B

Benchmarking for Graph Clustering and Partitioning

David A. Bader[1], Andrea Kappes[2], Henning Meyerhenke[2], Peter Sanders[2], Christian Schulz[2] and Dorothea Wagner[2]
[1]School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[2]Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Synonyms

Algorithm evaluation; Graph repository; Test instances.

Glossary

Benchmarking	Performance evaluation for comparison to the state of the art.
Benchmark suite	Set of instances used for benchmarking.

Definition

Benchmarking refers to a repeatable performance evaluation as a means to compare somebody's work to the state of the art in the respective field. As an example, benchmarking can compare the computing performance of new and old hardware.

In the context of computing, many different benchmarks of various sorts have been used. A prominent example is the Linpack benchmark of the TOP500 list of the fastest computers in the world, which measures the performance of the hardware by solving a dense linear algebra problem. Different categories of benchmarks include sequential versus parallel, microbenchmark versus application, or fixed code versus informal problem description. See, e.g., (Weicker 2002) for a more detailed treatment of hardware evaluation.

When it comes to benchmarking algorithms for network analysis, typical measures of interest are solution quality and running time. The comparison process requires the establishment of widely accepted *benchmark instances* on which the algorithms have to compete. In the course of the 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering (Bader et al. 2016), we have assembled a suite of graphs and graph generators intended for comparing graph algorithms with each other. While our particular focus has been on assembling instances for benchmarking graph partitioning and graph clustering algorithms, we believe the suite to be useful for related fields as well. This includes the broad field of network analysis (which includes graph clustering, also known as *community detection*) and various combinatorial problems.

The purpose of DIMACS Implementation Challenges is to assess the practical performance

R. Alhajj, J. Rokne (eds.), *Encyclopedia of Social Network Analysis and Mining*,
DOI 10.1007/978-1-4614-7163-9_23-1

of algorithms, in particular in problem settings where worst case and probabilistic analysis yield unrealistic results. Where analysis fails, experimentation can provide insights into realistic algorithm performance. By evaluating different implementations on the assembled benchmark suite, the challenges create a reproducible picture of the state of the art in the area under consideration. This helps to foster an effective technology transfer within the research areas of algorithms, data structures, and implementation techniques as well as a transfer back to the original applications.

Introduction

Graph partitioning and graph clustering (or community detection) are ubiquitous subtasks in many application areas. Generally speaking, both techniques aim at the identification of vertex subsets (*clusters*) with many internal and few external edges. In this work we concentrate our description on aspects important to the field of network analysis, in particular on community detection.

In its most general form, community detection does not require a fixed number k of clusters nor constraints on the size of the clusters. Instead, a quality function which measures both, the density inside clusters and the sparseness between them, is used. A variety of such functions has been proposed, among which the measure *modularity* has proven itself fairly reliable and largely in accordance with human intuition in the literature:

Problem 1 (Modularity Maximization). Given an undirected, weighted graph $G = (V, E, \omega)$ without parallel edges, find a partition C of V which optimizes the modularity objective function

$$Q(c) := \sum_{C \in c} \frac{\sum_{\{u,v\} \in E \atop u,v \in C} \omega(\{u,v\})}{\sum_{e \in E} \omega(e)} - \frac{\left(\sum_{v \in C} S(v)\right)^2}{4\left(\sum_{e \in E} \omega(e)\right)^2}.$$

Here we assume $e = \{u, v\} \in E$ is a multiset (i.e., a self-loop $u = v$ is allowed) and that the strength s(v) of a node v is the sum of the weights of its incident edges. Recently some criticism towards modularity has emerged. Fortunato and Barthelemy (Fortunato and Barthelemy 2007) demonstrate that global modularity optimization cannot distinguish between a single community and a group of smaller communities. Berry et al. (Berry et al. 2011) provide a weighting mechanism and Renaud a coarse-graining parametrization (Lambiotte, 2010) that both somewhat alleviate the resolution limit drawback. Yet, other problems remain (Good et al. 2010; Lancichinetti and Fortunato 2011). That is one reason why the 10th DIMACS Implementation Challenge had a second graph clustering category apart from modularity maximization. In this second category, algorithms are compared with respect to four different objective functions, with the goal to explicitly invite clustering algorithms that are not based on a specific objective function. These objective functions are based upon performance (van Dongen 2000), intracluster density, and intercluster conductance and expansion (Kannan et al. 2004).

