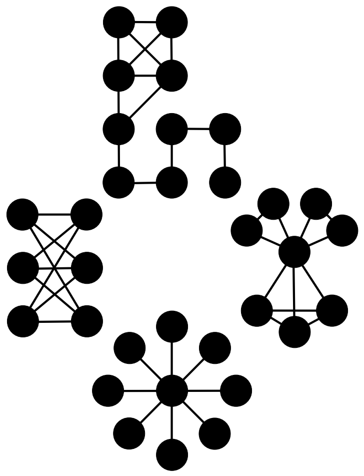
100 Vertices

Emiel Suilen
Supervised by Drs. D. van den Berg[1] and Dr. R. Bakhshi[1],
Prof. Dr. M. van Steen[1]
Prof. Dr. C. van Leeuwen[2]
epsuilen@few.vu.nl, d.vanden.berg@vu.nl, rbakhshi@few.vu.nl,
steen@cs.vu.nl
Cees.vanLeeuwen@ppw.kuleuven.be
Faculty of Exact Sciences, department of Computer Science
VU University Amsterdam[1]
Department of Psychology
Catholic University of Leuven[2]



January 24, 2013

Abstract

In graph theory, there are several properties of a graph, two of which are the cluster coefficient and the average path length. In this thesis, we research the variation in these two properties. In particular, for graphs of 100 vertices, we research what lower and upper bounds these properties can achieve. We describe these upper and lower bounds, and describe how these change under a growing number of edges. Using these bounds, we calculate how many different values these two properties can theoretically have. We use these limits to construct an algorithm, which finds graphs with specific properties. We deployed this algorithm on the DAS-3 and DAS-4 clusters, and plotted the results in a scatterplot, showing the effect of the number of edges on the two properties.

Many dots connect a line...

A thesis is never the work of a single person. Many people helped me in writing this, and they deserve my praise.

First of all, my two supervisors, Daan van den Berg and Rena Bakhshi. Daan, your energy and ideas were the key in deciding for this thesis. Thank you for tolerating my stubbornness, my humor and my consumption of the drop supply. We started the project long ago as between a student and a teacher, but it ended between friends. Rena, your endless cheer was important in dragging me through some of the harder parts of writing, and your comments were invaluable. This thesis would have been in a far worse shape without your input. And, now the word hammer[1] will never mean the same to me again.

I also thank Maarten van Steen for his critical comments. We can't rewrite standard mathematical theory and not mention it. I also want to thank Maarten and Cees van Leeuwen for being the second readers for this project. I hope they will enjoy the thesis.

I thank the many server admins to whom I became noticeable during this project. Cees van der Stoep, the other DAS4 admins, Spyros, Yannis, thank you for accepting my profuse apologies and not terminating my account.

I cannot forget the rest of the *Daanisten*, with who I spent many Mondays in discussing mathematics, bachelor projects, and general student nonsense. Roald, it was an honor to help you with your project. Mark, I am not very sorry for spamming your mail account by my runaway crawler-framework, but I am glad we did solve the last instance of Almost Squares. Florian, your insights are brilliant. I still believe your BSc project should have scored higher. Jeffrey, Patrick, Marijn, thank you for your comments, and good cheer during the meetings.

Joost Huizinga, Berend Weel, we discovered it, we named it. And nobody else knows.

Arjon, thank you for the feedback when looking for the APL tail formula. It gave me the idea for the floor and ceiling functions.

Christiaan, thank you for being there. I suppose the word pepernoten will never mean the same to you again, after my many, many pranks with them. I hope we can be guinea pigs to each others cooking for a long, long while.

Nina, thank you for helping me capture all the dotties. You frequently give me headaches from laughing, and I enjoy your laugh most of all. I always

[1]or saw, stapler, scissors...

count the days and I can't wait to hop[2] with you again.

And last, but not least, my parents and brother. I can only say thank you to my parents. No words exist to express my gratitude to them. Bastiaan, I am very proud at you finally studying medicine. I hope you can fulfill your dream, and become a great orthopedic surgeon.

[2]In this context, dance. Just to be clear.

Contents

Chapter 1

Introduction

Graphs are abstract representations of real life networks, with vertices to abstract the described objects, and edges between them to denote the links between the objects.

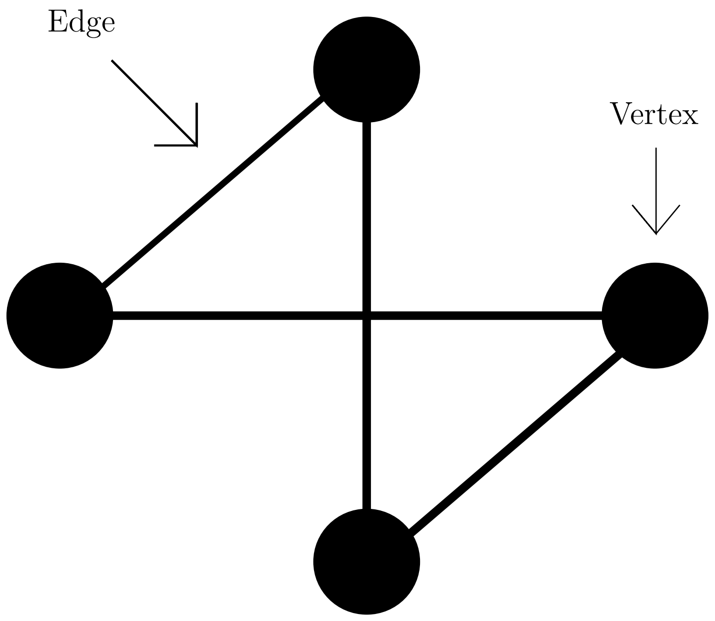


Figure 1.1: Example of a graph of 4 vertices

Graphs come in a variety of forms, but in this thesis, only undirected, unweighted, simple graphs are considered: all edges are equal, and if there is an edge from α to β, the same edge also connects β to α. There is only a single edge between two vertices, and no vertex has a loop. For notation purposes, throughout this thesis we will denote the set of vertices for a graph as V, and the size of the set $|V|$ as v. Similarly, a graph with a set of E edges has a total of $|E| = e$ edges. We will use G for a graph of 100 vertices, and an undetermined amount of edges. All considered graphs are connected, all vertices have at least one edge connecting them to the graph. To be connected, a graph of V vertices must have at least $v - 1$ edges. A fully connected graph, where every vertex is connected to all other vertices, has the following number of edges e:

$$e = \frac{v(v-1)}{2} \tag{1.1}$$

Throughout the thesis, we will consider *labeled* graphs, with the vertices labeled with lower case Greek letters. Graphs with differently labelled edges are considered nonisomorphic in this thesis, even if they correspond to the same unlabeled graph; for example, the two graphs in Figure 1.2 are not isomorphic.

α
β
γ
α
γ
β

(a) Graph $\alpha - \beta - \gamma$ (b) Graph $\alpha - \gamma - \beta$

Figure 1.2: Two non-isomorphic graphs

Graphs in real life: Graphs are used in various disciplines for the purpose of modeling. For example, in computer science graphs are used to represent different types of computer networks, in medicine to describe the human brain, in sociology to describe social networks and in biology to describe metabolic networks and foodwebs.

Mark Newman has written a thorough review of the daily occurence of graphs in his article *The structure and function of complex networks* [1]. Since this, the field of graphs has enjoyed tremendous interest [2] [3] [4] [5] [6]. The rise of social networking services like Facebook and Google+ has led to an increased interest in graphs, and their use in biology and other fields has grown even larger.

1.1 Structural properties of graphs

There are three common properties of a graph: cluster coefficient, average path length and degree distribution. In this thesis we will focus on the cluster

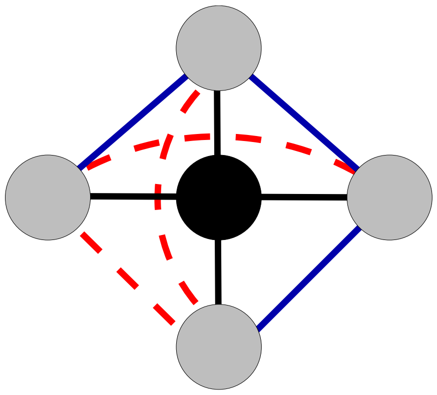


Figure 1.3: Example on how to calculate cluster coefficient.

coefficient and average path length.

