Comparative Analysis of Box-Covering Algorithms

Students' Scientific Conference, BME, 2020

by

Péter Tamás Kovács

Advisors:
Roland Molontay
Marcell Nagy

Contributor:
Botond Diviki-Nagy

BME, Institute of Mathematics
Department of Stochastics

Budapest University of Technology and Economics

2020

Contents

1 Introduction

Network science is a highly active research field aiming to describe and analyze networks. Relations between a group of entities can be naturally modeled as a graph where nodes and edges represent group members and their relations respectively, that makes network science a discipline of numerous potential applications in practical life.

The birth of the field is usually considered to be the formulation of the famous problem of The Seven Bridges of Königsberg by Euler in 1736. During the 20^{th} century, increased attention has been devoted to graph theory. From a theoretical point of view, notable contributions have been made on random graphs by – among others – Pál Erdős and Alfréd Rényi [1]. With the rapid increase in computing power, it has become possible to collect and analyze massive networks from different walks of life. Presently, it is widely believed that the formation of complex networks is not faithfully modeled by completely random processes. It is a challenging task to develop notions, which can capture important features of a real-world network.

Popular approaches often focus on the degree distribution of the nodes of a given network, on degree-degree correlations, on the clustering of the network, and various other measures. Networks are often labeled as belonging to loosely defined classes, e.g. networks can be 'scale-free' if their degree distribution follows power-law or 'small-world' if the average distance between two nodes grows with the logarithm of the number of nodes [2]. As it is apparent, the concise description of a complex network is far from trivial, so developing notions that enable one to reveal structural properties of such networks might be of great benefit.

One of the network groups is the class of *fractal networks*, introduced by Song, Havlin, and Makse [3]. The criterion of being fractal needs more explanation than the ones before, so it will be elaborated in the successive sections. To give a short idea, the notion of fractal networks is more or less the generalization of the notion of fractals from geometry. It is suggested that fractal networks have interesting properties that are mirrored in their robustness against external effects [6]. It is proposed in the literature that fractal networks are relatively robust against intentional attacks, which may provide an explanation why – for instance – numerous biological networks evolve towards fractal behavior [6].

In this paper, we review numerous algorithms that have been developed to analyze fractal networks. Particularly, to perform the so-called *box-covering* of the network, introduced later. We collected methods known from the literature, implemented them, and compared their performance on various real-world networks and theoretical network models. We think that the main contribution of this is the systematical comparison of many algorithms on numerous networks. In addition, we plan to release our code as an open-source Python package on GitHub, so our results may be reproduced and fellow researchers may leverage our collection of box-covering algorithms. *Although he is not part of the team anymore, we here acknowledge the work of Botond Diviki-Nagy in this project.*

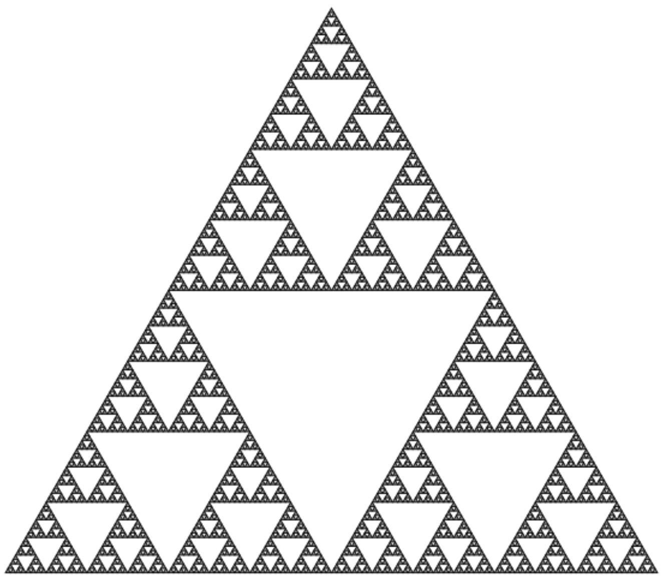

Figure 1: The Sierpinsky triangle [7] of fractal dimension ≈ 1.59 [8]

The rest of the paper is organized as follows: we first review the concept of fractalness in complex networks, then in section 2, we introduce the algorithms we tested. Section 3 is devoted to the networks on which the algorithms have been tested. Section 4 contains our methods and results with conclusions drawn in section 5.

1.1 Fractality in general

The term fractal comes from the geometry of Euclidean spaces. Objects we usually consider in a d dimensional space ($\mathbb{R}^d$) have a well-defined volume that could be evaluated in the following manner: we cover the whole space with a uniform grid of size l and count the total volume of hypercubes contained in our object. For example for a circle in two dimensions, we draw a grid consisting of squares of edge length l and count the number of squares inside the circle, then multiply with l^2. When one takes the $l \to 0$ limit, our results converge to the volume of the object.

If we consider the number of boxes that contain at least one point of the object, we usually find that asymptotically, this N number behaves as $N \propto l^{-d}$, where d is the dimension of the space. Now fractals are more 'complicated' in the sense that even though they live in $\mathbb{R}^d$, if we perform the above-detailed procedure, we find that $N \propto l^{-d_B}$, where d_B is the fractal dimension of the objects, generally not equal to d and not a natural number. A prominent example of fractals, the Sierpinsky triangle, is shown in Figure 1 .

1.2 Fractality of networks

The notion of fractal networks is quite reminiscent of ordinary fractals. To see the similarity, we recall the method by which the fractal dimension of networks can be measured: *box-covering* [3]

1. We say that a group of nodes fit into a box of size l_B, if for any pair of these nodes, their shortest distance is less than l_B.[1]

2. A network is covered by boxes of size l_B if its nodes are partitioned such that every group fits into one box of size l_B.

3. As the number of possible coverings is finite, there is a minimal number of boxes to cover the network with l_B sized boxes. This is what we denote by $N_B^{min}(l_B)$.

4. If $N_B^{min}(l_B) \propto l_B^{-d_B}$, the network is **fractal** with **fractal dimension** d_B.

Needless to say, this definition cannot strictly hold for finite networks. In practice, one calculates many $N_B(l_B)$ datapoints and then inspects them on a log-log plot to see if the dependence is power-law to a good approximation. What one can do is detailed in the next section, but we call attention to the fact that N_B^{min} can only be approximated in practice.

Obviously, fractality could be defined for weighted networks too, but in the following discussion, we only focus on *unweighted networks.*

[1]This is the usual convention, but for the algorithms, we will use a slightly different one. The reader shall behold this fact.

2 Algorithms

The box-covering of a network is known to be NP-hard, since it can be mapped to the famous vertex coloring problem that is indeed NP-hard [4]. Hence to have an algorithm of practically acceptable complexity, one must use approximating methods. As it is usual for semi-empirical methods, there are a good number of different proposals for the task.

To give some intuitive summary, we can say that most algorithms follow a greedy strategy where the distinction between algorithms is drawn by the actual greedy decision method. However, we present a number of approaches beyond the greedy paradigm.

It is also interesting to note that many algorithms use the notion of 'centres', which means that each box is assigned to a special, 'central' node. While this idea is natural (and indeed true) in $\mathbb{R}^n$, it may be misleading for a graph since the edges between vertices generally cannot be embedded into a Euclidean space. Just imagine a box containing a cyclic subgraph - no vertex is special by any means.

In the following part, we describe implemented algorithms in detail and our comments on them. Throughout the discussion, we denote the box size by l_B, the box radius (if applies) by r_B.

Before starting the discussion, we must precisely define what we mean by box sizes. Unfortunately, it sometimes happens in renowned papers too that their treatment of this question is inconsistent [4].

In our **implementation**, we define boxes such that for box-size l_B the distance between two nodes of the same box can be *less than or equal to* to l_B. For boxes defined by r_B radius, all nodes have a distance *less than or equal to* from the central node.