In contrast to graph clustering, the term graph partitioning usually implies that the number of partitions is fixed and the task is to partition the vertex set into blocks of (almost) equal size. Its main applications do not belong to network analysis; a very popular one is the preprocessing of data for parallel computing. The objective functions used for the partitioning subchallenges are the number of edges between the blocks and the maximum communication volume (Çatalyürek and Aykanat 1996) of the partition.

Participants of the challenge were invited to submit solutions to the different challenge categories on graph partitioning and graph clustering. This way different algorithms and implementations were tested against the benchmark instances. Thereby future researchers are enabled to identify techniques that are most effective for a respective partitioning or clustering problem – by using our benchmark set and by comparing their results to the challenge results and to those in the literature published afterwards. In this chapter we describe the benchmark suite and its assembly process. Moreover, we sketch some of the results obtained by the challenge participants using the benchmark graphs.

Key Points

Collecting the instances for the benchmark suite was performed with two main aspects in mind, diversity of source applications and diversity of instance sizes. Moreover, some graphs have been frequently used in previous work, whereas others are new or fairly recent. Some instances are based on real-world inputs, while others have been created using a generator. The generated graphs also vary in how closely they resemble real-world counterparts.

The solutions generated by the challenge participants using the benchmark suite constitute a valuable picture of the state of the art in graph partitioning and graph clustering. To better suit algorithms that do not explicitly optimize a traditional objective function (and to circumvent known flaws in these traditional objective functions), additional criteria to assess the quality of the submitted clusterings were evaluated.

Moreover, a nondiscriminatory way to assign scores to solvers that takes both running time and solution quality into account was used.

Key Applications

The collection can be widely used by the community for the development and performance evaluation of graph algorithms. More precisely, the provided collection of instances supports the execution and evaluation of a large number of experiments needed to tune parameters and to evaluate the performance of graph algorithms. At the same time, the collection allows robust and repeatable experiments since all instances have been long-term archived with public access (Bader et al. 2016).

Historical Background

Previous DIMACS Implementation Challenges addressed a large variety of algorithmic problems, several of them involving graphs and networks. Graph repositories similar to our benchmark suite exist as well. However, they often lack the size, broadness of source applications, and connection to a quality-driven competition.

An example repository widely used in combinatorial scientific computing is Chris Walshaw's graph partitioning archive (Soper et al. 2004). It stores 34 graphs and the best known graph partitions computed for these graphs. This archive has substantially simplified the improvement of graph partitioning algorithms over the last 15 years. Today, however, the instances contained therein have to be deemed rather small and also somewhat limited in terms of application areas. For example, there are no social networks contained in this archive.

The University of Florida Sparse Matrix Collection (Davis 2016), maintained by Tim Davis, is broader in terms of application areas and matrix sizes. Although social networks have been included recently as well, most matrices stem from technical applications.

Graph collections focusing on scale-free graphs such as social networks do exist, e.g., Arenas (2012) and Newman (2012). Two more recent but also prominent examples are the Stanford Large Network Dataset Collection SNAP (Leskovec 2016) and the Koblenz Network Collection (KONECT) (Kunegis 2013). In most cases these existing collections lack a significant comparison of how a larger number of different algorithms perform on the data – at least regarding inexact solutions to complex problems such as graph clustering and partitioning.

Proposed Solution and Methodology

With the 10th DIMACS Implementation Challenge and its graph collection we addressed both of these issues. Our collection contains more than 100 graphs of various origins and assembled in different categories. (In addition we link to the Walshaw archive and the University of Florida Sparse Matrix Collection.) We took care that our collection contains instances best suited for partitioning in technical applications as well as instances particularly intended for clustering and related network analysis tasks. Additionally the challenge results provide guidance as to how

different algorithms perform on different classes of graphs.

The driving considerations during the assembly of the graph collection were to include a sufficiently large variety of application sources (thereby instance structures) and graph sizes. In that line of reasoning we identified three higher-level classes from which to select: purely random graphs, generated graphs with close resemblance to data from real-world applications, and actual real-world data. Our intent is to offer a good diversity in order to provide a meaningful benchmark for network analysis and graph partitioning algorithms.