1.1.1 Cluster Coefficient

One of the two values used in determining the configuration and quality of a graph is the cluster coefficient. The cluster coefficient defines how well connected vertices are among each other. The cluster coefficient has definitions for a single vertex, and for an entire graph.

The cluster coefficient CC_i for a vertex V_i is the ratio of the number of existing edges e_{exi} over the number of edges possible e_{pos} between vertices adjacent to V_i.

$$CC_i = \frac{e_{exi}}{e_{pos}} \tag{1.2}$$

So, for a graph of five vertices, as shown in Figure 1.3, the cluster coefficient of the central vertex can be calculated in the following way: three existing edges, divided by a total of six possible edges, and by Equation 1.2 it follows that the cluster coefficient is $\frac{1}{2}$.

This defines the cluster coefficient for any vertex that has two or more edges. There are also graphs with a vertex that has only one edge, e.g., see Figure 1.4. In this example, all black vertices have only one edge, and are thus connected to a single neighbor. Both the number of existing edges e_{exi}, as well as the number of possible edges e_{pos}, between that neighbor, is zero. This leads to a cluster coefficient of $\frac{0}{0}$, which is formally undefined. We will define a cluster coefficient value for such a vertex, but we can only do this, after introducing the cluster coefficient for an entire graph.

The cluster coefficient, CC, for an entire graph is the average of all cluster coefficients of the individual vertices of a graph. This gives us the following

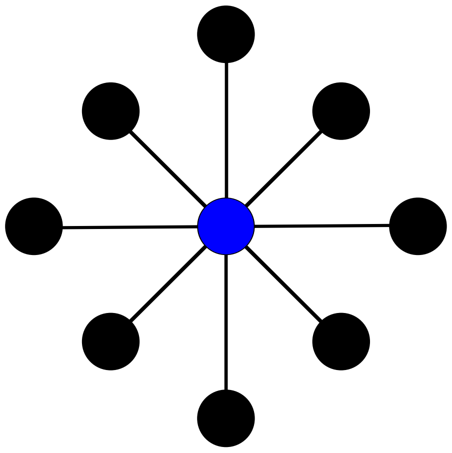


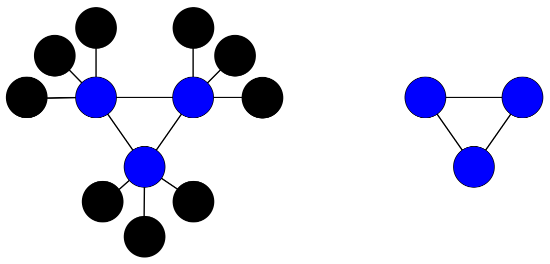
Figure 1.4: Hub graph

formula:

$$CC = \frac{1}{v}\sum_{i=1}^{v} CC_i \tag{1.3}$$

We define the cluster coefficient of a vertex with less than two edges to be zero, and we will do so for two reasons:

1. We would like two graphs of equal v and e to be comparable in their cluster coefficient, independent of the degree of their vertices. If a vertex with a single edge had an undefined cluster coefficient, and was not taken into account for the cluster coefficient of the entire graph, this would produce counterintuitive results, as shown in Figure 1.5.



(a) 12 vertices, 12 out of 66 possible edges

(b) 3 vertices, 3 out of 3 possible edges

Figure 1.5: If only the blue vertices were taken into account for the CC, both graphs would have a CC of 1.

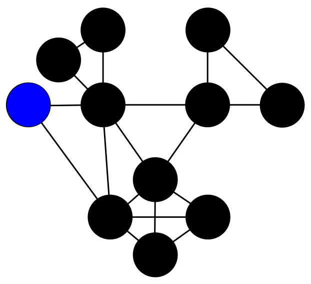
2. By keeping the cluster coefficient for vertices with degree smaller than 2 undefined, the removal of specific edges from a graph could raise its cluster coefficient. In our opinion, raising the cluster coefficient by removing edges is intuitively implausible and undesirable, as shown in Figure 1.6b.

This is a natural choice, and it has also been used for analysis of networks in general (e.g., [7, 8]) as well as for the complex systems (e.g., [9, 10, 11]). This special case is implemented in the commonly used NetworkX library of Python in the CC algorithm [12].

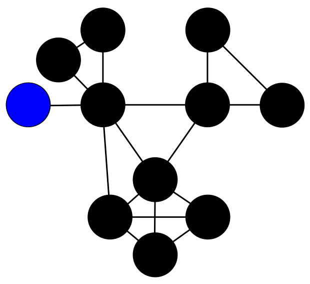
Now that we have the definition for the cluster coefficient for a vertex with a single edge, we can now define the cluster coefficient:

Definition 1.1.1

- *The cluster coeffcient of a vertex is* 0 *if it has degree smaller than 2.*
- *A vertex ϕ with degree n is connecting n vertices to vertex ϕ. The cluster coefficient of vertex ϕ is the number of edges connecting the n vertices with each other, divided by the potentional number of edges between the n vertices, and can be calculated with Equation 1.2.*
- *The cluster coefficient of a graph is the average of all cluster coefficient of its vertices, calculated by Equation 1.3.*



(a) With 11 vertices, this graph has a CC of 0.78



(b) With one edge less, this graph has a CC of 0.7

Figure 1.6: Defining the cluster coefficient as 0 for a vertex with a single edge

1.1.2 Average Path Length

The average path length of a graph G is the shortest path length between two vertices in that graph, averaged over all pairs of vertices.

Definition 1.1.2 *For a single vertex, the average path length of a vertex ϕ to the other $v-1$ vertices, is defined as follows: the average of the sum of the shortest paths between a single vertex, and all other vertices (with size of $v-1$) in the graph. For $\Delta(\phi,\psi)$ being the distance between vertex ϕ and vertex ψ, the average path length for vertex ϕ can be calculated as:*

$$APL_\phi = \frac{1}{v-1}\sum_{\psi=1}^{v-1}\Delta(\phi,\psi) \tag{1.4}$$

For the graph shown in Figure 1.7, the average path length of the black vertex is calculated by Definition 1.1.2, $\frac{3}{2}$.

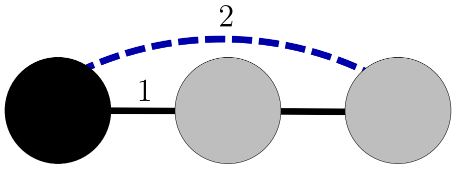


Figure 1.7: Example of how to calculate the average path length.

Definition 1.1.3 *For the entire graph, the average path length for a graph of v vertices is the sum of the individual average path lengths of each vertex.*

$$APL = \frac{1}{v}\sum_{i=1}^{v} APL_i \tag{1.5}$$

For example, the average path length of the graph in Figure 1.7 is thus $\frac{4}{3}$. Alternatively, the average path length of the same graph can be computed as depicted in Figure 1.8. The distance between any two adjacent vertices is 1, while the distance between the extreme vertices is 2. Adding all distances together, and dividing that sum by $v(v-1)$ will also produce the same average path length of $\frac{4}{3}$. So following the example of Figure 1.8:

$$\frac{(1+2)+(1+1)+(1+2)}{3\cdot(3-1)} = \frac{8}{6} = \frac{4}{3}$$

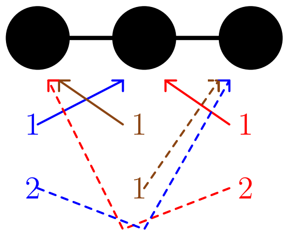


Figure 1.8: Individual distances from one vertex to another

1.1.3 Statespace

The combination of the cluster coefficient and the average path length can also be defined as a state of a graph, as a graph always has a cluster coefficient and an average path length. A graph with a fixed number of vertices and a set number of edges has a distinct number of states. Each one of these states represents one or more graphs with a specific cluster coefficient and a specific average path length.