In the **evaluation** part, we converted these values to the widely used convention: that one should have *strictly less* for l_B. This means that the 'equivalent box-size' is $\tilde{l_B} = l_B + 1$ and $\tilde{l_B} = 2r_B + 1$, where the rhs. contains values introduced in the above paragraph.

In this section, we use the *less than or equal to* ('implementation') convention for the algorithms.

2.1 Greedy coloring

This family of algorithms maps the boxing problem to the famous graph coloring problem. The idea is the following: let us consider the *dual graph* of our original network: this graph consists of the same vertices and 'dual edges' [4]. This means that two vertices are connected by an edge in the dual graph if and only if their distance in the original network is greater than l_B. The idea is illustrated in Figure 2 with $l_B = 2$.

After these preliminaries, we aim to color the vertices of the dual graph such that **i)** neighbouring vertices cannot have the same colour **ii)** we seek to use the

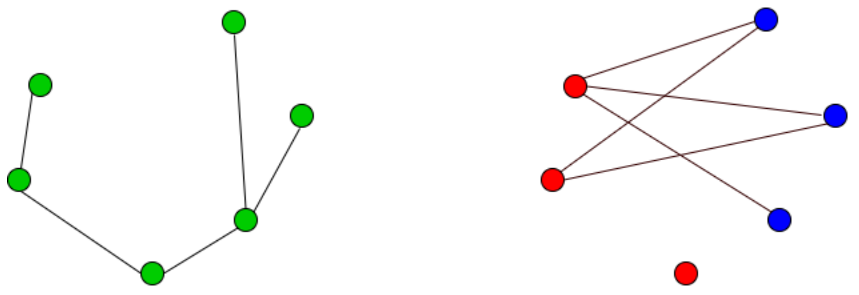

Figure 2: A graph (left) and its dual (right). We also show a valid coloring of the dual graph. (We used l_B=2.)

least colours. This problem is equivalent to the box-covering of the original network when we identify vertices of the same 'dual' colour as belonging into one box in the original network.

Unfortunately, graph colouring is NP-hard. A well known approximating algorithm is *greedy coloring.* This algorithm consists of two main steps:

1. Order the nodes by some method
2. Iterate over the above sequence: assign the smallest possible colour ID to every node

The algorithm is given completely if the ordering method is specified. In our sample results, we used a random sequence. Even though more advanced methods exist, we think that this *naive* approach is a sensible baseline for other algorithms.
It must be noted that the implementation relies on the implementation of greedy colouring in the `networkx` package [9], so one could have easily selected any other built-in strategy.

2.2 Random sequential

This is our first algorithm that uses the idea of burning, meaning that once some nodes have fit into one box, they are allocated to that box, are 'burned out'.

The original idea for the random sequential algorithm came from [12]. In every step, an unburned node is randomly chosen[2] and we sort unburned nodes to this center that are not farther than r_B. These nodes together form a box.

2.3 Compact Box Burning (CBB)

This algorithm also uses the concept of burning nodes. The original idea was presented in [4]. The main point is to grow boxes such that we pick new nodes randomly from a set that contains all nodes that are not farther than l_B from any node already in the box. This guarantees that the box is compact[3] in the sense of the authors.

[2]Note that in the original paper, all the nodes participated in the random selection! The authors argue that in some cases, that was necessary to obtain the desired behavior.

[3]Roughly speaking we cannot add any more nodes.

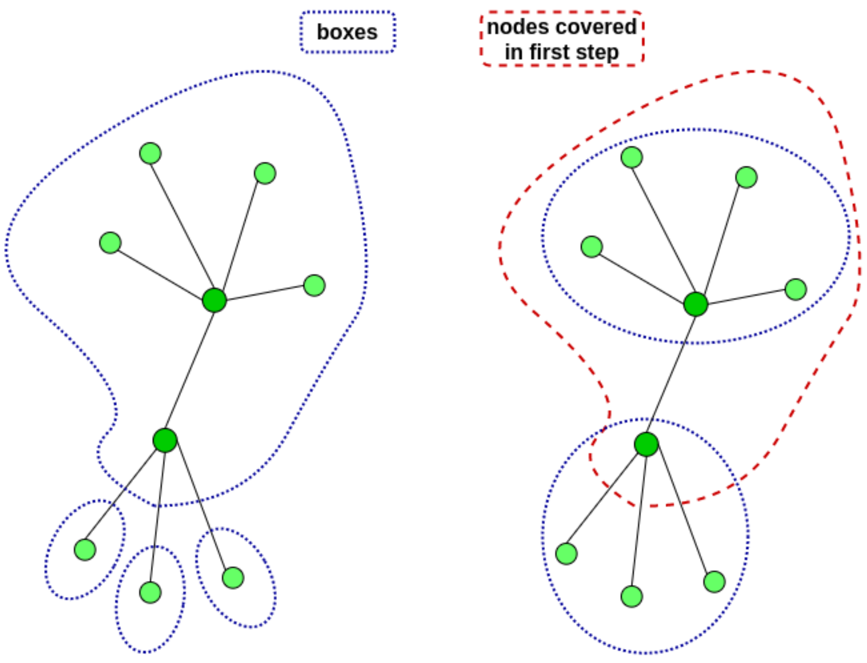

Figure 3: Illustration for MEMB. The left graph shows the outcome when covered nodes would not be allowed as centers, the right illustrates when they are. (Using r_B=1)

2.4 Maximal Excluded Mass Burning (MEMB)

This algorithm was also presented in [4]. Instead of using l_B, this method uses the notion of centered boxes: every box has a special node, a center. Boxes are constructed such that every member node of the box is not farther than r_B from the center. The algorithm guarantees that all the boxes are connected, two nodes of the same box can always be connected with a path inside the box.

The algorithm may be considered as an improvement of the random sequential method in two aspects but the analogy fails because the way burning is implemented is different from random sequential.

- First, the next center is chosen based on the *maximal excluded mass*, that is the number of uncovered nodes not farther than r_B from the center[4].
- Once the next center is chosen, all uncovered nodes within r_B are covered, but not yet assigned to boxes.
- Finally, after every node is covered, non-center nodes are assigned to centers in a way that the resulting boxes are connected.

The reason why boxes are only formed in the end, not on the fly burnt can be understood from Figure 3. The authors' goal was to avoid 'hub-failures' by selecting

[4]Including the center too if it is uncovered. It is also possible to have a covered node as the next center.

the first centers and then assigning the remaining nodes to them.

2.5 Ratio of Excluded Mass to Closeness Centrality (REMCC)

This algorithm can be thought as an altered version of MEMB [15]. In every step, a new center is chosen from uncovered nodes such that the choice maximizes the 'f point', a novel metric in the paper, which is the excluded mass times the average shortest path to all other nodes. All nodes in the r_B ball of the center are covered then. In this procedure, centers are selected from *uncovered*, as opposed to MEMB.

(In oue current implementation, no boxing scheme is introduced, only the centers are determined.)

2.6 MCWR

This algorithm comes from [16]. In fact, it is a combination of MEMB and the random sequential (RS) algorithm. In addition, a new way of keeping track of excluded mass is proposed.

The core concept is that we modify MEMB such that before choosing a new center, we toss a biased coin so that we perform the usual MEMB steps with probability p and choose a random center in the remaining cases[5]. After finding a new center, the excluded masses are updated: only nodes inside the newly covered vertices' r_B ball are affected. This is the 'novel scheme' for excluded mass. The boxing of non-center nodes is organized as in MEMB.

It is obvious that p is a hyperparameter that needs tuning. Intuitively, one could argue that smaller p would imply the algorithm to 'be closer to random sequential' hence be faster and less accurate.

2.7 Merge algorithm

The algorithm was proposed in [17]. This algorithm is in some sense reminiscent of mathematical induction because for a given box size, we try to merge boxes inherited from a smaller l_B box size. The procedure starts from box-size 0 with all nodes forming a box of one node.

