Graph partition

In mathematics, a **graph partition** is the reduction of a graph to a smaller graph by partitioning its set of nodes into mutually exclusive groups. Edges of the original graph that cross between the groups will produce edges in the partitioned graph. If the number of resulting edges is small compared to the original graph, then the partitioned graph may be better suited for analysis and problem-solving than the original. Finding a partition that simplifies graph analysis is a hard problem, but one that has applications to scientific computing, VLSI circuit design, and task scheduling in multiprocessor computers, among others.[1] Recently, the graph partition problem has gained importance due to its application for clustering and detection of cliques in social, pathological and biological networks. For a survey on recent trends in computational methods and applications see Buluc et al. (2013).[2] Two common examples of graph partitioning are minimum cut and maximum cut problems.

Problem complexity

Typically, graph partition problems fall under the category of NP-hard problems. Solutions to these problems are generally derived using heuristics and approximation algorithms.[3] However, uniform graph partitioning or a balanced graph partition problem can be shown to be NP-complete to approximate within any finite factor.[1] Even for special graph classes such as trees and grids, no reasonable approximation algorithms exist,[4] unless P=NP. Grids are a particularly interesting case since they model the graphs resulting from Finite Element Model (FEM) simulations. When not only the number of edges between the components is approximated, but also the sizes of the components, it can be shown that no reasonable fully polynomial algorithms exist for these graphs.[4]

Problem

Consider a graph $G = (V, E)$, where V denotes the set of n vertices and E the set of edges. For a (k,v) balanced partition problem, the objective is to partition G into k components of at most size $v \cdot (n/k)$, while minimizing the capacity of the edges between separate components.[1] Also, given G and an integer $k > 1$, partition V into k parts (subsets) $V_1, V_2, ..., V_k$ such that the parts are disjoint and have equal size, and the number of edges with endpoints in different parts is minimized. Such partition problems have been discussed in literature as bicriteria-approximation or resource augmentation approaches. A common extension is to hypergraphs, where an edge can connect more than two vertices. A hyperedge is not cut if all vertices are in one partition, and cut exactly once otherwise, no matter how many vertices are on each side. This usage is common in electronic design automation.

Analysis

For a specific $(k, 1 + \varepsilon)$ balanced partition problem, we seek to find a minimum cost partition of G into k components with each component containing a maximum of $(1 + \varepsilon)\cdot(n/k)$ nodes. We compare the cost of this approximation algorithm to the cost of a $(k,1)$ cut, wherein each of the k components must have the same size of (n/k) nodes each, thus being a more restricted problem. Thus,

$$\max_i |V_i| \leq (1+\varepsilon) \left\lceil \frac{|V|}{k} \right\rceil .$$

We already know that (2,1) cut is the minimum bisection problem and it is NP-complete.[5] Next, we assess a 3-partition problem wherein $n = 3k$, which is also bounded in polynomial time.[1] Now, if we assume that we have a finite approximation algorithm for (k, 1)-balanced partition, then, either the 3-partition instance can be solved using the balanced (k,1) partition in G or it cannot be solved. If the 3-partition instance can be solved, then (k, 1)-balanced partitioning problem in G can be solved without cutting any edge. Otherwise, if the 3-partition instance cannot be solved, the optimum (k, 1)-balanced partitioning in G will cut at least one edge. An approximation algorithm with a finite approximation factor has to differentiate between these two cases. Hence, it can solve the 3-partition problem which is a contradiction under the assumption that $P = NP$. Thus, it is evident that (k,1)-balanced partitioning problem has no polynomial-time approximation algorithm with a finite approximation factor unless $P = NP$.[1]

The planar separator theorem states that any n-vertex planar graph can be partitioned into roughly equal parts by the removal of $O(\sqrt{n})$ vertices. This is not a partition in the sense described above, because the partition set consists of vertices rather than edges. However, the same result also implies that every planar graph of bounded degree has a balanced cut with $O(\sqrt{n})$ edges.

Graph partition methods

Since graph partitioning is a hard problem, practical solutions are based on heuristics. There are two broad categories of methods, local and global. Well-known local methods are the Kernighan–Lin algorithm, and Fiduccia-Mattheyses algorithms, which were the first effective 2-way cuts by local search strategies. Their major drawback is the arbitrary initial partitioning of the vertex set, which can affect the final solution quality. Global approaches rely on properties of the entire graph and do not rely on an arbitrary initial partition. The most common example is spectral partitioning, where a partition is derived from approximate eigenvectors of the adjacency matrix, or spectral clustering that groups graph vertices using the eigendecomposition of the graph Laplacian matrix.