With generators at hand, an experimenter can scale to (nearly) arbitrary graph sizes, retaining the general structure of the graphs while increasing their sizes. This is, for example, important when performing weak scaling studies for the experimental analysis of parallel algorithms. It is worth mentioning that, since the instance sizes are only limited by architectural constraints, generators provide a means to "grow" instance sizes with future architectural improvements. This way the current state of a collection does not age as quickly as without generators. More details on the graphs generated to resemble real-world inputs follow below.

Generated random graphs offer an additional benefit. They are usually easier to analyze with theoretical methods than other graph types. The Erdős-Rényi (ER) model, for example, has experienced significant consideration in theoretical works. Many important properties of these graphs were proved in this long course of research, see, e.g., Bollobás (Bollobás 1985). Due to the lack of resemblance of typical ER graphs to real-world inputs, an active line of research is developing alternative models. The random graphs we included and their generators (as well as some recent developments) are described later in this section in more detail.

Finally, real-world graphs add the necessary confidence that an algorithm's performance in terms of running time and quality on the collection resembles its performance on the represented real-world applications. The real-world graphs we included are also described below.

In the remainder of this section, we first explain the preprocessing performed to unify the instances in our collection. After that, the individual categories of the collection are described.

Preprocessing

Graph partitioning and, with some exceptions, also graph clustering are usually applied to undirected graphs. A common preprocessing step is therefore to *symmetrize* the graph prior to partitioning, i.e., to make the graph undirected by including an undirected edge between two vertices a and b if and only if there exists an edge from a to b or from b to a. If both directions are present in the original graph, there are several possibilities to assign a weight to the resulting undirected edge. We chose the following approach: if the input graph is unweighted, an edge between two vertices is considered as the information that they are related in some sense, independent of the strength of this connection. Hence, if an edge in both directions exists, it is translated into an unweighted, undirected edge. On the other hand, if the input graph is weighted, we add up the edge weights of both directions, as the connection between the vertices is typically stronger if both directions exist. Analogously, the weight of parallel edges is summed up only in case of weighted networks, otherwise parallel edges are removed. Self-loops are always removed (with the exception of some synthetic Kronecker graphs, for which versions with self-loops and parallel edges and versions without exist). Only a handful of the real-world networks included in the benchmark contain (few) parallel edges and self-loops; therefore, this decision does not alter the structure of the networks considerably.

Furthermore, the graph format used in the benchmark is a slight extension of the format that some well-established partitioners such as Metis (Karypis and Kumar 1999) use. This format supports only integer edge weights, but some of the real-world graphs use fractional weights in very different orders of magnitude. It would have been possible to define an extension of this format to allow for fractional weight. However, this might have prevented some solvers to enter

the challenge. Multiplying the edge weights by a suitable power of 10 to get integer weights would have been another approach. Yet, as the edge weights are of very different ranges, each graph would have needed its own normalization and without rounding, the resulting edge weights could be too large to fit in standard integer types. Although we are aware that this causes a loss of information, we felt that the neatest solution was to make the respective graphs unweighted. One of the benchmark graphs (cond-mat-2005) originally contains edges with a weight inf. As their meaning is not clear and none of the objective function is well-defined in case of infinite weights, we discarded these edges, as well as all edges with an edge weight of 0.

Random Graphs

The *Erdős-Rényi random graph* generator in the collection creates graphs according to the well-known $G(n, p)$ model (which is very similar to the original model proposed by Erdos and Renyi, but was actually devised by Gilbert [Gilbert 1959]). The included graphs have been generated with $p = \frac{1.5 \ln n}{n}$, where the value of p is chosen with the intent to obtain connected graphs with high probability. This class of graphs is well-studied in theory. It is also known that typical Erdös-Rényi graphs do not resemble real-world graphs. The class was included nevertheless for its theoretical importance and due to the easy generation of large graphs with high average degree.

The graphs in the category Kronecker are generated using the Graph500 benchmark (Bader et al. 2010). This benchmark's purpose is to measure the performance of computer systems when processing graph-structured workloads. More specifically, our instances are derived from an R-MAT generator which is part of the benchmark. R-MAT graphs (Chakrabarti et al. 2004) are generated by sampling from a perturbed Kronecker product. They are scale-free and reflect many properties of real social networks. All files have been generated with the R-MAT parameters A = 0.57, B = 0.19, C = 0.19, and D = 0.05 and edge factor 48, i.e., the number of edges equals $48n$, where n is the number of vertices. The original Kronecker files contain self-loops and multiple edges. These properties are also present in real-world data sets. However, as some tools cannot handle these "artifacts," we present "cleansed" versions of the data sets (yielding simple graphs) as well.