All possible states of a graph with v vertices and e edges form the statespace for such a graph, distributed between the upper and lower bounds for the cluster coefficient and average path length for $G(v, e)$, and contains all possible combinations of cluster coefficient and average path length values a graph can have. All of these states can be projected onto a scatterplot. A scatterplot depicts states for which there exists at least one graph with such a state.

Every (red) point in the scatterplot represents an existing graph with a specific combination of cluster coefficient and average path length. Each point depicts one or more graphs, as isomorphic graphs will all be represented in a single point in the scatterplot. The scatterplot does not show how many graphs are found for any depicted point, and some locations in the statespace may represent combinations of cluster coefficient and average path length for which no graph G of v vertices and e edges is known to exist.

Figure 1.9 shows a sample scatterplot for a graph of 100 vertices and 300 edges.

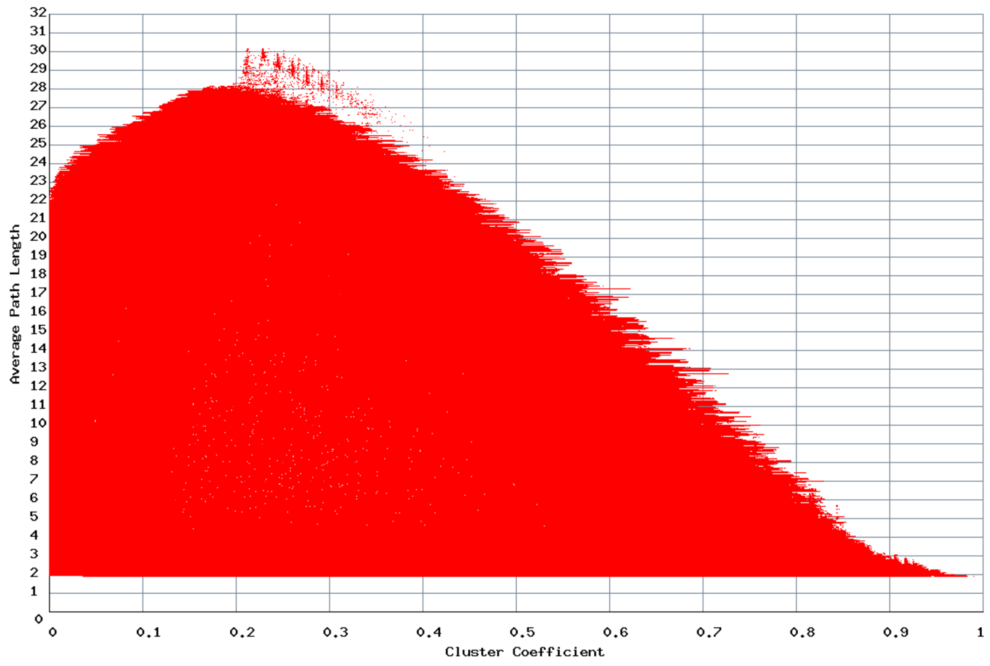


Figure 1.9: Sample scatterplot for a graph of 100 vertices and 300 edges

As mentioned before, a graph with v vertices and e edges has a statespace, which can be depicted in a scatterplot. A graph with another number of edges, not equal to e, will have another statespace, and is depicted by another scatterplot. This will be shown in this thesis.

1.2 Scope of the thesis

In this thesis, the statespace of graphs with 100 vertices will be explored, for $100, 150, 200, \ldots, 4950$ edges. Firstly, we'll identify upper and lower boundary values for the cluster coefficient and average path length for connected, undirected graphs, and their corresponding properties. Second, we'll explore the space between these boundaries by iteratively modifying connection configurations of the boundary cases to determine which combinations values of cluster coefficient and average path length can be attained. Although the method is not exhaustive, it gives a valuable insight in the variability of cluster coefficient and average path length and values for which graphs actually exist. For every number of edges e, we present a scatterplot of the values found in the corresponding statespace, resulting in 98 scatterplots, one for each value of $e \in \{100, 150, 200, \ldots, 4950\}$.

Chapter 2

Theoretical Limits

To understand the size and shape of the statespace, we have to specify the upper and lower bounds on the values for both the average path length (APL) and the cluster coefficient (CC).

2.1 Average Path Length

We first compute the upper and lower bound on the values of APL and CC for graphs $G = \langle V, E \rangle$, given $|V| = v$ and $|E| = e$.

2.1.1 Minimum Average Path Length: Star Graph

The lower bound on the value of the APL for a graph with v vertices and e edges equals 1. In such a graph any two vertices can reach each other in only one hop. In fact, this is only possible in a fully connected graph, which has exactly $\frac{v \cdot (v-1)}{2}$ edges.

For graphs with less than $\frac{v \cdot (v-1)}{2}$ edges, the lower bound on the APL is actually greater than 1 for the following reason: the shortest path between vertices ϕ and ψ, $\Delta(\phi, \psi)$ is 1 if and only if there is an edge $e_{\phi\psi} \in E$ between them. If there is no such edge, the shortest possible path length $\Delta(\phi, \psi)$ is equal to 2.

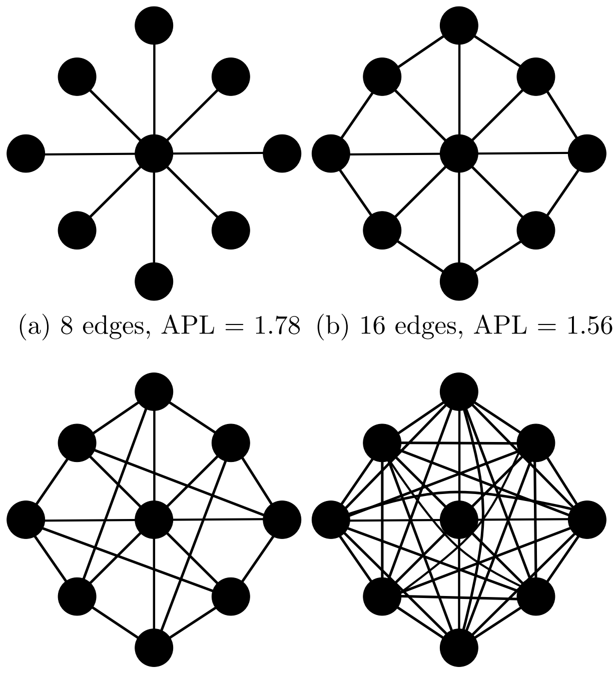
A graph with $v - 1$ edges has the minimal APL, if it is constructed as follows: one (central) vertex is connected to all other vertices, each by a single edge. This construction is likened to a star, or a *hub-and-spokes*; see the example in Figure 2.1a. The APL of such a graph is:

$$\begin{aligned} APL_{min} &= 2 - \frac{2(v-1)}{v(v-1)} \\ &= 2\left(1 - \frac{1}{v}\right) \end{aligned} \tag{2.1}$$

If more edges are added, each of these added edges $E_{\phi\psi}$ will reduce a shortest path length $\Delta(\phi, \psi)$ of 2 to a shortest path length $\Delta(\phi, \psi)$ of 1. The remaining edges, can be distributed among the peripheral vertices, and each of these added edges will reduce the APL by $2\left(\frac{1}{v} \cdot \frac{1}{v-1}\right)$, to an overall APL of:

$$APL_{star} = 2\left(1 - \left(\frac{1}{v} \cdot \frac{e}{v-1}\right)\right) \tag{2.2}$$

This is shown in Figure 2.1, where eight additional edges are added to the graph in Figure 2.1a, resulting in the graph on Figure 2.1b and and, subsequently, more edges are added until the graph reaches an APL of 1, as shown in Figure 2.1c and 2.1d.


(a) 8 edges, APL = 1.78 (b) 16 edges, APL = 1.56

(c) 20 edges, APL = 1.44 (d) 36 edges, APL = 1.00

Figure 2.1: Examples of Star-graphs

For a graph G and an increasing number of edges, the APL will decrease linearly, as shown in Figure 2.2.