Let C be the list of input clusters. In one step, list D is returned, which is the merged version of C, computed as follows:

- First, a random member of C, c_k is chosen
- We then check all elements of C for being able to be merged with c_k (the union of the clusters would still respect the box-size condition)
- We draw a random cluster from the ones found in the previous step. It is merged with c_k

[5]Here, the choice is made such that covered nodes but the ones with $m_{ex} = 0$ are allowed

- The new cluster is stored in D while the parents are removed from C. Cycle goes on until C is empty.[6]

Then, the box size is incremented and the whole procedure is repeated. The number of boxes for given l_B, $N_B(l_B)$, is the cardinality of D after merging[7].

2.8 Overlapping Box Covering Algorithm (OBCA)

This method was originally proposed in [21]. The authors suggest that instead of burning boxes on the fly, one should only mark possible boxes while processing data and then choose the final boxing in the end.

The algorithm proceeds by first only creating box proposals, during which possible center nodes are iterated over in an ascending order wrt. degree.

- A node is a possible center if it is uncovered.
- In a 'proposed box', such nodes are included whose distance from one another is at most l_B. They are chosen from nodes whose distance from the center is at most l_B
- Increase the 'covering frequency' of all nodes contained in the newly proposed box. This means that *proposed boxes* can overlap, hence the name Overlapping Box Covering Algorithm.

After this, we revisit the proposed boxes. A box is called 'redundant' if only contains nodes that are at least in another proposed box (so their covering frequency is higher than 1). Redundant boxes are deleted and the covering frequency of the just-deleted box's nodes is decreased by 1. After the iteration is over, only non-redundant boxes survive.

[6]If c_k cannot be merged with any other cluster from C, then it is added to D and removed from C.

[7]So then the initial condition, corresponding to box-size 0 has N_B equaling to the number of nodes.

3 Investigated networks

We have tested the implemented algorithms on numerous real-world networks and theoretical network models. This section introduces the employed networks.

Additionally, the size and degree distribution of the networks is compared. This will characterize the networks to some extent.

3.1 Theoretical models

3.1.1 The UV-flower

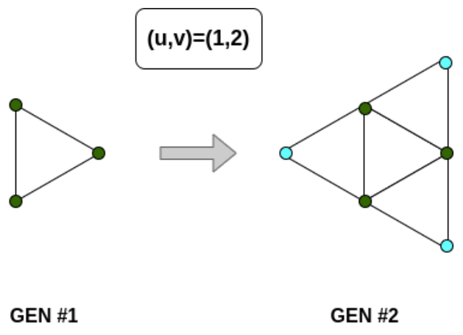

Figure 4: The first two generations of UV21

The UV flower is indeed a well known model yielding fractal networks [10].

The structure of the network is best understood considering its creation: we start with a circular graph of u+v nodes as the first (g=1) generation. In every generation, we replace all existing edges, with a u and v long path each[8]. This process is repeated until generation n (g=n).

We cite some essential properties of UV flowers[10]. Their degree distribution is power-law: $P(k) \propto k^{-\gamma}$ where $P(k)$ is the relative frequency of nodes with degree k and for the exponent[9]:

$$\gamma = 1 + \frac{ln(u+v)}{ln\ 2} \tag{1}$$

Moreover, the fractal-dimension of the network, for $1 < u \leq v$:

$$d_B = \frac{ln(u+v)}{ln(u)} \tag{2}$$

For $u = 1$, the network is not fractal but small-world.

We have considered networks with (u,v,n)=(2,2,5) and (2,4,4). These networks are of 684 nodes, 1024 edges, and 1038 nodes, 1296 edges respectively.

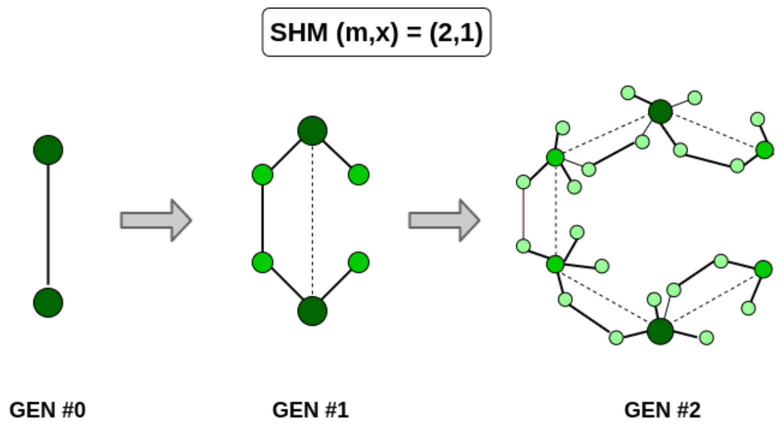

Figure 5: The first three generations of the SHM model with m=2, x=1

3.1.2 The SHM model

Introduced by Song, Havlin, and Makse, the growth of this network is controlled by a continuous p parameter [10]. The idea is to reverse the renormalization process in some sense. In every generation, $k \cdot m$ new nodes are linked to each previously existing node, with k being the respective degree. Between any two linked 'parent' nodes, the edge is removed with *1-p* probability, and x new edges are formed between their 'children' nodes. The process is repeated until the desired generation number is reached. Generation #0 starts with two connected nodes. The evolution starts from here, and g iterations are performed. We identify an SHM network with (g,m,x,p) parameters.

Again, we cite relevant properties [10]. The network is pure fractal for $p = 0$. For this type of network, the degree distribution $P(k) \propto k^{-\gamma}$, with

$$\gamma = 1 + \frac{ln(2m+x)}{ln(m)} \tag{3}$$

and the fractal-dimension:

$$d_B = \frac{ln(2m+x)}{ln(m)} \tag{4}$$

During the investigations, we considered (g,m,x,p)=(4,2,2,0) and (5,2,1,0) networks. These networks have 1038 nodes, 1296 edges and 3126 nodes, 3125 edges respectively.

3.1.3 Grid graphs

Grid graphs are another class of fractal networks. We investigated 'two-dimensional' grid graphs where the structure can be thought of as an $n \times n$ grid, with edges between neighbouring points.

The fractal dimension of such grid networks is obviously $d_B = 2$.

We investigated 'two-dimensional' networks of n=30 and 50. (With 900 nodes, 1740 edges and 2500 nodes, 4900 edges respectively.)

[8]So then (u+v-2) new nodes are added per edge.

[9]This dependence is usual with scale-free networks.

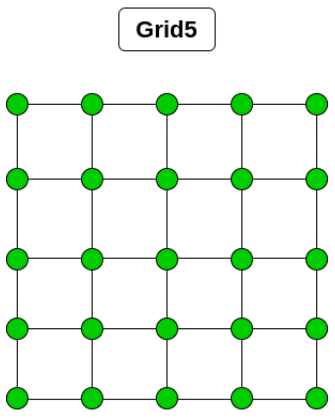

Figure 6: An 5×5 grid graph

3.2 Real-world networks

3.2.1 The EColi network

This network contains the cellular metabolic data of the Escherichia Coli bacterium [23]. We used the maximal connected component for the box-covering purposes that had 2859 nodes and 6890 edges between them.

3.2.2 The Enzyme network

This network is a relatively small one, with a maximal connected component of 125 nodes and 141 links between them, acquired from [24] [25]. We note that we considered the enzyme links as follows: if two enzymes were in any type of directed connection, we considered them to be connected with an undirected edge.

3.2.3 Minnesota network

This network is an undirected, unweighted network representing Minnesota's road network[10] [24] [25]. The maximal connected component of the network consists of 2640 nodes and 3302 edges.

Degree distribution of real networks

To have some idea about the degree distribution of real-world networks, we prepared histograms depicting it. We can see that *Minnesota* and *Enzyme* do not have high degree nodes. (Eg. Minnesota might be reminiscent of a grid graph or similar.) On the contrary, *EColi* has some hubs of a very high degree.

[10] I am not convinced that this is the way one can model the road network but this usage is promoted by the data repository of origin.