Delaunay and random geometric graphs are taken from KaPPa (Holtgrewe et al. 2010) (Karlsruhe Parallel Partitioner). Here, *rggX* is a *random geometric graph* with 2^X nodes. Each node represents a random point in the unit square, and edges connect nodes whose Euclidean distance is below 0.55y $\sqrt{\ln n / n}$. This threshold is chosen in order to ensure that the graph is almost connected. The graph *DelaunayX* is the Delaunay triangulation of 2^X random points in the unit square.

Other noteworthy generative models not or hardly included in the benchmark set are Barabasi-Albert (Albert and Barabási 2002), BTER (Seshadhri et al. 2012), Chung-Lu (Aiello et al. 2001), Dorogovtsev-Mendes (Dorogovtsev and Mendes 2003), and random hyperbolic graphs (Krioukov et al. 2010). We mention them here to give a more comprehensive overview. The LFR model (Lancichinetti et al. 2008) is even particularly designed for benchmarking graph clustering algorithms. The implementation accepts, among others, vertex degrees and community sizes as parameters and generates a clustered graph accordingly.

Most graphs from the above models can be generated with the network analysis toolkit NetworKit 4.0 (Staudt et al. 2015) – except for BTER and the two generators available in KaPPa. BTER also has a public implementation (Kolda et al. 2013). For some models there exist further improved implementations. For Barabasi-Albert, for example, Meyer and Penschuck proposed an external-memory and GPGPU variant (Meyer and Penschuck 2016), which was later improved and implemented in distributed memory by Sanders and Schulz (Sanders and Schulz 2016). Also the Chung-Lu model has a parallel generator for distributed memory (Alam and Khan 2017). The random hyperbolic graph generator by von Looz et al. (von Looz et al. 2015), in turn, is only shared-memory parallel but also capable of generating billions of edges quickly.

Generated Graphs with Real-World Structure

Each graph in the star mixture section of the benchmark represents a star-like structure of different graphs $S_0,\ldots, S_t$. Here the graphs $S_1,\ldots, S_t$ are weakly connected to the center S_0 by random edges. The total number of random edges added between each S_i and S_0 is less than 3% out of the total number of edges in S_i. The graphs are mixtures of the following structures: social networks, finite-element graphs, VLSI chips, peer-to-peer networks, and matrices from optimization solvers. These graphs were submitted by Safro et al. and are included into the benchmark because they are potentially hard graphs for graph partitioning. A similar motivation is pursued by Gutfraind et al.'s multiscale generator Musketeer, which takes a real-world graph and obfuscates it by random changes.

Two classic random models of social networks are *preferential attachment* (Barabási and Albert 1999) and *small world* (Watts and Strogatz 1998). In the context of graph clustering, *planted partition* or $G(n, p_{in}, p_{out})$ graphs are frequently used to validate algorithms (Lancichinetti and Fortunato 2009). These networks do not exhibit common properties of real-world social networks like a power-law degree distribution (which can be provided by the aforementioned LFR generator). However, their use is typically motivated by the knowledge of a ground-truth clustering that is used in the generation process and can be used to compare algorithms independent of specific objective functions. We included one graph of each category in the benchmark set as we deemed it interesting to see to what extent algorithmic behavior on these graphs coincides with the behavior on real-world data. Since the creation of the original benchmark set, LFR has become another established generator for such ground-truth data, at least for smaller networks, and we recommend its use as well.

Although we do not store dynamic graphs, three so-called *frames* (static instances within the same dynamic sequence) from three dynamic mesh sequences each are included in our collection. These sequences resemble two-dimensional adaptive numerical simulations. The generator is explained in some detail by Marquardt and Schamberger.

Computational task graphs model temporal dependencies between tasks to be solved, here for applications working on data streams. The generated graphs can be used for performance analysis of algorithms and the development of improved hardware parameters. These graphs have been submitted by Ajwani et al.