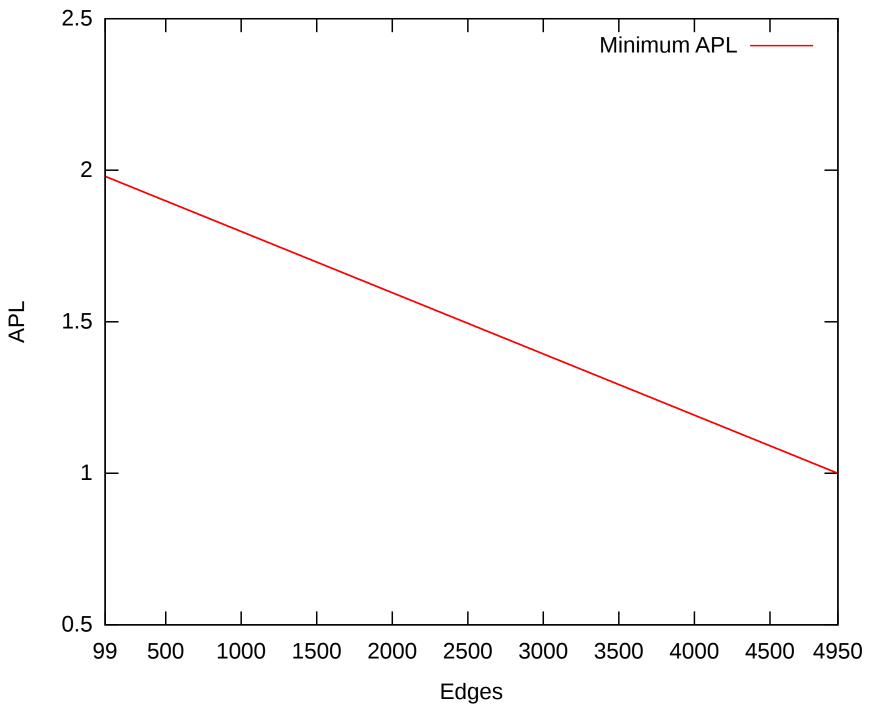


Figure 2.2: The development of the minimum APL over an increasing number of edges

2.1.2 Maximum Average Path Length: Tail graph

A graph with v vertices has the greatest APL possible if its vertices are arranged in a aline, as shown in Figure 2.3. However, such a line can only be constructed for a graph of v vertices and $v - 1$ edges. If a graph has more edges, it needs to be wired in such a way, that there still exists a long line, or tail, of vertices to increase its APL.

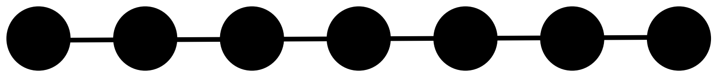


Figure 2.3: The graph with 7 vertices and the highest APL

In 2003, Lovejoy and Loch [13] introduced their *Clique Plus Path-graph*, (CPP-graph), a graph that has such a tail, and is believed to have, in specific

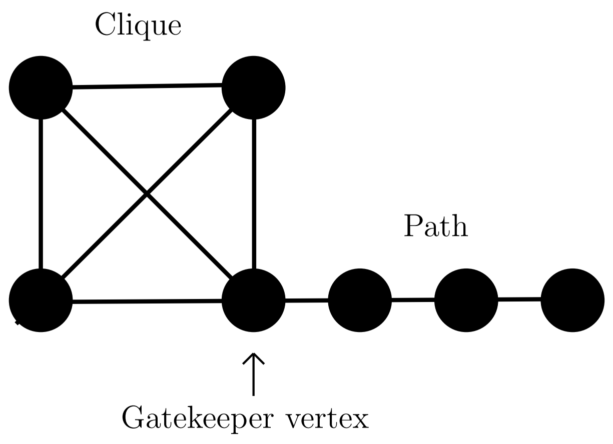


Figure 2.4: A Clique Plus Path graph, connected at the gatekeeper vertex

circumstances, the maximal APL. It consists of a densely connected group of vertices (or clique) and a tail, which is called a path in their paper. The tail is connected at the gatekeeper vertex to the clique, as demonstrated in Figure 2.4. The gatekeeper vertex is part of both the clique and the tail.

The idea behind the CPP-graph is that, when a graph has more than $v-1$ edges, it can, under certain circumstances, use a subset of the tail vertices to form a clique. Thus, graphs with v vertices and more than $v-1$ edges can be concstructed to increase their APL. The APL increases due to the vertices in the clique subgraph of such a CPP with the tail subgraph of the graph.

The APL of a CPP-graph cannot be increased by rewiring any single edge, but is highly dependent on the number of edges required to make the clique. In other wordsm the construction of a CPP graph restricts the number of edges the graph can have (i.e. for a clique subgraph and a tail subgraph). To relax this restriction, we introduce the *Tail-graph*, wherein a clique is connected to a long tail of vertices by a joint vertex. The joint vertex is connected by joint edges to the clique, as shown in Figure 2.5. This configuration eliminates the condition that the number of edges must be specific, to create both a clique and a tail. With any number of edges between $v-1$ and $\dfrac{v(v-1)}{2}$, a Tail-graph can be constructed.

The Tail-graph $G(V,E)$ consists of 3 subgraphs, with the following sets of vertices and edges:

- Clique: V_c vertices and E_c edges. These are colored red in Figure 2.5.
- Joint: a single vertex V_j, with E_j edges connecting it to the clique and

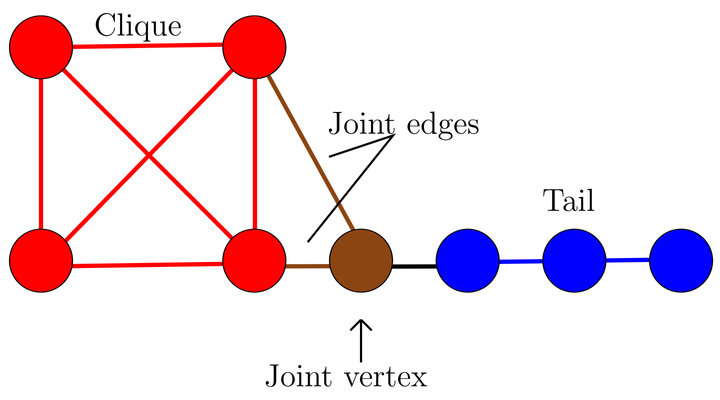


Figure 2.5: A Tail graph, connected at the joint vertex

a single edge to the tail. These are colored brown in Figure 2.5.

- Tail: V_t vertices and E_t edges. These are colored blue in Figure 2.5.

In order to compute the APL of a Tail graph, we need to provide the cardinality of the vertex and edge sets, and calculate the APL of each of three Tail subgraphs. We will start with the cardinalities, by showing how a Tail-graph can be constructed.

For a graph of v vertices and e edges:

1. Construct a line of v vertices and $v-1$ edges. If the number of edges is larger then $v-1$, $e-v-1=e_l$ edges are left to be used in the clique and the joint-edges.
2. These e_l edges are used to form a clique of v_c vertices. A clique of v_c vertices uses v_c vertices from the tail, as well as v_c-1 edges from the tail, to form the clique. Figure 2.6 shows the v_c vertices in red, using v_c-1 blue edges from the tail subgraph to construct part of the clique.

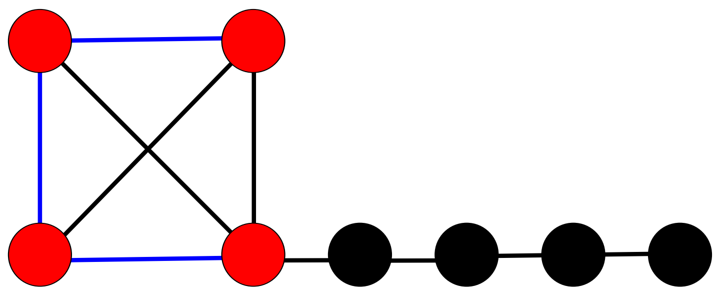


Figure 2.6: A Tail graph, using v_c and v_c-1 edges from the tail.