Multi-level methods

A multi-level graph partitioning algorithm works by applying one or more stages. Each stage reduces the size of the graph by collapsing vertices and edges, partitions the smaller graph, then maps back and refines this partition of the original graph.[6] A wide variety of partitioning and refinement methods can be applied within the overall multi-level scheme. In many cases, this approach can give both fast execution times and very high quality results. One widely used example of such an approach is METIS,[7] a graph partitioner, and hMETIS, the corresponding partitioner for hypergraphs.[8] An alternative

approach originated from [9] and implemented, e.g., in scikit-learn is spectral clustering with the partitioning determined from eigenvectors of the graph Laplacian matrix for the original graph computed by LOBPCG solver with multigrid preconditioning.

Spectral partitioning and spectral bisection

Given a graph $G = (V, E)$ with adjacency matrix A, where an entry A_{ij} implies an edge between node i and j, and degree matrix D, which is a diagonal matrix, where each diagonal entry of a row i, d_{ii}, represents the node degree of node i. The Laplacian matrix L is defined as $L = D - A$. Now, a ratio-cut partition for graph $G = (V, E)$ is defined as a partition of V into disjoint U, and W, minimizing the ratio

$$\frac{|E(G) \cap (U \times W)|}{|U| \cdot |W|}$$

of the number of edges that actually cross this cut to the number of pairs of vertices that could support such edges. Spectral graph partitioning can be motivated[10] by analogy with partitioning of a vibrating string or a mass-spring system and similarly extended to the case of negative weights of the graph.[11]

Fiedler eigenvalue and eigenvector

In such a scenario, the second smallest eigenvalue (λ_2) of L, yields a *lower bound* on the optimal cost (c) of ratio-cut partition with $c \geq \frac{\lambda_2}{n}$. The eigenvector ($V_2$) corresponding to λ_2, called the *Fiedler vector*, bisects the graph into only two communities based on the ***sign*** *of the corresponding vector entry*. Division into a larger number of communities can be achieved by repeated *bisection* or by using *multiple eigenvectors* corresponding to the smallest eigenvalues.[12] The examples in Figures 1,2 illustrate the spectral bisection approach.

Modularity and ratio-cut

Minimum cut partitioning however fails when the number of communities to be partitioned, or the partition sizes are unknown. For instance, optimizing the cut size for free group sizes puts all vertices in the same community. Additionally, cut size may be the wrong thing to minimize since a good division is not just one with small number of edges between communities. This motivated the use of Modularity (Q)[13] as a metric to optimize a balanced graph

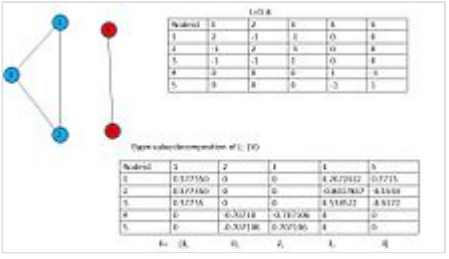

Figure 1: The graph *G* = (5,4) is analysed for spectral bisection. The linear combination of the smallest two eigenvectors leads to [1 1 1 1 1]' having an eigen value = 0.

partition. The example in Figure 3 illustrates 2 instances of the same graph such that in *(a)* modularity (Q) is the partitioning metric and in *(b)*, ratio-cut is the partitioning metric.

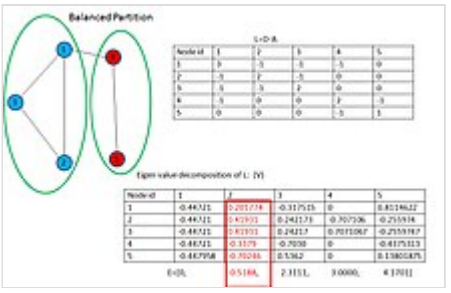

Figure 2: The graph *G* = (5,5) illustrates that the Fiedler vector in red bisects the graph into two communities, one with vertices {1,2,3} with positive entries in the vector space, and the other community has vertices {4,5} with negative vector space entries.

Applications

Conductance

Another objective function used for graph partitioning is Conductance which is the ratio between the number of cut edges and the volume of the smallest part. Conductance is related to electrical flows and random walks. The Cheeger bound guarantees that spectral bisection provides partitions with nearly optimal conductance. The quality of this approximation depends on the second smallest eigenvalue of the Laplacian λ_2.