Real-World Graphs

The benchmark includes a large number of real-world networks stemming from many different applications. Since scientific computing is a major application area using graph partitioning, we included graphs that have been used in *numerical simulations* of various kinds.

The partitioning of *road networks* is an important technique when it comes to preprocessing for shortest path algorithms (Bauer et al. 2010). The graphs that can be found in this section are road networks from whole continents, e.g., Europe, as well as from whole countries, e.g., Germany. These graphs were submitted by Kobitzsch and are based on data from the Open Street Map project.

Parallel direct methods for solving linear systems yield another important application of graph partitioning. We therefore included a subset of graph representations of matrices from the University of Florida Sparse Matrix Collection (Davis 2016).

In the context of graph clustering, the analysis of social networks is one of the most important applications. The part of the benchmark suite especially addressed to clustering algorithms reflects this by including a variety of real-world social networks. Most of these are taken from the webpages of Newman (2012) and Arenas (2012) and have been previously used to compare and evaluate clustering in the context of modularity-maximization.

A special subcategory of social networks are *coauthorship networks*. In a scientific context, coauthorship networks link scientists that have coauthored at least one publication. The DIMACS Benchmark includes coauthorship graphs from the field of astrophysics, condensed matter and high-

energy theory, network science, and computer and information science. Closely related to these are *copaper* and *citation* networks. Copaper graphs are compiled analogously to coauthorship graphs by linking papers if they share at least one author. In contrast to that, citation networks link papers with another if one cites the other. The benchmark set contains graphs of both kind based on publications in computer and information science.

Graph clustering has also been successfully applied to *web graphs*, where edges link webpages based on hyperlinks. A subset of the web graphs we included are gathered by the Laboratory of Web Algorithms in Milano by domain-wise crawls performed between 2000 and 2007. In the context of the challenge, these networks are particularly interesting due to their size; in fact, the graph combining 12 monthly snapshots of the .uk domain comprises over 3 billion edges, which makes it the largest network in the whole benchmark set.

Apart from these, the part of the benchmark set explicitly addressed to clustering contains a variety of (mainly) small networks from various application areas such as biology and political science. All of these are well-known in the modularity-based clustering community. For details on particular networks and references, we refer to the challenge webpage (Bader et al. 2016).

The graphs in the *redistricting* category represent US states. They are used for solving the redistricting problem, i.e., determining new electoral boundaries, for example, due to population changes. Each node represents a block from the 2010 census. Two nodes share an edge if their blocks are adjacent.

Illustrative Example

As running times would have been prohibitive for the whole set of benchmark instances, participants of the competition were only required to submit clusterings for a subset of instances, the final challenge testbed. This subset was announced 2 weeks before the deadline. To illustrate the performance of different algorithms on graphs from different categories, Table 1 shows the best modularity values achieved by the submitted solvers on the final challenge testbed. From the 15 solvers in this category, two clearly lead the field. CGGCi_RG (Ovelgönne and Geyer-Schulz 2012) iteratively combines several high-quality clusterings to find a solution with higher quality. In contrast to that, VNS (Aloise et al. 2012) uses the metaheuristic *variable neighborhood search*, a variant of local search. With few exceptions, VNS achieves the best results on networks with up to approximately 100, 000 vertices, but is outperformed by CGGi_RG in larger networks. An interesting observation is that ParMod (Çatalyürek et al. 2012), a technique based on recursive bipartitions, attains the best modularity values on two graphs. Neither the size nor the density of these graphs is exceptional, but unlike the majority of graphs used for this competition, they exhibit a mesh-like structure.

In addition to quality, running time is also an important aspect when choosing an algorithm for a certain application. This is why the DIMACS Challenge included a second subchallenge for each objective function, where both quality and speed contributed to the final scores. More specifically, the scoring is based on the *Pareto Count* of a submitted algorithm on an instance, i.e., the number of competing algorithms that are both faster and achieve a higher quality. In this category, a relatively fast agglomerative solver named RG (Ovelgönne and Geyer-Schulz 2012) obtained the best scores. While the differences in running time might not seem very important in the context of small instances, they were in fact huge on larger instances. Considering, for example, the raw running times on the webgraph uk-2002, RG needs approximately 13 min to compute a clustering, which is more than 600 times faster than the running time of CGGCi_RG, while the difference in modularity is less than 0.001. This running time can be further improved by using parallel algorithms. For example, one of the submissions is able to cluster this instance in only 30 s by using

