criteria for clusterings of simple and weighted graphs in Chapter 9.

Another type of graph encountered in the vector model is that of a *neighbourhood graph* satisfying certain constraints (see [94] for a detailed account). Examples are the Minimum Spanning Tree, the Gabriel Graph, the Relative Neighbourhood Graph, and the Delaunay Triangulation. For example, the Gabriel Graph of a constellation of vectors is defined as follows. For each pair of vectors (corresponding with entities) a sphere is drawn which has as centre the geometric mean of the vectors, with radius equal to half the Euclidean distance between the two vectors. Thus, the coordinates of the two vectors identify points in Euclidean space lying diametrically opposed on the surface of the sphere. Now, two entities are connected in the associated Gabriel Graph if no other vector lies in the sphere associated with the entities. The step from neighbourhood graph to clustering is made by formulating heuristics for the removal of edges. The connected components of the resulting graph are then interpreted as clusters. The heuristics are in general local in nature and are based upon knowledge and/or expectations regarding the process via which the graph was constructed.

The difficulties in applying graph methods to threshold graphs and neighbourhood graphs are illustrated in Section 10.6. The *MCL* algorithm is tested for graphs derived by a threshold criterion from a grid-like constellation of points, and is shown to produce undesirable clusterings under certain conditions. The basic problem is that natural clusters with many elements and with relatively large diameter place severe constraints on the parametrization of the *MCL* process needed to retrieve them. This causes the *MCL* process to become expensive in terms of space and time requirements. Moreover, it is easy to find constellations such that the small space of successful parametrizations disappears entirely. This experiment clearly shows that the *MCL* algorithm is not well suited to certain classes of graphs, but it is argued that these limitations apply to a larger class of (tractable) graph clustering algorithms.