Immunization

Graph partition can be useful for identifying the minimal set of nodes or links that should be immunized in order to stop epidemics.[14]

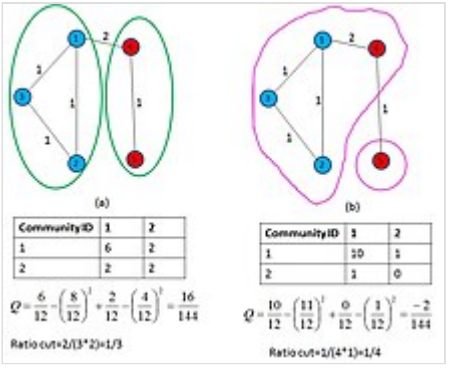

Figure 3: Weighted graph *G* may be partitioned to maximize *Q* in (a) or to minimize the ratio-cut in (b). We see that (a) is a better balanced partition, thus motivating the importance of modularity in graph partitioning problems.

Other graph partition methods

Spin models have been used for clustering of multivariate data wherein similarities are translated into coupling strengths.[15] The properties of ground state spin configuration can be directly interpreted as communities. Thus, a graph is partitioned to minimize the Hamiltonian of the partitioned graph. The Hamiltonian (H) is derived by assigning the following partition rewards and penalties.

- Reward internal edges between nodes of same group (same spin)
- Penalize missing edges in same group
- Penalize existing edges between different groups
- Reward non-links between different groups.

Additionally, Kernel-PCA-based Spectral clustering takes a form of least squares Support Vector Machine framework, and hence it becomes possible to project the data entries to a kernel induced feature space that has maximal variance, thus implying a high separation between the projected communities.[16]

Some methods express graph partitioning as a multi-criteria optimization problem which can be solved using local methods expressed in a game theoretic framework where each node makes a decision on the partition it chooses.[17]

For very large-scale distributed graphs classical partition methods might not apply (e.g., spectral partitioning, Metis[7]) since they require full access to graph data in order to perform global operations. For such large-scale scenarios distributed graph partitioning is used to perform partitioning through asynchronous local operations only.

Software tools

scikit-learn implements spectral clustering with the partitioning determined from eigenvectors of the graph Laplacian matrix for the original graph computed by ARPACK, or by LOBPCG solver with multigrid preconditioning.[9]

METIS[7] is a graph partitioning family by Karypis and Kumar. Among this family, kMetis aims at greater partitioning speed, hMetis,[8] applies to hypergraphs and aims at partition quality, and ParMetis[7] is a parallel implementation of the Metis graph partitioning algorithm.

KaHyPar[18][19][20] is a multilevel hypergraph partitioning framework providing direct k-way and recursive bisection based partitioning algorithms. It instantiates the multilevel approach in its most extreme version, removing only a single vertex in every level of the hierarchy. By using this very fine grained *n*-level approach combined with strong local search heuristics, it computes solutions of very high quality.

Scotch[21] is graph partitioning framework by Pellegrini. It uses recursive multilevel bisection and includes sequential as well as parallel partitioning techniques.

Jostle[22] is a sequential and parallel graph partitioning solver developed by Chris Walshaw. The commercialized version of this partitioner is known as NetWorks.

Party[23] implements the Bubble/shape-optimized framework and the Helpful Sets algorithm.

The software packages DibaP[24] and its MPI-parallel variant PDibaP[25] by Meyerhenke implement the Bubble framework using diffusion; DibaP also uses AMG-based techniques for coarsening and solving linear systems arising in the diffusive approach.

Sanders and Schulz released a graph partitioning package KaHIP[26] (Karlsruhe High Quality Partitioning) that implements for example flow-based methods, more-localized local searches and several parallel and sequential meta-heuristics.

The tools Parkway[27] by Trifunovic and Knottenbelt as well as Zoltan[28] by Devine et al. focus on hypergraph partitioning.

References

1. Andreev, Konstantin; Räcke, Harald (2004). "Balanced graph partitioning". *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. Barcelona, Spain. pp. 120–124. CiteSeerX 10.1.1.417.8592 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.417.8592). doi:10.1145/1007912.1007931 (https://doi.org/10.1145%2F1007912.1007931). ISBN 978-1-58113-840-5.