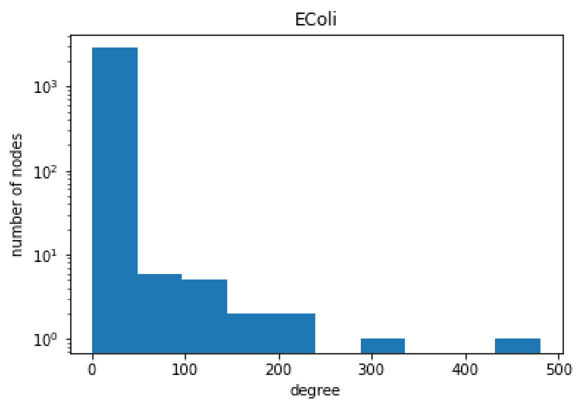

Figure 7: The degree distribution of the EColi network. Behold the log scale.

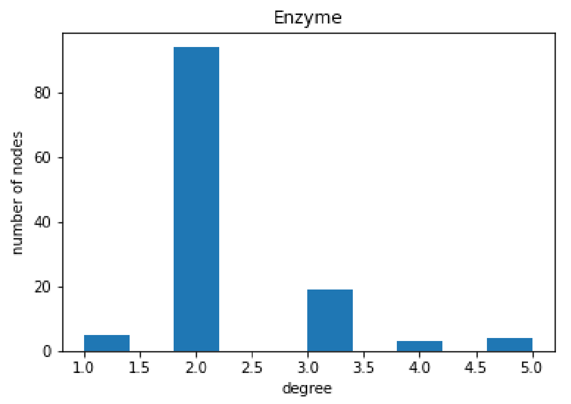

Figure 8: The degree distribution of the Enzyme network

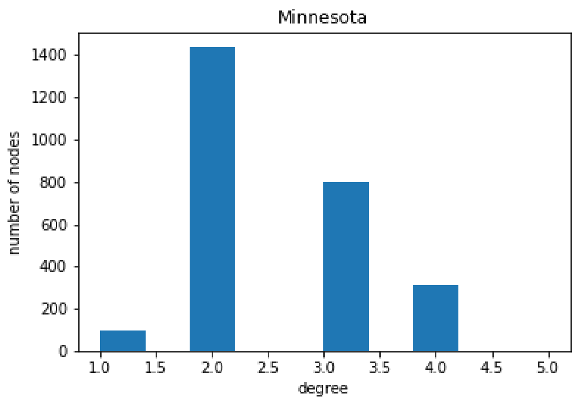

Figure 9: The degree distribution of Minnesota network

4 Methods and results

In this section, we present a way to assess the performance of the implemented box-covering algorithms. We stress at this point, that there may be no good answer to the question: 'Which algorithm is the best?' In general, we can only say that it depends on the network and the desired precision.

To elaborate on a practically more useful answer, we have chosen a particular method to quantify the performances of algorithms that seem to be reasonable for us. In this contribution, we decided to focus on the returned box-number of the algorithms. It is a natural choice since the goal of any algorithm is to find the minimal number of boxes to cover the network.[11]

To do so, we first defined a *goodness-score* that describes how good a given algorithm on a particular network with fixed box-size performs. After this, we *aggregate* these scores such that box-sizes with too few boxes are left out. Finally, the performance is judged by these derived quantities over all considered networks. This last step is somewhat subjective but we felt that it may be more useful to any reader to establish some 'bigger-picture' than to present a massive block of data with some heuristic metric.

4.1 Data for a given algorithm

In this part, we describe what type of data is assembled for a given algorithm and network.

4.1.1 Box-numbers

While performing measurements with the implemented algorithms, we aim to get a picture of how the algorithm performs with a given box size. Because most of the algorithms are non-deterministic, we need to run the algorithm multiple times and then proceed with the empirical distribution of these values.

In our study, we ran every algorithm with a given box-size 15 times.

4.1.2 Box-sizes

The previous procedure is repeated for every box-size. Box sizes are chosen such that when it comes to comparison of results of different algorithms, we have equivalent sizes. Behind this obscure statement stays the fact that some algorithms are parametrized by a *box-radius*, others by a *box-size*. These values need to be brought into alignment.

[11]Another sensible option would be the fractal dimension determined by box-numbers for a multiple number of box-sizes. In addition to this method being ill-defined because of outliers and possibly imperfect fractality, we could not proceed with this approach because of what follows. To our surprise, the measured fractal dimensions of theoretical graph models are quite far from the 'ground truth' value. We conjecture that this may be the effect of the finite size of investigated networks. Needless to say, this problem needs further considerations.

The particular values of box-sizes were chosen such that we had measurements for box-sizes distributed from small values to approximately the diameter of the network.[12]

4.1.3 Measurement of execution times

For a whole iteration with one algorithm and all box sizes, the execution time is measured. It must be noted that because of the differences in the algorithms, they do not necessarily need the same (equivalent) box sizes. It is simply because the methods differ in the efficiency of covering the network. To account for this, we have implemented a stopping criterion as follows: if there is a point while the iteration that the number of boxes is 1 for all 15 times, then no bigger box-sizes are investigated and the measurements with the algorithm finish. (Due to the details in its nature, this cannot be done in the *merge algorithm* but this turns out to have no real consequences then *merge* is the fastest anyway.)

We note that the time of 'preparatory' operations i.e. computing shortest path data for the graph is only measured once and added to the execution time of 15 iterations of the given algorithm.

At this point, we note that the measurement of execution times was not performed under strictly regularized conditions. Such an arrangement would have been achieved when carefully controlling all miscellaneous processes on the host device, that was in fact my notebook[13]. Henceforth, the values of execution times shall be considered only as *tentative values* from whose one can determine if an algorithm is 'relatively slow', 'moderate', or 'relatively fast'.[14]

4.2 Aggregation methods

To gain insights, we need to process the raw data that we acquired in the manner described in the previous section.

4.2.1 Benchmark

To start the investigations, we defined a benchmark value for every (network, box-size) pair. This value is an estimation that 'a reasonable algorithm' should perform like this. We have chosen it to be the minimal box-number that has been achieved with the *greedy* algorithm, which is a standard approximating algorithm for the graph coloring problem.

4.2.2 Goodness score

Next, we defined the goodness score of a given box-number: $g_{ij} = (b_{ij} - B_i)/B_i$ where b_{ij} is the box number for the i^{th} box-size computed in the $j = 1, 2, ..., 15$ iteration, and B_i is the respective benchmark value. (Of course, it also depends on

[12]Usually with logarithmical spacing.

[13]Processor: Intel Core i7-6700Q, RAM allocated to boxing: 4-5 GB

[14]This seems to be a flaw at this point but later, since we want to establish some general remarks on the algorithms across all networks, merely this is what we need.