3. To determine v_c, we find the maximum number of vertices v_c where:

$$\begin{aligned} e_l &\geq \frac{1}{2}v_c(v_c-1)-(v_c-1) \\ e_l &\geq \frac{1}{2}(v_c-1)^2 \end{aligned} \tag{2.3}$$

   Hence, up to $\frac{1}{2}(v_c-1)^2 = e_c$ edges are used in this clique. If $e_l = e_c$, the Tail-graph is also a CPP-graph.

4. Any remaining edges are the e_j joint edges, in addition to a single edge from the tail:

$$e_j = e_l - e_c + 1 \tag{2.4}$$

   This is also shown in Figure 2.7, with the remaining edge colored red, and the edge from the tail colored blue. Both of these edges connect the clique with the blue joint-vertex.

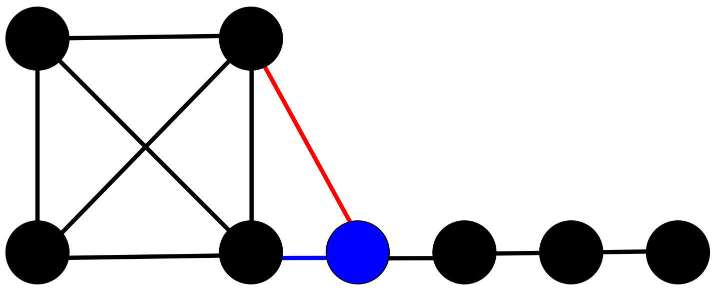


Figure 2.7: A Tail graph, with 2 joint edges.

5. The number of vertices in the tail, v_t, is the number of vertices of the graph, minus the number of vertices in the clique, minus the joint vertex:

$$v_t = v - v_c - 1 \tag{2.5}$$

   This also determines the number of edges in the tail, as $e_t = v_t - 1$.

Note that, $v = v_c + v_j + v_t$, but $e = e_c + e_j + e_t + 1$. The single remaining edge connects the joint vertex to the tail, and is colored black in Figure 2.5.

With the cardinalities of the sets determined, we can now calculate the APL of the Tail-graph. To do so, all individual paths of the vertices in the Tail subgraphs are added together, and then averaged. We observe, that even though the path between vertices $\Delta(\phi,\psi) = \Delta(\phi,\psi)$, this will not result in a greater path length, as in the end the result is divided for by all vertices, so $\frac{\Delta(\phi,\psi)+\Delta(\phi,\psi)}{2}$. Calculating the path lengths of the Tail subgraphs can be done in the following way:

- From the clique-vertices to other clique-vertices, the joint vertex and the tail vertices. The clique has v_c vertices and e_c edges:
  - Any vertex in the clique is connected to all the other vertices in the clique by a path of length 1. Therefore, the sum of the APLs of the vertices in the clique, to other vertices in the clique, is:

    $$1 \cdot v_c \cdot (v_c - 1) \tag{2.6}$$

    This coincides with the maximum number of edges from Definition 1.1.
  - The clique also has connections to the joint, with e_j paths of a length 1, and $v_c - e_j$ paths of a length 2:

    $$e_j + 2(v_c - e_j) \tag{2.7}$$

    This is shown in Figure 2.8. The clique has e_j vertices (colored blue) which have a path of length 1 to the (yellow colored) joint vertex. (The edges for these paths are also colored blue.) The remaining $v_c - e_j$ vertices in the clique (colored red) do not have a path of length 1 to the joint vertex. These remaining $v_c - e_j$ vertices do have a path of length 2, by having a path of length 1 to a (blue) vertex, which has a path of length 1 to the (yellow) joint-vertex. Adding the lengths of these two paths together produces a path with a length 2 for the (red) vertices.

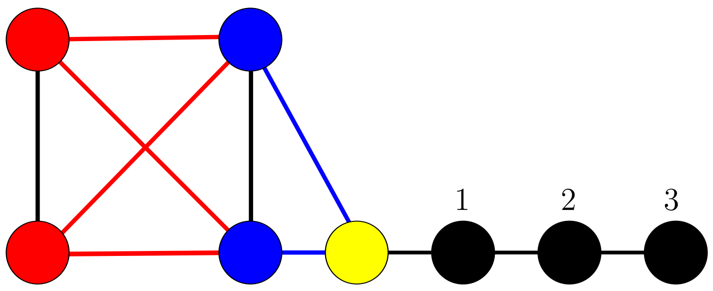


Figure 2.8: A Tail graph, with colored vertices to show the connections for the clique vertices. The numbers above the black vertices show the path length to the joint vertex (colored yellow).

  - All vertices in the clique also have paths to the vertices in the tail, and all of these paths traverse the joint-vertex. This means that the length of the path between the clique vertices and the tail vertices is the same as the length of the path between the clique vertices and the joint vertex, multiplied by the number of tail vertices, and the path between the joint vertex and the tail vertices. As Figure 2.8 shows, this is a sum of the number of tail subgraph vertices: $\sum_{i=1}^{v_t} i$. Thus, we combine the product of Formula 2.7 and the number of tail vertices with the sum of the tail subgraph vertices:

$$v_t(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i \tag{2.8}$$

This yields the following equation for the APLs of the vertices in the clique subgraph of the Tail-graph:

$$APL_{clique} = \frac{1}{v(v-1)} \cdot \left(v_c(v_c - 1) + (v_t + 1)(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i \right) \tag{2.9}$$

- Joint: The joint vertex has paths to the clique subgraph, and to the tail subgraph vertices.
  - For the vertices in the clique, there are e_j paths of length 1 between the joint-vertex and the clique, and $v_c - e_j$ paths of length 2. These are the same paths as explained in Equation 2.7:

$$e_j + 2(v_c - e_j)$$

  - For paths to the vertices in the tail, the path length is the sum of the vertices in the tail:

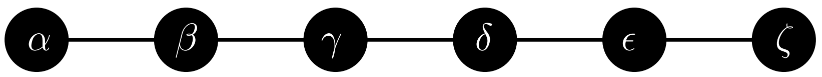
$$\sum_{i=1}^{v_t} i \tag{2.10}$$

This yields the following equation for the APL of the joint vertex:

$$APL_{joint} = \frac{1}{v(v-1)} \left(e_j + 2(v_c - e_j) + \sum_{i=1}^{v_t} i \right) \tag{2.11}$$

- Tail: the tail vertices of the graph have paths to the clique, to the joint, and to the other tail vertices.

– For all paths between any pair of tail subgraph vertices, the average paths are a sum of all such paths, leading to the following effect (which was explained previously in Figure 1.8.) Let us explain this on an example of six vertices, as shown in Figure 2.9.



$\Delta(\alpha,\beta)=1$	$\Delta(\beta,\alpha)=1$	$\Delta(\gamma,\beta)=1$	$\Delta(\delta,\gamma)=1$	$\Delta(\epsilon,\delta)=1$	$\Delta(\zeta,\epsilon)=1$
$\Delta(\alpha,\gamma)=2$	$\Delta(\beta,\gamma)=1$	$\Delta(\gamma,\delta)=1$	$\Delta(\delta,\epsilon)=1$	$\Delta(\epsilon,\zeta)=1$	$\Delta(\zeta,\delta)=2$
$\Delta(\alpha,\delta)=3$	$\Delta(\beta,\delta)=2$	$\Delta(\gamma,\alpha)=2$	$\Delta(\delta,\beta)=2$	$\Delta(\epsilon,\gamma)=2$	$\Delta(\zeta,\gamma)=3$
$\Delta(\alpha,\epsilon)=4$	$\Delta(\beta,\epsilon)=3$	$\Delta(\gamma,\delta)=2$	$\Delta(\delta,\zeta)=1$	$\Delta(\epsilon,\beta)=3$	$\Delta(\zeta,\beta)=2$
$\Delta(\alpha,\zeta)=5$	$\Delta(\zeta,\epsilon)=4$	$\Delta(\gamma,\zeta)=3$	$\Delta(\delta,\alpha)=3$	$\Delta(\epsilon,\alpha)=4$	$\Delta(\zeta,\alpha)=1$

Figure 2.9: Example to show the path length of vertices in the tail subgraph

Let us sum the individual steps. Each new row of paths to another vertex grows in a specific fashion, with $v = 6$:

α	β	γ	δ	ϵ	ζ	Sum	Formula		
1	1	1	1	1	1	6	v	$=$	$1v+0$
2	1	1	1	1	2	8	$v+2$	$=$	$1v+2$
3	2	2	2	2	3	14	$v+6$	$=$	$2v+2$
4	3	2	2	3	4	18	$v+12$	$=$	$2v+6$
5	4	3	3	4	5	24	$v+18$	$=$	$3v+6$

Note that, in the right hand side of the formula, the factor of v increases every odd step, while the second summand increases every even step. This yields the following formula:

$$\sum_{i=1}^{v_t-1}\left(\left\lceil\frac{i}{2}\right\rceil v_t+\sum_{j=1}^{\left\lfloor\frac{i}{2}\right\rfloor}2j\right) \tag{2.12}$$

- For the paths from the tail vertices to the joint vertex, Formula 2.10 applies again:

$$\sum_{i=1}^{v_t} i$$

- For the paths to the vertices in the clique, Formula 2.8 applies again:

$$v_t(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i$$

Thus, for the tail vertices combined, this produces the following formula:

$$\begin{aligned} APL_{combined} &= \frac{1}{v(v-1)} \left(\sum_{i=1}^{v_t - 1} \left(\left\lceil \frac{i}{2} \right\rceil v_t + \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} 2j \right) + \right. \\ &\quad \left. \sum_{i=1}^{v_t} i + v_t(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i \right) \end{aligned} \tag{2.13}$$

Finally, putting all terms of Formula 2.9, Formula 2.11 and Formula 2.13 together results in the following formula:

$$\begin{aligned} APL_{tail} &= \frac{1}{v(v-1)} \cdot \left(v_c(v_c - 1) + (v_t + 1)(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i + \right. \\ &\quad e_j + 2(v_c - e_j) + \sum_{i=1}^{v_t} i + \sum_{i=1}^{v_t - 1} \left(\left\lceil \frac{i}{2} \right\rceil v_t + \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} 2j \right) + \\ &\quad \left. \sum_{i=1}^{v_t} i + v_t(e_j + 2(v_c - e_j)) + v_c \sum_{i=1}^{v_t} i \right) \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{v(v-1)} \cdot \Bigg(v_c(v_c-1) + (2v_t+2)(e_j + 2(v_c - e_j)) + (2v_c+2)\sum_{i=1}^{v_t} i + \\
&\quad \sum_{i=1}^{v_t-1}\Bigg(\left\lceil \frac{i}{2} \right\rceil v_t + \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} 2j\Bigg)\Bigg) \\
&= \frac{1}{v(v-1)} \cdot \Bigg(v_c^2 + 2v_t(2v_c - e_j) + 3v_c - 2e_j + (2v_c+2)\frac{v_t(v_t+1)}{2} + \\
&\quad \sum_{i=1}^{v_t-1}\Bigg(\left\lceil \frac{i}{2} \right\rceil v_t + \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} 2j\Bigg)\Bigg)
\end{aligned} \tag{2.14}
$$

We observe that, for a graph G and an increasing number of edges, the maximal APL of the Tail-graph decreases exponentially, while the minimal APL decreases linearly, as shown in Figure 2.10.

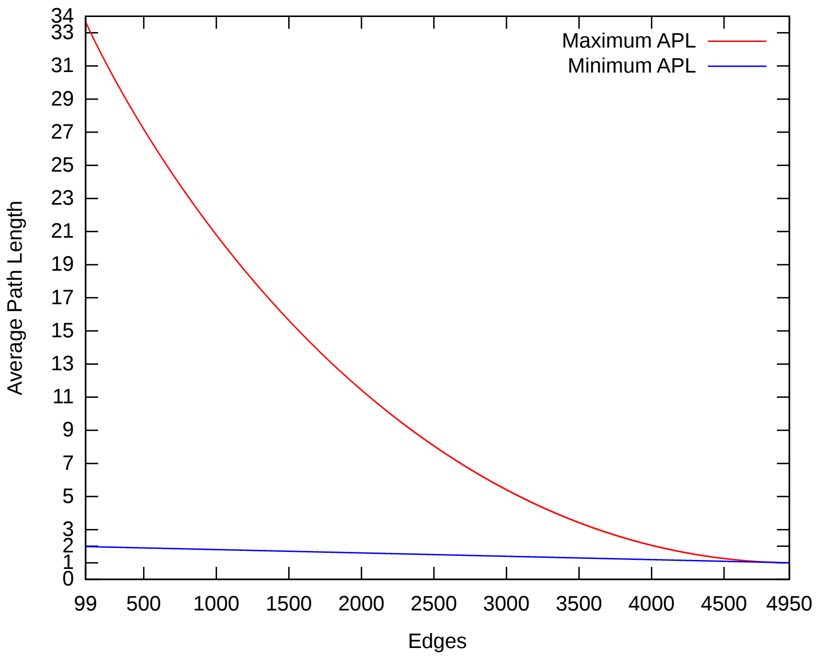


Figure 2.10: The development of the maximum APL and minimal APL with an increasing number of edges

2.2 Cluster Coefficient

Graphs, referred in this thesis as Distributed Map and JEB-graphs, are graphs with the minimal and maximal CC, respectively. We say that a graph has minimal (maximal) CC, if there is no other graph with lower (higher) CC for a given number of vertices and edges, respectively.

2.2.1 Minimal Cluster Coefficient: Distributed Map Graph

We first explain the lower bound on the value of the CC.

For graphs with v vertices and $v-1$ edges, all graphs have a CC 0.

For graphs with v vertices and $v-1 < e \leq \frac{1}{4}v^2$ edges, Takahashi [5] has shown that there are graphs with a CC 0. Such a graph, $G(V,E)$ with v vertices and e edges, can be constructed in the following way:

1. Divide the set of V vertices into two sets, L for the left set, and R for the right set. Both subsets are of equal size when the number of vertices, v, are even, and are of unequal size, otherwise.

2. Every vertex from set L has to be connected to all the other vertices from set R. If l and r are used to indicate the size of set L and set R respectively, then the total number of edges wired in such a way is equal to $l \cdot r$, or $\frac{1}{2}v \cdot \frac{1}{2}v = \frac{1}{4}v^2$.

A graph with the edges wired in an even distribution between the subsets L and R, is shown in Figure 2.11.

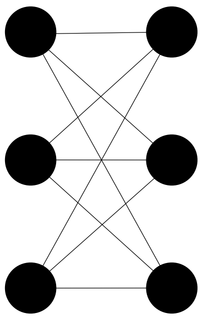


Figure 2.11: A graph of 6 vertices with the CC 0 and the largest number of edges for such a CC.

This means that, for graphs of 100 vertices, a wiring of edges can be found with the CC 0, and the number of edges can be 99 or larger, but smaller than 2501.

When the number of edges in a graph is larger than $\frac{1}{4}v^2$, its CC can no longer be 0. Any edge added to subset L or R will impact two vertices from the set in which it was added, and all vertices from the opposite set. This is shown in Figure 2.12, where one (blue) edge is added between vertex α and β, which also changes the cluster coefficients of vertices δ, ϵ and ζ.

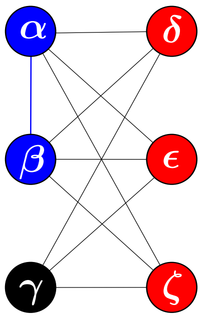


Figure 2.12: One added edge impacts 5 vertices.

To find the graph with the lowest CC, we have to minimize the increase in the CC when edges are added to such a graph. This can be done by maximizing the number of edges a single vertex has from one of the subsets. Because the formula of the CC is a fraction, the effect of adding a single edge to the numerator part is smaller when the denominator part of the fraction is larger; e.g. for any $n, d \neq 0, x = \frac{n+1}{d+1} - \frac{n}{d+1}, y = \frac{n+1}{d} - \frac{n}{d}, x < y$.