2. Buluc, Aydin; Meyerhenke, Henning; Safro, Ilya; Sanders, Peter; Schulz, Christian (2013). "Recent Advances in Graph Partitioning". arXiv:1311.3144 (https://arxiv.org/abs/1311.3144) [cs.DS (https://arxiv.org/archive/cs.DS)].
3. Feldmann, Andreas Emil; Foschini, Luca (2012). "Balanced Partitions of Trees and Applications". *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science*: 100–111.
4. Feldmann, Andreas Emil (2012). "Fast Balanced Partitioning is Hard, Even on Grids and Trees". *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*. arXiv:1111.6745 (https://arxiv.org/abs/1111.6745). Bibcode:2011arXiv1111.6745F (https://ui.adsabs.harvard.edu/abs/2011arXiv1111.6745F).
5. Garey, Michael R.; Johnson, David S. (1979). *Computers and intractability: A guide to the theory of NP-completeness* (https://archive.org/details/computersintract0000gare). W. H. Freeman & Co. ISBN 978-0-7167-1044-8.
6. Hendrickson, B.; Leland, R. (1995). *A multilevel algorithm for partitioning graphs*. Proceedings of the 1995 ACM/IEEE conference on Supercomputing. ACM. p. 28.
7. Karypis, G.; Kumar, V. (1999). "A fast and high quality multilevel scheme for partitioning irregular graphs". *SIAM Journal on Scientific Computing*. **20** (1): 359. CiteSeerX 10.1.1.39.3415 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.3415). doi:10.1137/S1064827595287997 (https://doi.org/10.1137%2FS1064827595287997). S2CID 3628209 (https://api.semanticscholar.org/CorpusID:3628209).
8. Karypis, G.; Aggarwal, R.; Kumar, V.; Shekhar, S. (1997). *Multilevel hypergraph partitioning: application in VLSI domain*. Proceedings of the 34th annual Design Automation Conference. pp. 526–529.
9. Knyazev, Andrew V. (2006). *Multiscale Spectral Graph Partitioning and Image Segmentation*. Workshop on Algorithms for Modern Massive Data Sets Stanford University and Yahoo! Research.
10. J. Demmel, [1] (https://people.eecs.berkeley.edu/~demmel/cs267/lecture20/lecture20.html), CS267: Notes for Lecture 23, April 9, 1999, Graph Partitioning, Part 2
11. Knyazev, Andrew (2018). *On spectral partitioning of signed graphs*. Eighth SIAM Workshop on Combinatorial Scientific Computing, CSC 2018, Bergen, Norway, June 6–8. arXiv:1701.01394 (https://arxiv.org/abs/1701.01394). doi:10.1137/1.9781611975215.2 (https://doi.org/10.1137%2F1.9781611975215.2).
12. Naumov, M.; Moon, T. (2016). "Parallel Spectral Graph Partitioning" (https://research.nvidia.com/publication/parallel-spectral-graph-partitioning). *NVIDIA Technical Report*. nvr-2016-001.
13. Newman, M. E. J. (2006). "Modularity and community structure in networks" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1482622). *PNAS*. **103** (23): 8577–8696. arXiv:physics/0602124 (https://arxiv.org/abs/physics/0602124). Bibcode:2006PNAS..103.8577N (https://ui.adsabs.harvard.edu/abs/2006PNAS..103.8577N). doi:10.1073/pnas.0601602103 (https://doi.org/10.1073%2Fpnas.0601602103). PMC 1482622 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1482622). PMID 16723398 (https://pubmed.ncbi.nlm.nih.gov/16723398).
14. Y. Chen, G. Paul, S. Havlin, F. Liljeros, H.E. Stanley (2009). "Finding a Better Immunization Strategy". *Phys. Rev. Lett*. **101** (5): 058701. doi:10.1103/PhysRevLett.101.058701 (https://doi.org/10.1103%2FPhysRevLett.101.058701). PMID 18764435 (https://pubmed.ncbi.nlm.nih.gov/18764435).
15. Reichardt, Jörg; Bornholdt, Stefan (July 2006). "Statistical mechanics of community detection". *Phys. Rev. E*. **74** (1): 016110. arXiv:cond-mat/0603718 (https://arxiv.org/abs/cond-mat/0603718). Bibcode:2006PhRvE..74a6110R (https://ui.adsabs.harvard.edu/abs/2006PhRvE..74a6110R). doi:10.1103/PhysRevE.74.016110 (https://doi.org/10.1103%2FPhysRevE.74.016110). PMID 16907154 (https://pubmed.ncbi.nlm.nih.gov/16907154). S2CID 792965 (https://api.semanticscholar.org/CorpusID:792965).