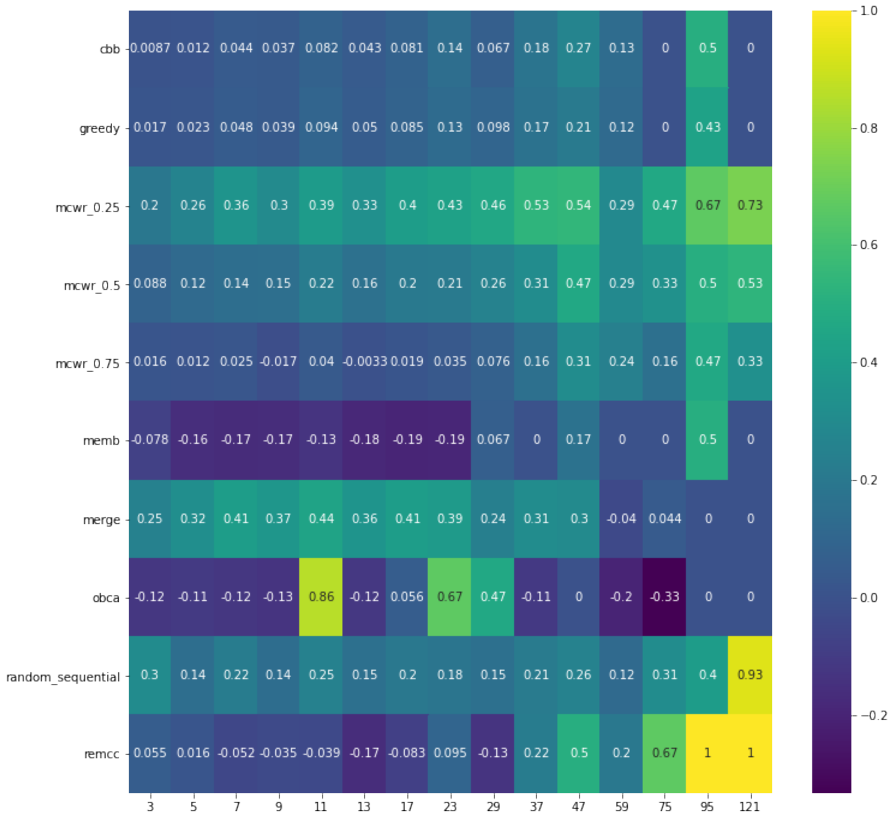

Figure 10: Illustration of outliers with g_{ij} goodness scores averaged in index j. The OBCA algorithm has one at box-size 11 clearly. (Data for the *grid50* graph.)

the network.) Note that this measure is indeed 'dimensionless' since it gives the relative difference from the benchmark. Thus, it makes sense to aggregate these scores.

4.2.3 Aggregation, cutoff in box-size

We employed two different kinds of aggregation.

First, we only averaged the g_{ij} scores for a given box-size. (So in index j.) This was primarily done to inspect the course of these values evolving with the box-size. This type of analysis is capable to point to outliers as follows: it may be possible that due to the interplay between the network structure and the algorithm, an otherwise decent algorithm produces a very high box-number at a given box-size. This is illustrated in Figure 10. We will not directly use this data but it is good to have it at hand to perform a deeper analysis.

Cutoff

Secondly, we realize that computing goodness scores at low box numbers may not be very useful because, at low box-numbers, only one box difference causes a substantial difference in the goodness score. Moreover, the fractality of complex networks tends to break down towards small box-numbers (big box-sizes) so these points might be discarded when computing the fractal dimension anyway.

Motivated by this, we will only consider such goodness scores in the further investigation, that correspond to a benchmark box-number greater or equal than a given *cutoff-size*. This cutoff was chosen to be 10 in most cases but for small networks, it had to be chosen smaller to consider a meaningful number of boxes. Table 1 shows the cutoff values and the number of box-sizes greater or equal than it per network.

	ecoli	enzyme	shm4220	shm5210	uv225	uv244	minnesota	grid30	grid50
cutoff	10	5	10	10	5	10	10	5	10
box no.	4	4	9	10	4	4	10	6	9

Table 1: Cutoff values of the analyzed networks. The box number refers to the number of box sizes that are greater or equal than the cutoff value.

Aggregation

With these preliminaries, it is very easy to understand the final aggregation: for a given (network, algorithm) pair, we consider all g_{ij} values that correspond to a box-size allowed by the cutoff. Their mean and standard deviation is computed.

This means that we can describe the performance of an algorithm on a given network with the (mean goodness, standard deviation of goodness, execution time) triad. These values can be visualized in multiple ways. Here we present an approach we think to be fruitful: every algorithm is represented by a patch in the (mean goodness, execution time) plane and the area of the patch is proportional to the variance (square of the standard deviation). Such a graph is presented as an illustration in figure 11. We found this representation helpful in comparing the performance of algorithms.

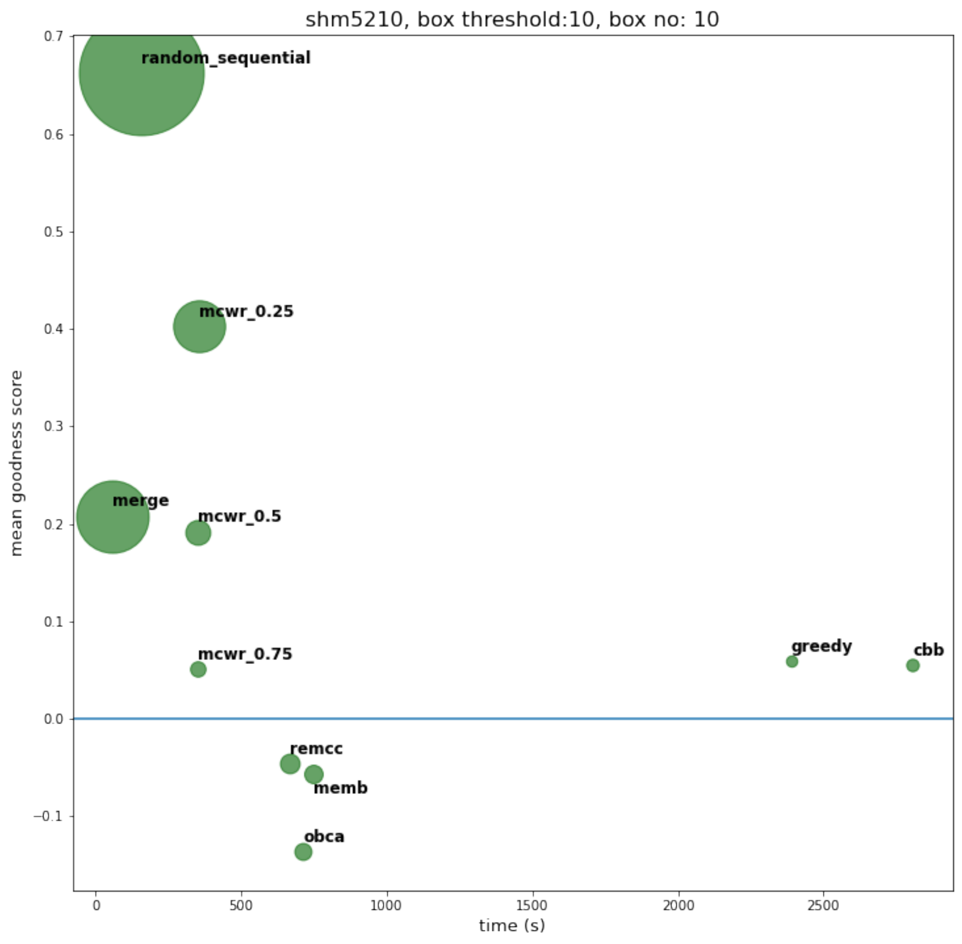

Figure 11: Sample figure illustrating our novel scheme for gaining insights

4.3 Results

After generating the above plots, we may turn to the final – more subjective – part of the analysis: using the measurements to assess overall performance. Here we mainly worked by the inspection of the above-generated graphs. In each plot, we analyzed if a given algorithm is relatively accurate, low variance, and fast.

To support our conclusions, we present the mean (aggregated) goodness scores, their standard deviation, and the respective execution times as heatmaps and in tabular format.