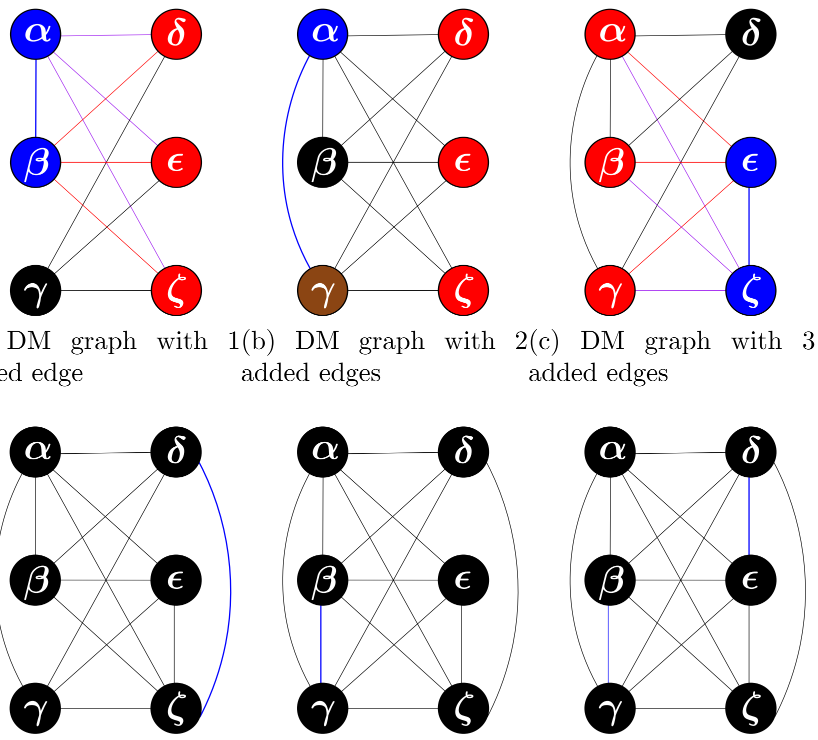
As a reminder, the numerator of the CC is the number of existing edges between the adjacent vertices, while the denominator is the total number of possible edges. The denominator increases when the degree of a vertex increases, and thus, maximizing the degree of a vertex will minimize the effect of adding an edge between the vertices adjacent to that vertex. This increase of the degree is done by connecting a vertex from the subset $L(R)$ to all other vertices in the subset $L(R)$, respectively.

Every vertex of the subset L is already connected with all vertices in the subset R, and vice versa. This forces us to alternate the maximization of the degree of vertices. First a vertex from the subset L needs its degree maximized, then the degree of a vertex from subset R, then the subset L, and so on. Otherwise, one of the two subsets would eventually get vertices with the CC 1, thus actually maximizing the CC, instead of minimizing it.

To demonstrate the effect of adding edges in such a fashion to a graph with the CC 0 and $\frac{1}{4}v^2$ edges, consider the graph in Figure 2.11. We will now describe the effects (on the CC) of adding a new edge to such a graph. We call such graph a *Distributed Map*-graph (DM-graph). We do not give a proof that, for a graph with more than $\frac{1}{4}v^2$ edges, the value of the CC of the

DM-graph is the theoretical lowest bound. Note that, even with extensive rewiring of all DM-graphs, we never found a graph with a smaller CC, with the same number of edges.

For example, let us add an edge between two vertices in the subset L. It will impact two vertices, α and β, from subset L, and all the vertices from the subset R, which is shown in Figure 2.13a.



(a) DM graph with 1 added edge

(b) DM graph with 2 added edges

(c) DM graph with 3 added edges

(d) DM graph with 4 added edges

(e) DM graph with 5 added edges

(f) DM graph with 6 added edges

Figure 2.13: DM graph with added edges

- Vertices α and β have its CC increased by $\frac{2r}{r(r-1)}$. That is, both vertices have four vertices adjacent to them: α has the vertices β, δ, ϵ and ζ, and β has α, δ, ϵ and ζ. Between the adjacent vertices of α, there are six possible edges. However, only the three (red) edges exist. The same applies for vertex β, which only has the three (purple) edges. This gives both of the (blue) vertices a CC $\frac{3}{6}$.

- Vertex γ in L, without a new edge, has a CC 0. The vertices adjacent to γ do not have any edges between them, they are not interconnected.

- For the subset R, the CC is increased by $\frac{2\cdot 1}{l(l-1)}$. Each (red) vertex has three adjacent vertices (vertices α, β and γ) from subset L, but only one edge exists between them, between vertices α and β. This gives each of the (red) vertices the CC $\frac{1}{3}$.

Table 2.1, also shows the CCs for all the individual vertices.

Vertex	Neighbors	Edges between neighbors	CC
α	β, δ, ϵ, ζ	$\beta\delta$, $\beta\epsilon$, $\beta\zeta$	$\frac{2\cdot 3}{4\cdot 3} = \frac{3}{6}$
β	α, δ, ϵ, ζ	$\alpha\delta$, $\alpha\epsilon$, $\alpha\zeta$	$\frac{2\cdot 3}{4\cdot 3} = \frac{3}{6}$
γ	δ, ϵ, ζ	-	0
δ	α, β, γ	$\alpha\beta$	$\frac{2\cdot 1}{3\cdot 2} = \frac{2}{6}$
ϵ	α, β, γ	$\alpha\beta$	$\frac{2\cdot 1}{3\cdot 2} = \frac{2}{6}$
ζ	α, β, γ	$\alpha\beta$	$\frac{2\cdot 1}{3\cdot 2} = \frac{2}{6}$

Table 2.1: CC of individual vertices with one added edge

Hence, by adding a single edge, the CC of the DM-graph is now $\frac{1}{6} \cdot (\frac{3}{6} + \frac{3}{6} + 0 + \frac{2}{6} + \frac{2}{6} + \frac{2}{6}) = \frac{1}{6} \cdot \frac{12}{6} = \frac{2}{6}$, and this is the smallest possible increment.

To continue with the example, we will now add a second edge. This edge has to be added to one of the two vertices from subset L. This is shown in Figure 2.13b. The vertex that is now adjacent to α is labeled γ, and is shown in brown color. Then:

- The CC of α is increased by $\frac{2\cdot 1}{(r+1)(r+2)}$. This increase of the CC is smaller, compared to the CC of the graph with a new edge between β and γ. Such a new edge would have resulted in an increase of the CC by $2 \cdot \frac{2r}{r(r-1)}$.
- The CC of β remains $\frac{3}{6}$.
- The CC of γ is increased by $\frac{2\cdot 2}{r(r-1)}$, and is now $\frac{3}{6}$.
- The CC of each vertex of the subset R is increased by $\frac{2\cdot 1}{l(l-1)}$, to a total of $\frac{2\cdot 2}{l(l-1)}$.

So, the CC of α is $\frac{6}{10}$. Since α is now connected to all other vertices, its CC will linearly increase with every edge added to the graph. This is, for this vertex, the smallest possible increase of CC.

We say that, whenever a vertex is adjacent to all other vertices in the graph, it not only belongs to the subset L or R, but also to the subset M.

Vertex	Neighbors	Edges between neighbors	CC
α	β, γ, δ, ϵ, ζ	$\beta\delta$, $\beta\epsilon$, $\beta\zeta$, $\gamma\delta$, $\gamma\epsilon$, $\gamma\zeta$	$\frac{2\cdot6}{5\cdot4} = \frac{6}{10}$
β	α, δ, ϵ, ζ	$\alpha\delta$, $\alpha\epsilon$, $\alpha\zeta$	$\frac{2\cdot3}{4\cdot3} = \frac{3}{6}$
γ	α, δ, ϵ, ζ	$\alpha\delta$, $\alpha\epsilon$, $\alpha\zeta$	$\frac{2\cdot3}{4\cdot3} = \frac{3}{6}$
δ	α, β, γ	$\alpha\beta$, $\alpha\gamma$	$\frac{2\cdot2}{3\cdot2} = \frac{4}{6}$
ϵ	α, β, γ	$\alpha\beta$, $\alpha\gamma$	$\frac{2\cdot2}{3\cdot2} = \frac{4}{6}$
ζ	α, β, γ	$\alpha\beta$, $\alpha\gamma$	$\frac{2\cdot2}{3\cdot2} = \frac{4}{6}$

Table 2.2: CC of individual vertices with two added edges

E.g. vertex α is adjacent to vertices $\beta, \gamma, \delta, \epsilon$ and ζ, and vertex α is now also in subset M.