16. Alzate, Carlos; Suykens, Johan A. K. (2010). "Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **32** (2): 335–347. doi:10.1109/TPAMI.2008.292 (https://doi.org/10.1109%2FTPAMI.2008.292). ISSN 0162-8828 (https://search.worldcat.org/issn/0162-8828). PMID 20075462 (https://pubmed.ncbi.nlm.nih.gov/20075462). S2CID 200488 (https://api.semanticscholar.org/CorpusID:200488).
17. Kurve, A.; Griffin, C.; Kesidis G. (2011) "A graph partitioning game for distributed simulation of networks", *Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks*: 9–16
18. Schlag, S.; Henne, V.; Heuer, T.; Meyerhenke, H.; Sanders, P.; Schulz, C. (2015-12-30). "K-way Hypergraph Partitioning via n-Level Recursive Bisection". *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics. pp. 53–67. arXiv:1511.03137 (https://arxiv.org/abs/1511.03137). doi:10.1137/1.9781611974317.5 (https://doi.org/10.1137%2F1.9781611974317.5). ISBN 978-1-61197-431-7. S2CID 1674598 (https://api.semanticscholar.org/CorpusID:1674598).
19. Akhremtsev, Y.; Heuer, T.; Sanders, P.; Schlag, S. (2017-01-01). "Engineering a direct k-way Hypergraph Partitioning Algorithm". *2017 Proceedings of the Nineteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics. pp. 28–42. doi:10.1137/1.9781611974768.3 (https://doi.org/10.1137%2F1.9781611974768.3). ISBN 978-1-61197-476-8.
20. Heuer, Tobias; Schlag, Sebastian (2017). "Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure". In Iliopoulos, Costas S.; Pissis, Solon P.; Puglisi, Simon J.; Raman, Rajeev (eds.). *16th International Symposium on Experimental Algorithms (SEA 2017)*. Leibniz International Proceedings in Informatics (LIPIcs). Vol. 75. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. pp. 21:1–21:19. doi:10.4230/LIPIcs.SEA.2017.21 (https://doi.org/10.4230%2FLIPIcs.SEA.2017.21). ISBN 978-3-95977-036-1.
21. Chevalier, C.; Pellegrini, F. (2008). "PT-Scotch: A Tool for Efficient Parallel Graph Ordering". *Parallel Computing*. **34** (6): 318–331. arXiv:0907.1375 (https://arxiv.org/abs/0907.1375). doi:10.1016/j.parco.2007.12.001 (https://doi.org/10.1016%2Fj.parco.2007.12.001). S2CID 10433524 (https://api.semanticscholar.org/CorpusID:10433524).
22. Walshaw, C.; Cross, M. (2000). "Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm". *SIAM Journal on Scientific Computing*. **22** (1): 63–80. Bibcode:2000SJSC...22...63W (https://ui.adsabs.harvard.edu/abs/2000SJSC...22...63W). CiteSeerX 10.1.1.19.1836 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.1836). doi:10.1137/s1064827598337373 (https://doi.org/10.1137%2Fs1064827598337373).
23. Diekmann, R.; Preis, R.; Schlimbach, F.; Walshaw, C. (2000). "Shape-optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM". *Parallel Computing*. **26** (12): 1555–1581. CiteSeerX 10.1.1.46.5687 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.5687). doi:10.1016/s0167-8191(00)00043-0 (https://doi.org/10.1016%2Fs0167-8191%2800%2900043-0).
24. Meyerhenke, H.; Monien, B.; Sauerwald, T. (2008). "A New Diffusion-Based Multilevel Algorithm for Computing Graph Partitions". *Journal of Parallel Computing and Distributed Computing*. **69** (9): 750–761. CiteSeerX 10.1.1.702.7275 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.702.7275). doi:10.1016/j.jpdc.2009.04.005 (https://doi.org/10.1016%2Fj.jpdc.2009.04.005). S2CID 9755877 (https://api.semanticscholar.org/CorpusID:9755877).
25. Meyerhenke, H. (2013). *Shape Optimizing Load Balancing for MPI-Parallel Adaptive Numerical Simulations*. 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering. pp. 67–82.

26. Sanders, P.; Schulz, C. (2011). *Engineering Multilevel Graph Partitioning Algorithms*. Proceedings of the 19th European Symposium on Algorithms (ESA). Vol. 6942. pp. 469–480.
27. Trifunovic, A.; Knottenbelt, W. J. (2008). "Parallel Multilevel Algorithms for Hypergraph Partitioning". *Journal of Parallel and Distributed Computing*. **68** (5): 563–581. CiteSeerX 10.1.1.641.7796 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.641.7796). doi:10.1016/j.jpdc.2007.11.002 (https://doi.org/10.1016%2Fj.jpdc.2007.11.002).
28. Devine, K.; Boman, E.; Heaphy, R.; Bisseling, R.; Catalyurek, Ü. (2006). *Parallel Hypergraph Partitioning for Scientific Computing*. Proceedings of the 20th International Conference on Parallel and Distributed Processing. p. 124.

Further reading

- Bichot, Charles-Edmond; Siarry, Patrick (2011). *Graph Partitioning: Optimisation and Applications*. ISTE – Wiley. ISBN 978-1-84821-233-6.