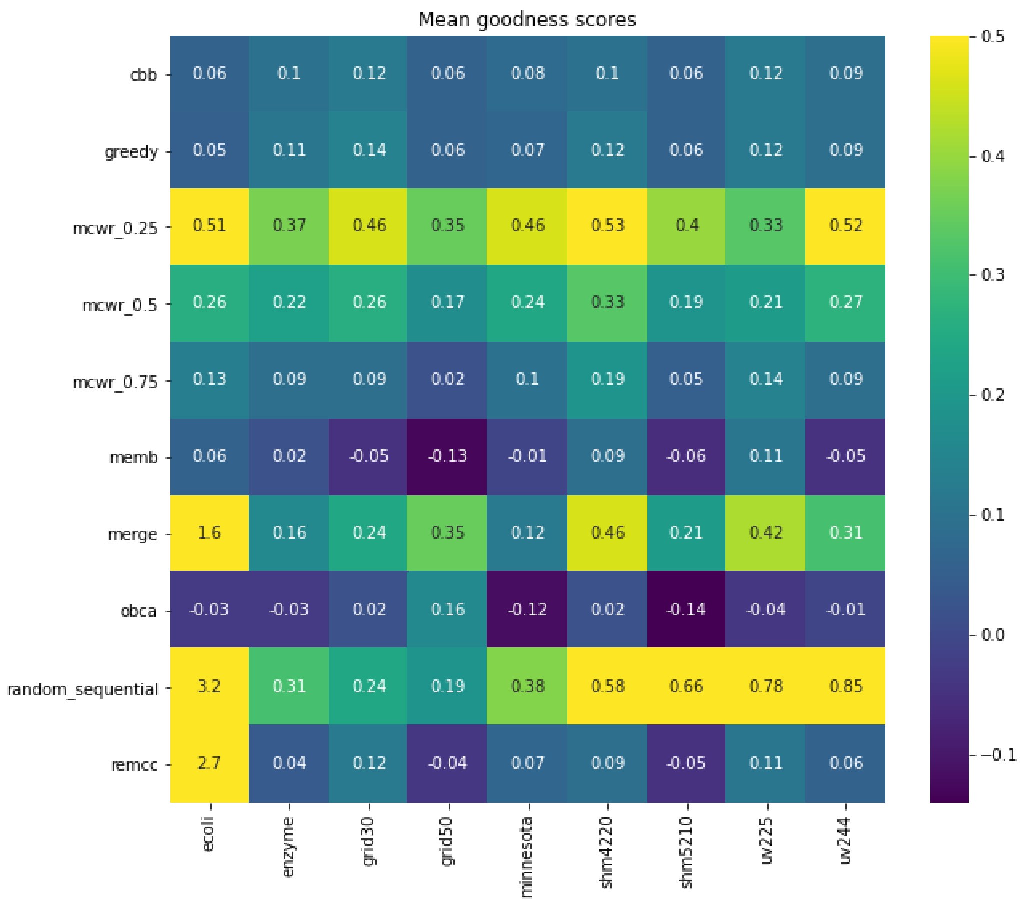

Figure 12: The mean goodness scores for each (network, algorithm pair)

	ecoli	enzyme	grid30	grid50	minnesota	shm4220	shm5210	uv225	uv244
cbb	0.06	0.10	0.12	0.06	0.08	0.10	0.06	0.12	0.09
greedy	0.05	0.11	0.14	0.06	0.07	0.12	0.06	0.12	0.09
mcwr_0.25	0.51	0.37	0.46	0.35	0.46	0.53	0.40	0.33	0.52
mcwr_0.5	0.26	0.22	0.26	0.17	0.24	0.33	0.19	0.21	0.27
mcwr_0.75	0.13	0.09	0.09	0.02	0.10	0.19	0.05	0.14	0.09
memb	0.06	0.02	-0.05	-0.13	-0.01	0.09	-0.06	0.11	-0.05
merge	1.63	0.16	0.24	0.35	0.12	0.46	0.21	0.42	0.31
obca	-0.03	-0.03	0.02	0.16	-0.12	0.02	-0.14	-0.04	-0.01
random sequential	3.18	0.31	0.24	0.19	0.38	0.58	0.66	0.78	0.85
remcc	2.74	0.04	0.12	-0.04	0.07	0.09	-0.05	0.11	0.06

Table 2: Mean goodness scores for all (network, algorithm) pairs

From the mean goodness scores, *MEMB* and *OBCA* seem to be stand out from the others. Algorithms *merge*, *random sequential* and *MCWR_0.25* seem to be bad, random sequential particularly.

It is interesting to observe that on networks *Enzyme, Grid30, Grid50* and *Minnesota*, the 'fast-but-inaccurate' algorithms (*merge, random sequential*) perform generally better than on other networks. This could be easily thought of as a consequence of these graphs having only low-degree nodes. Explaining in a speculative manner, there might be no 'special' nodes (eg. hubs), so *merge* and especially *random sequential* improve their performance because randomizing box formation does not imply too much drop in accuracy. (And indeed, it really speeds up box-covering.) This trend is not observed for other algorithms.

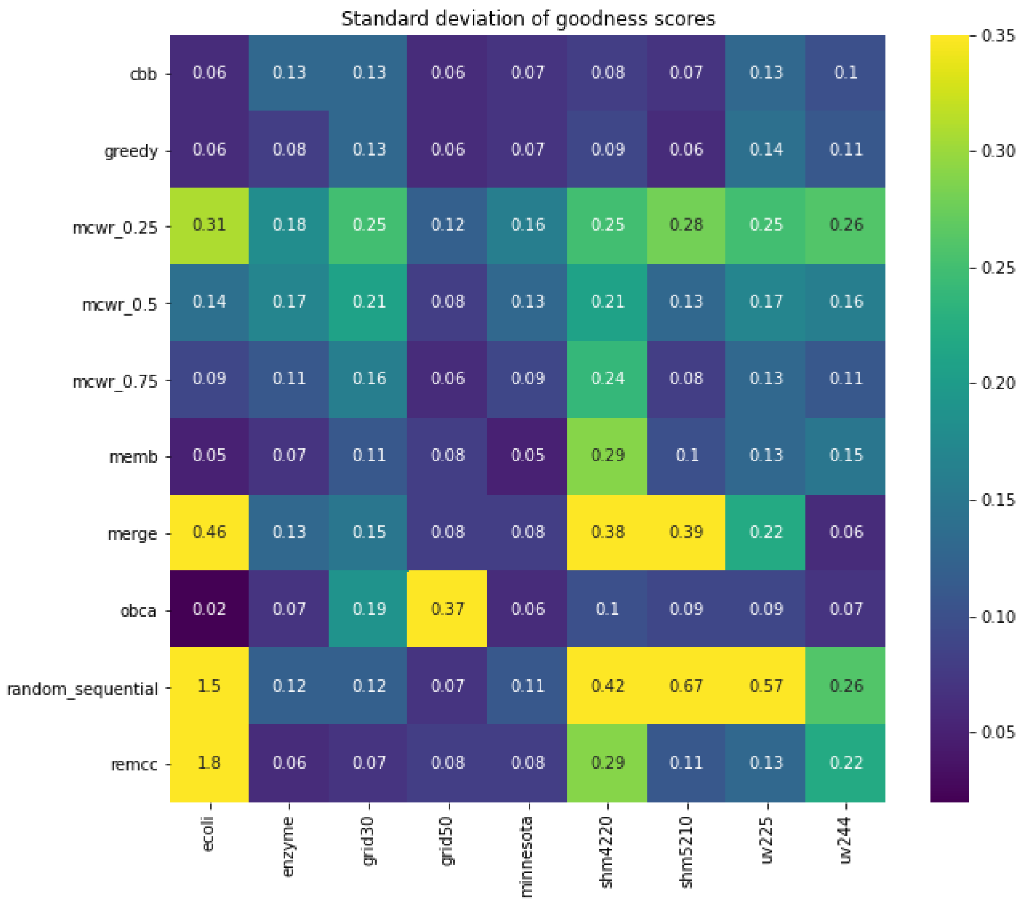

Figure 13: The standard deviation of the goodness scores for each (network, algorithm) pair

	ecoli	**enzyme**	**grid30**	**grid50**	**minnesota**	**shm4220**	**shm5210**	**uv225**	**uv244**
cbb	0.06	0.13	0.13	0.06	0.07	0.08	0.07	0.13	0.10
greedy	0.06	0.08	0.13	0.06	0.07	0.09	0.06	0.14	0.11
mcwr_0.25	0.31	0.18	0.25	0.12	0.16	0.25	0.28	0.25	0.26
mcwr_0.5	0.14	0.17	0.21	0.08	0.13	0.21	0.13	0.17	0.16
mcwr_0.75	0.09	0.11	0.16	0.06	0.09	0.24	0.08	0.13	0.11
memb	0.05	0.07	0.11	0.08	0.05	0.29	0.10	0.13	0.15
merge	0.46	0.13	0.15	0.08	0.08	0.38	0.39	0.22	0.06
obca	0.02	0.07	0.19	0.37	0.06	0.10	0.09	0.09	0.07
random sequential	1.48	0.12	0.12	0.07	0.11	0.42	0.67	0.57	0.26
remcc	1.78	0.06	0.07	0.08	0.08	0.29	0.11	0.13	0.22

Table 3: Standard deviation of goodness scores for all (network, algorithm) pairs

Standard deviations also point to the inaccuracy of *merge* and *random sequential* and *MCWR_0.25*, *MCWR_0.5*. It is not so apparent from the figure but more detailed analysis reveals (with figures like fig. 10) that the performance of the *OBCA* algorithm is quite versatile for *grid* networks. (Depending on box size, the mean goodness scores vary much - see the outlier for *grid50* on fig. 10 for example.)