Benchmarking for Graph Clustering and Partitioning, Table 1 Best modularity scores achieved by challenge participants on the challenge testbed

Graph	Modularity	Solver
as-22july06	0.678267	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
astro-ph	0.744621	VNS (Aloise et al. 2012)
audikw l	0.917983	VNS (Aloise et al. 2012)
belgium.osm	0.994940	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
cage15	0.903173	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
caidaRouterLevel	0.872042	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
celegans_metabolic	0.453248	VNS (Aloise et al. 2012)
citationCiteseer	0.823930	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
coAuthorsCiteseer	0.905297	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
cond-mat-2005	0.746254	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
coPapersDBLP	0.866794	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
email	0.582829	VNS (Aloise et al. 2012)
er-factl.5-scale25	0.077934	comm-el (Jason Riedy et al. 2012)
eu-2005	0.941554	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
G_n_pin_pout	0.500098	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
in-2004	0.980622	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
kron_g500-s-lognl6	0.065056	VNS (Aloise et al. 2012)
kron_g500-s-logn20	0.050350	CGGC_RG (Ovelgönne and Geyer-Schulz 2012)
ldoor	0.969370	ParMod (Çatalyürek et al. 2012)
luxembourg.osm	0.989621	VNS (Aloise et al. 2012)
memplus	0.700473	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
PGPgiantcompo	0.886564	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
polblogs	0.427105	VNS (Aloise et al. 2012)
power	0.940851	VNS (Aloise et al. 2012)
preferentialAttachment	0.315994	VNS (Aloise et al. 2012)
rgg_n_2_17_s0	0.978324	VNS (Aloise et al. 2012)
smallworld	0.793042	VNS (Aloise et al. 2012)
uk-2002	0.990301	CGGCi_RG (Ovelgönne and Geyer-Schulz 2012)
uk-2007-05	0.480210	comm-el-xmt2 (Jason Riedy et al. 2012)
333SP	0.989095	ParMod (Çatalyürek et al. 2012)

a GPU (Fagginger Auer and Bisseling 2012), with a modularity that is still larger than 0.97.

Consequently, the question which algorithm is the "best" cannot always be answered globally. Instead, the answer often depends on application-specific parameters like the size and structure of certain instances, as well as the available hardware and a custom trade-off between quality and running time. Comparing the results of different algorithms on various benchmark instances can assist the choice of an appropriate algorithm.

The results of the DIMACS Challenge have initiated new algorithmic work in the area of parallel computing. We choose to highlight two parallel modularity-driven community detection algorithms, PLM (Staudt and Meyerhenke 2016) and Nerstrand (LaSalle and Karypis 2015). They currently offer the best trade-off between running time and solution quality on commodity hardware, with Nerstrand being slightly faster. The largest graph in the benchmark set, the web graph uk-2007-05 with 3.3 billion edges, can be clustered by these two in less than 3 min on a shared-memory server with a modularity value above 0.99.

Future Directions

With the graph archive of the 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering we have introduced a comprehensive collection of graphs that can be used for the assessment of graph partitioning and network analysis algorithms. With the archive we hope to simplify the development of improved solution techniques in these areas by allowing algorithm engineers to compare the performance of their implementations to the state of the art.

A deliberate limitation is to not consider dynamic graphs, directed graphs, nor hypergraphs. As instances of this type were not considered in the challenge, they were not included in the collection, either. We do consider all these omitted graph types useful, though, and they represent interesting applications.

Of particular interest for network analysis are directed and dynamic graphs. There is no lack of data (albeit not all of them are publicly accessible). As an example, the dynamic interaction of social network users over time constitutes a dynamic graph that is of particular interest to social media enterprises and online marketers. When compiling dynamic instances into a collection, it should be considered that dynamic graphs are more difficult to assemble or generate in a consistent way – issues such as a suitable interval length and space-saving storage formats arise.

We encourage interested colleagues to start a new collection using a similar methodology, this time focusing on the types of graphs we omitted. Such an effort would certainly be beneficial to the network analysis community.