Without completeness, such 'outliery' behaviour was observed for most algorithms on network *SHM4220*. Again, we observe that *merge* and *random sequential* perform much better in terms of variance on the 'no-hub' networks mentioned before.

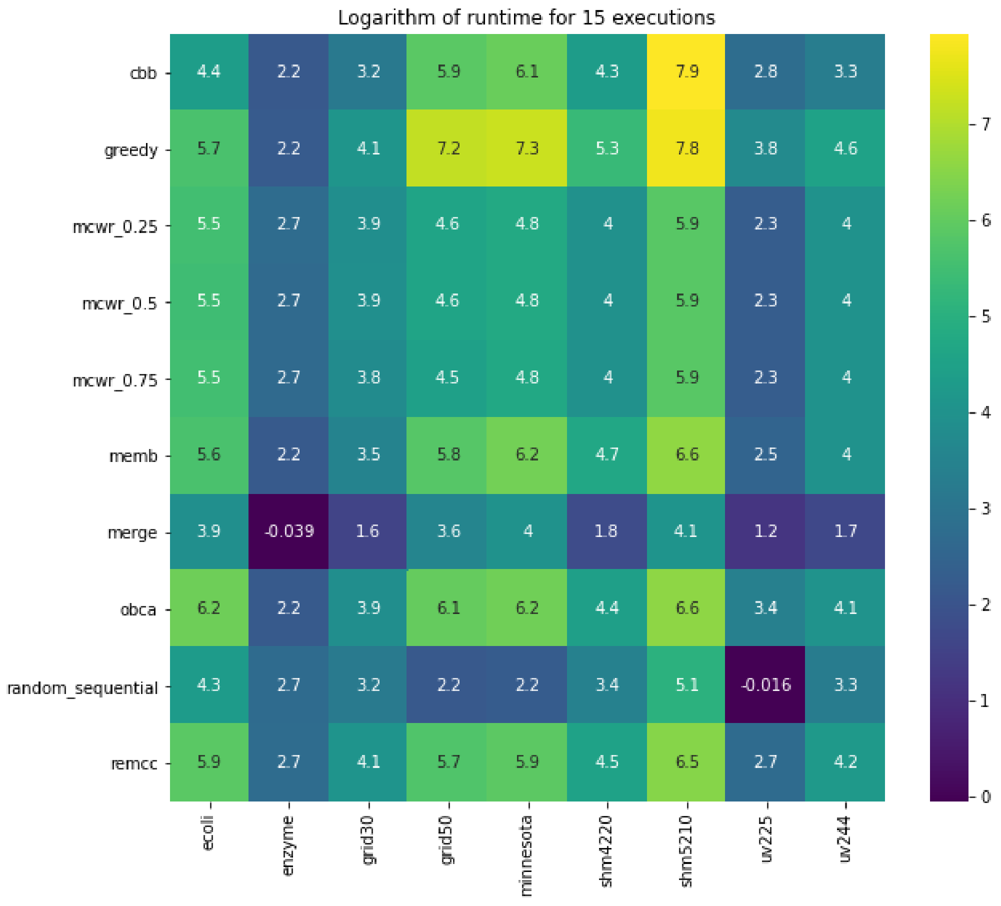

Figure 14: The *natural logarithm* of execution times in seconds, for each (network, algorithm) pair

	ecoli	**enzyme**	**grid30**	**grid50**	**minnesota**	**shm4220**	**shm5210**	**uv225**	**uv244**
cbb	78.5	8.7	24.4	369.6	459.0	76.4	2805.5	16.3	27.4
greedy	287.3	9.2	61.6	1340.5	1479.5	206.0	2388.4	42.9	96.3
mcwr_0.25	250.8	15.4	48.2	100.9	117.7	53.1	355.3	9.7	54.2
mcwr_0.5	232.9	14.5	49.0	101.3	118.2	53.0	351.5	9.7	54.2
mcwr_0.75	250.7	14.9	43.9	85.8	116.1	52.5	351.3	9.7	56.0
memb	283.8	8.8	33.7	338.8	514.6	111.9	748.8	12.3	54.7
merge	49.3	1.0	4.7	36.6	55.7	6.0	60.0	3.3	5.5
obca	478.9	8.8	48.2	429.1	496.3	85.4	713.4	30.2	57.4
random sequential	73.8	14.9	25.2	8.7	9.5	31.1	157.6	1.0	26.9
remcc	365.6	15.4	59.6	314.0	375.4	94.5	664.8	15.2	69.7

Table 4: Execution times in seconds for all (network, algorithm) pairs

As we mentioned before, execution times shall only be thought of as tentative values, nonetheless, we can make out clear trends. As a rule of thumb, precision and speed are in negative correlation.

The standard method, *greedy*, is very slow compared to others. On the other end of the spectrum, *merge* is probably the fastest even though it calculates box numbers for all box-sizes up to the maximal. Surprisingly, *random sequential* shows volatile performance over our networks as *CBB* does too. Algorithms *MEMB* and *OBCA* show moderately high times.

The case of *MCWR* deserves particular attention. It was claimed that this method essentially speeds up *MEMB* because, in every step, we randomly choose between a random- and a maximal excluded mass driven center nomination method. Surprisingly enough, despite execution times usually being smaller than for *MEMB* (extent depending on the network), we have almost the same execution times for all p values. This implies that the algorithm shall be revisited. I suppose that at the center nomination in the random branch, one shall exclude covered, non-center nodes. (They may spoil speed at the end when almost every node is covered.) The administration of excluded masses may be improved too.

5 Conclusion, closing remarks

In this summarizing section, we shall distill some general conclusion from all presented data. Needless to say, it is less exact and prone to reflect the particular ideas we have about the presented methods. Hence the reader is encouraged to form his/her own opinion based on our results.

5.1 Conclusion based on our investigations

- We primarily assess performance by the mean goodness score since it reflects how well an algorithm performs. In this aspect, **MEMB** and **OBCA** might be the best. They both operate with moderately large execution times and usually tolerable variation. We note that the former might outperform the latter in speed (at least for our networks). Nonetheless, both methods shall be kept in mind since they operate on fairly different principles, so in a new scenario trying both may be of help.

- When pulling execution times into consideration, one should also consider **MCWR** which is in fact a crossover between *MEMB* and *random sequential* with the promise of elevated speed. I feel that this direction should be considered in a deeper analysis in the future. Particularly, one shall make up for having execution times almost insensible to the p value. This might be achieved by optimizing the random choice. If this could be done, then *MCWR* would be a decent method to choose when *MEMB* and *OBCA* are too slow. In addition to this, it would be interesting to modify the algorithm such that the probability of choosing between branches is not constant, but employing lower p values when only low-excluded mass nodes are left. (Thus it may not matter much which to choose as the next center.)