Acknowledgements The authors would like to thank all contributors to the 10th DIMACS Implementation Challenge graph collection. Tim Davis provided valuable guidelines for preprocessing the data. Financial support by the sponsors DIMACS, the Command, Control, and Interoperability Center for Advanced Data Analysis (CCICADA), Pacific Northwest National Laboratory, Sandia National Laboratories, Intel Corporation, and Deutsche Forschungsgemeinschaft (DFG) is gratefully acknowledged.

Cross-References

- ▶ Clustering Algorithms
- ▶ Communities Discovery and Analysis in Online and Offline Social Networks: Link-Based Overlapping and Nonoverlapping Communities
- ▶ Community Detection, Current and Future Research Trends
- ▶ Community Discovery and Analysis in Large-Scale Online/Offline Social Networks
- ▶ Extracting and Inferring communities via Link Analysis
- ▶ Extracting Social Networks from Data
- ▶ Large Networks, Analysis of
- ▶ Simulated Datasets
- ▶ Social Network Datasets
- ▶ Sources of Network Data
- ▶ Synthetic Datasets

References

Aiello W, Chung F, Lu L (2001) A random graph model for power law graphs. Exp Math 10(1):53–66

Alam M, Khan M (2017) Parallel algorithms for generating random networks with given degree sequences. Int J Parallel Prog 45(1):109–127

Albert R, Barabási AL (2002) Statistical mechanics of complex networks. Rev Mod Phys 74(1):47

Aloise D, Caporossi G, Perron S, Hansen P, Liberti L, Ruiz M (2012) Modularity maximization in networks by variable neighborhood search. In: Proceedings of graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop, 2012. Contemporary mathematics, vol 588. American Mathematical Society

Arenas A. Network data sets. http://deim.urv.cat/~aarenas/data/welcome.htm. Online. Accessed 28 Sept 2012

Bader DA, Berry J, Kahan S, Murphy R, Jason Riedy E, Will-cock J (2010) Graph 500 benchmark 1 ("search"), version 1.1. Technical report, Graph 500

Bader D, Meyerhenke H, Sanders P, Wagner D (2012) 10th DIMACS implementation challenge. http://www.cc.gatech.edu/dimacs10/. Online. Accessed 17 Apr 2016

Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512

Bauer R, Delling D, Sanders P, Schieferdecker D, Schultes D, Wagner D (2010) Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. ACM J Exp Algorithmics 152.3:2.1–2.3:2.31

Berry JW, Hendrickson B, LaViolette RA, Phillips CA (2011) Tolerating the community detection resolution limit with edge weighting. Phys Rev E 83:056119

Bollobás B (1985) Random graphs. Academic Press, London

Çatalyürek ÜV, Aykanat C (1996) Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. In: Ferreira A, Rolim J, Saad Y, Yang T (eds) Parallel algorithms for irregularly structured problems. Lecture notes in computer science, vol 1117. Springer, Berlin/Heidelberg, pp 75–86. doi:10.1007/BFb0030098

Çatalyürek ÜV, Kaya K, Langguth J, Ucar B (2012) A divisive clustering technique for maximizing the modularity. In: Proceedings of graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop, 2012. Contemporary mathematics, vol 588. American Mathematical Society

Chakrabarti D, Zhan Y, Faloutsos C (2004) R-MAT: A recursive model for graph mining. In: Proceedings of the 4th SIAM international conference on Data Mining (SDM), Orlando. SIAM

Davis T (2016) The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/ research/sparse/matrices. Online. Accessed 17 Apr2016

van Dongen SM (2000) Graph clustering by flow simulation. PhD thesis, University of Utrecht

Dorogovtsev SN, Mendes JFF (2003) Evolution of networks: from biological nets to the internet and WWW. Oxford University Press, Oxford

Fagginger Auer BO, Bisseling RH (2012) Graph coarsening and clustering on the GPU. In: Proceedings of graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop, 2012. Contemporary mathematics, vol 588. American Mathematical Society

Fortunato S, Barthelemy M (2007) Resolution limit in community detection. Proc Natl Acad Sci 104:36–41

Gilbert H (1959) Random graphs. Ann Math Stat 30(4):1141–1144

Good BH, de Montjoye Y-A, Clauset A (2010) Performance of modularity maximization in practical contexts. Phys Rev E 81:046106

Holtgrewe M, Sanders P, Schulz C (2010) Engineering a Scalable High Quality Graph Partitioner. In: 24th IEEE international parallal and distributed processing symposium, 2010

Kannan R, Vempala S, Vetta A (2004) On clusterings: good, bad, spectral. J ACM 51(3):497–515

Karypis G, Kumar V (1999) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20:359–392

Kolda TG, Pinar A, Plantenga T, Seshadhri C (2013) A scalable generative graph model with community structure. arXiv preprint arXiv:1302.6636

Krioukov D, Papadopoulos F, Kitsak M, Vahdat A, Boguna M (2010) Hyperbolic geometry of complex networks. Phys Rev E 82(3):036106

Kunegis J (2013) KONECT: the koblenz network collection. In: Carr L, Laen-der AHF, Lóscio BF, King I, Fontoura M, Vrandecic D, Aroyo L, de Oliveira JPM, Lima F, Wilde E (eds) 22nd International World Wide Web conference, WWW '13, Rio de Janeiro 13–17 May 2013, Companion Volume, pp 1343–1350. International World Wide Web Conferences Steering Committee/ACM

Lambiotte R (2010) Multi-scale modularity in complex networks. In: 8th International symposium on modeling and optimization in mobile, ad-hoc and wireless networks (WiOpt 2010), 31 May–4 June. University of Avignon, Avignon, pp 546–553. IEEE

Lancichinetti A, Fortunato S (2009) Community detection algorithms: a comparative analysis. Phys Rev E 80(5)

Lancichinetti A, Fortunato S (2011) Limits of modularity maximization in community detection. Phys Rev E 84:066122

Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110

LaSalle D, Karypis G (2015) Multi-threaded modularity based graph clustering using the multilevel paradigm. J Parallel Distrib Comput 76:66–80

Leskovec J. Stanford Network Analysis Package (SNAP). http://snap.stanford.edu/index.html. Online. Accessed 17 Apr 2016

von Looz M, Meyerhenke H, Prutkin R (2015) Generating random hyperbolic graphs in subquadratic time. In: Elbassioni KM, Makino K (eds) Algorithms and computation – 26th international symposium, ISAAC 2015, Nagoya 9–11 Dec 2015, Proceedings. Lecture notes in computer science, vol 9472. Springer, pp 467–478

Meyer U, Penschuck M (2016) Generating massive scale-free networks under resource constraints. In: Goodrich MT, Mitzenmacher M (eds) Proceedings of the eighteenth workshop on algorithm engineering and experiments, ALENEX 2016, Arlington, pp 39–52. SIAM 2016

Newman M. Network data. http://www-personal.umich.edu/~mejn/netdata/. Online. Accessed 28 Sept 2012

Ovelgönne M, Geyer-Schulz A (2012) An ensemble learning strategy for graph clustering. In: Proceedings of graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop, 2012. Contemporary mathematics, vol 588. American Mathematical Society

Riedy EJ, Meyerhenke H, Ediger D, Bader DA (2012) Parallel community detection for massive graphs. In: Proceedings of graph partitioning and graph clustering, 10th DIMACS implementation challenge workshop, 2012. Contemporary mathematics, vol 588. American Mathematical Society

Sanders P, Schulz C (2016) Scalable generation of scale-free graphs. Inform Process Lett 116(7):489–491

Seshadhri C, Kolda TG, Pinar A (2012) Community structure and scale-free collections of Erdős-Rényi graphs. Phys Rev E 85(5)

Soper AJ, Walshaw C, Cross M (2004) A combined evolutionary search and multilevel optimisation approach to graph-partitioning. J Glob Optim 29(2):225–241

Staudt CL, Meyerhenke H (2016) Engineering parallel algorithms for community detection in massive networks. IEEE Trans Parallel Distrib Syst 27(1):171–184

Staudt C, Sazonovs A, Meyerhenke H (2015) Networkit: A tool suite for large-scale complex network analysis. CoRR, abs/1403.3005

Watts DJ, Strogatz SH (1998) Collective dynamics of "small-world" networks. Nature 393:440–442

Weicker R (2002) Benchmarking. In: Calzarossa M, Tucci S (eds) Performance evaluation of complex systems: techniques and tools, Lecture notes in computer science, vol 2459. Springer, Berlin/Heidelberg, pp 231–242