- Analysing results, **merge** algorithm stands out in terms of speed. Goodness scores are far from optimal but mostly acceptable and merge is an acceptable method if our network is so big that other methods require intolerably much time. It is stressed that in the current version of the algorithm, there is a *caveat* that must be addressed to have more reliable results. It is that when two clusters are merged, they are 'thrown away' in the sense that for their union (the new cluster) it is not considered if it could be merged with another cluster. Consequences are especially dire when one has a network of relatively small diameter. (Besides this, finding a mergeable pair could be done much more efficiently - this is not an algorithm but rather a code issue.)

- The remaining methods, **CBB**, **greedy**, **REMCC** and **random sequential** cannot be recommended *based on our results.* Despite being an established algorithm, *greedy with random colouring sequence* is too slow to be practical. We do not 'denounce' the greedy paradigm in general, since *our results only applied to the random sequence strategy.* But I am not sure if selecting a different strategy would considerably speed up matters. For *CBB* and *REMCC*, they both seem to be inferior to *MEMB.* Finally, *random sequential* showed low accuracy and volatile speed so it is clearly inferior to *merge.*

- The choice of the employed algorithm may also depend on the network structure besides network size. We found on a limited number of networks that fast and straightforward methods (*merge* and *random sequential*) tend to perform considerably better on no-hub networks with nodes only of small degree. Such a clear trend cannot be made out for other algorithms.

5.2 Possible improvements, further directions

We finish our contribution by listing a few possible research directions for the future:

- The implemented algorithms should be compared on more networks. This should be only a matter of time and computational resources. During this effort:

- The resource-intensiveness of algorithms will have to be reduced. Recently, we have run into the issue that our current implementation uses too much memory to run on our local device with approximately 4 GB RAM allocated to running box covering. I believe that a key issue will be the efficient storage of the shortest-path data of the nodes. (This data is indeed non-sparse so maybe, it should be calculated on the fly so to avoid the unnecessary computations and memory allocation for nodes that are too distant to fit into one box.)

- When considering more networks, it could be investigated if the performance of algorithms correlates with well-known network features. In our work, we only analysed the relative advantage of randomized algorithms (*merge, random sequential*) on 'no-hub' networks. These observations may help in selecting an appropriate algorithm for a given network.

- In addition to the implemented algorithms we tried out, one might want to modify **merge** and **MCWR** algorithms slightly for better performance, as detailed in the preceding section. More strategies of **greedy** should be tried out too.

- If scaling to larger networks is achieved, one should turn attention towards theoretical network models with known fractal-dimension. It would be a very natural and sensible method to assess the performance of box-covering algorithms to compare the measured fractal-dimension to the ground-truth value. Lately, we have performed such analysis on a limited number of network models (size was constrained by our architecture). We found a surprisingly big discrepancy between measured and theoretical values. Investigations support the idea that the ground truth fractal-dimension is only attained asymptotically with increasing network size. This issue must be dealt with in the long run.

- Testing more algorithms. In fact, we did not include all algorithms we implemented because some of them were too slow or included too many hyper-parameters (e.g. *simulated annealing, differential evolution, particle swarm optimization*).

References

[1] Random Graphs, Béla Bollobás, Cambridge University Press Online publication date: March 2011 Print publication year: 2001 Online ISBN: 9780511814068

[2] Jeong, H., Tombor, B., Albert, R. et al. The large-scale organization of metabolic networks. Nature 407, 651–654 (2000). https://doi.org/10.1038/35036627

[3] Song C, Havlin S, Makse HA. Self-similarity of complex networks. Nature. 2005 Jan 27;433(7024):392-5. doi: 10.1038/nature03248. PMID: 15674285.

[4] Song, Chaoming Gallos, Lazaros Havlin, Shlomo Makse, Hernan. (2007). How to calculate the fractal dimension of a complex network: The box covering algorithm. Journal of Statistical Mechanics: Theory and Experiment. 2007. 10.1088/1742-5468/2007/03/P03006

[5] https://en.wikipedia.org/wiki/Network_science (23.10.2020)

[6] Lazaros K. Gallos, Chaoming Song, Hernán A. Makse, A review of fractality and self-similarity in complex networks, Physica A: Statistical Mechanics and its Applications, Volume 386, Issue 2, 2007, Pages 686-691,

[7] https://upload.wikimedia.org/wikipedia/commons/e/ea/Sierpinski8.svg (23.10.2020)

[8] https://en.wikipedia.org/wiki/List_of_fractals_by_Hausdorff_dimension (23.10.2020)

[9] https://networkx.org/ (22.10.2020)

[10] Hernán D Rozenfeld, Shlomo Havlin, and Daniel Ben-Avraham. Fractal and transfractal recursive scale-free nets. New Journal of Physics, 9(6):175, 2007.

[11] Kuang, Li & Zheng, Bojin & Li, Deyi & Li, Yuanxiang & Sun, Yu. (2013). A Fractal and Scale-free Model of Complex Networks with Hub Attraction Behaviors. Science China Information Sciences. 58. 10.1007/s11432-014-5115-7.

[12] Kim, Js & Goh, K-I & Kahng, B. & Kim, Doochul. (2007). A box-covering algorithm for fractal scaling in scale-free networks. Chaos (Woodbury, N.Y.). 17. 026116. 10.1063/1.2737827.

[13] C. Yuan, Z. Zhao, R. Li, M. Li and H. Zhang, "The Emergence of Scaling Law, Fractal Patterns and Small-World in Wireless Networks," in IEEE Access, vol. 5, pp. 3121-3130, 2017, doi: 10.1109/ACCESS.2017.2674021.

[14] Chaoming Song et alJ. Stat. Mech. (2007) P03006

[15] Zheng, Wei & Pan, Qian & Chen, Sun & Deng, Yu-Fan & Zhao, Xiao-Kang & Kang, Zhao. (2016). Fractal Analysis of Mobile Social Networks. Chinese Physics Letters. 33. 038901. 10.1088/0256-307X/33/3/038901.

[16] Liao, Hao & Wu, Xingtong & Wang, Bing & Wu, Xiangyang & Zhou, Mingyang. (2019). Solving the speed and accuracy of box-covering problem in complex networks. Physica A: Statistical Mechanics and its Applications. 523. 10.1016/j.physa.2019.04.242.

[17] Locci, Mario & Concas, Giulio & Tonelli, Roberto & Turnu, Ivana. (2010). Three Algorithms for Analyzing Fractal Software Networks. WSEAS Transactions on Information Science and Applications. 7.

[18] Storn, R., Price, K. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997). https://doi.org/10.1023/A:1008202821328

[19] Kuang, Li. (2014). A differential evolution box-covering algorithm for fractal dimension on complex networks. 10.1109/CEC.2014.6900383.

[20] L. Kuang, F. Wang, Y. Li, H. Mao, M. Lin and F. Yu, "A discrete particle swarm optimization box-covering algorithm for fractal dimension on complex networks," 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, 2015, pp. 1396-1403, doi: 10.1109/CEC.2015.7257051.

[21] Sun, YuanYuan & Zhao, Yujie. (2014). Overlapping-box-covering method for the fractal dimension of complex networks. Physical Review E. 89. 10.1103/PhysRevE.89.042809.

[22] Haixin Zhang, Yong Hu, Xin Lan, Sankaran Mahadevan, Yong Deng, Fuzzy fractal dimension of complex networks, Applied Soft Computing, Volume 25, 2014, Pages 514-518, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2014.08.019.

[23] Wolfram Research, "Escherichia Coli Whole Network" from the Wolfram Data Repository (2019), Source Metadata: Hawoong Jeong, Bálint Tombor, Réka Albert, Zoltán N. Oltvai and Albert-László Barabási: The large-scale organization of metabolic networks Nature 407, 651 (2000)

[24] The Network Data Repository with Interactive Graph Analytics and Visualization, Ryan A. Rossi and Nesreen K. Ahmed, AAAI, 2015
http://networkrepository.com

[25] The Network Data Repository with Interactive Graph Analytics and Visualization, Ryan A. Rossi and Nesreen K. Ahmed, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015
http://networkrepository.com