The CC of the entire graph is now $\frac{1}{6} \cdot (\frac{6}{10} + \frac{3}{6} + \frac{3}{6} + \frac{4}{6} + \frac{4}{6} + \frac{4}{6}) = \frac{3}{5}$. By adding a single edge, the CC of the entire graph is increased by $\frac{8}{30}$, from $\frac{2}{6}$ to $\frac{3}{5}$.

We again continue the example, and add another edge. Similair to the vertices of the subset L, we now add edges to a single vertex in the subset R. We will add an edge being between two vertices in the subset R, the vertices ϵ and ζ in Figure 2.13c.

- Like vertices α and β, both vertices ϵ and ζ have four neighbors (another blue vertex and the red vertices). However, in contrast to Figure 2.13a, there are more edges: the three edges from both ϵ and ζ towards α, β and γ, and the two edges added to subset L, with a total of $3 + 2 = 5$ edges. Thus, both vertices ϵ and ζ now have the CC $\frac{5}{6}$.

- The remaining vertex δ keeps its CC $\frac{2}{3}$.

- Vertices α, β and γ are also affected by this added edge. Vertex α now has the CC $\frac{7}{10}$, as it is in subset M, and any extra edge will affect its CC.

- Vertices β and γ have their CC increased to $\frac{4}{6}$.

Thus, the CC of the resulting graph, is $\frac{1}{6} \cdot (\frac{7}{10} + 2(\frac{4}{6}) + 2(\frac{5}{6}) + \frac{4}{6}) = \frac{131}{180}$.

Three remaining edges can be added, and each will increase the CC in the same way. All vertices in the set M are affected, all vertices from the opposite subset are affected, and all vertices that have the new edge attached are affected. By now, any added edge will affect most other vertices in the graph. This is shown in Figures 2.13d-2.13f.

In fact, the DM-graph is the graph with the smallest possible CC, because it has as many vertices as possible in the subset M.This ensures that any edge that is added will have the smallest possible increase in CC.

Vertex	Neighbors	Edges between neighbors	CC
α	$\beta,\ \gamma,\ \delta,\ \epsilon,\ \zeta$	$\beta\delta,\ \beta\epsilon,\ \beta\zeta,\ \gamma\delta,\ \gamma\epsilon,\ \gamma\zeta,\ \epsilon\zeta$	$\frac{2\cdot7}{5\cdot4} = \frac{7}{10}$
β	$\alpha,\ \delta,\ \epsilon,\ \zeta$	$\alpha\delta,\ \alpha\epsilon,\ \alpha\zeta,\ \epsilon\zeta$	$\frac{2\cdot4}{4\cdot3} = \frac{4}{6}$
γ	$\alpha,\ \delta,\ \epsilon,\ \zeta$	$\alpha\delta, \alpha\epsilon, \alpha\zeta, \epsilon\zeta$	$\frac{2\cdot4}{4\cdot3} = \frac{4}{6}$
δ	α, β, γ	$\alpha\beta, \alpha\gamma$	$\frac{2\cdot2}{3\cdot2} = \frac{4}{6}$
ϵ	α, β, γ	$\alpha\beta, \alpha\gamma, \alpha\delta, \beta\delta, \gamma\delta$	$\frac{2\cdot5}{3\cdot2} = \frac{5}{6}$
ζ	α, β, γ	$\alpha\beta, \alpha\gamma, \alpha\zeta, \beta\zeta, \gamma\zeta$	$\frac{2\cdot5}{3\cdot2} = \frac{5}{6}$

Table 2.3: CC of individual vertices with three added edges

The CC of a DM-graph can be formulated, but it essentially is a formula for counting added edges. We therefore refer to the formula and its explanation in Addendum B. Figure 2.14 shows the increase of the CC for a growing number of edges.

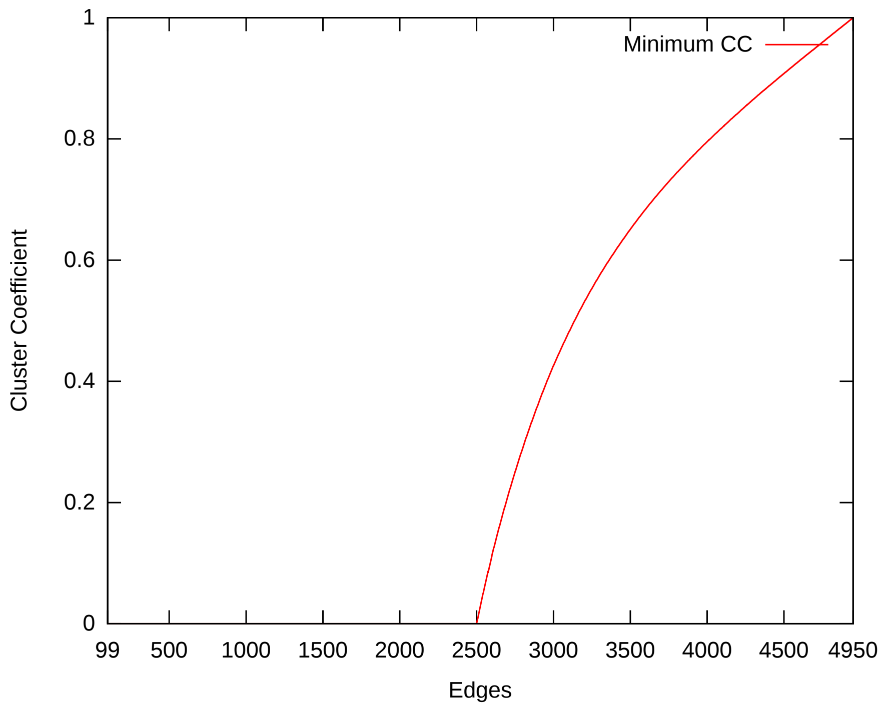


Figure 2.14: The development of the minimum CC for different number of edges

2.2.2 Maximal Cluster Coefficient

The maximal CC of a graph is 1. Such a graph is fully connected, and it has (according to Equation 1.1) $e_{max} = \frac{v(v-1)}{2}$ edges. For a graph with fewer edges than that, the CC 1 can not be achieved.

Note that the CC 1 for an entire graph can only be achieved if all vertices have an individual CC 1 (according to Equation 1.3). When a graph has fewer edges than e_{max}, the graph can not have the CC 1. The greatest CC a graph of v vertices and less than e_{max} edges can have, is a CC $> \frac{v-1}{v}$. A graph has such a CC when $v-1$ vertices have an individual CC 1, and when the remaining vertex has a CC that is as great as possible. This results in the CC $> \dfrac{v-1}{v}$, or in the case of the graph G, a CC $> \dfrac{99}{100}$.

Such a CC can be constructed in the following way:

- One central vertex, connected to all other vertices; this is a graph with minimal APL, and is a Star-graph.
- All other vertices are connected in cliques, thereby making every vertex in such a clique to have the CC 1.

This forms a so called *JEB-graph*, shown in Figure 2.15. Note that JEB-graphs are a subclass of the Star-graphs, and have the minimal APL.

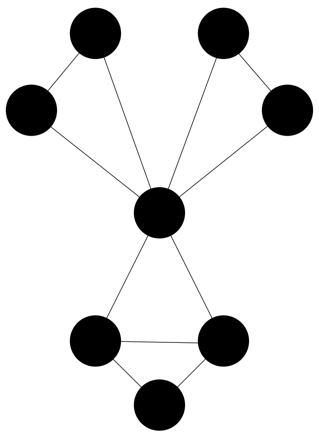


Figure 2.15: A JEB-graph of 8 vertices with the CC 0.905

Since the central vertex is connected to all the other vertices, its CC has in the nominator the maximum number of connections for a graph of $v-1$ edges.

$$CC_{max} = \frac{1}{v} \cdot \left(\frac{2(e-v+1)}{(v-1)(v-2)} + v - 1 \right) \tag{2.15}$$

The term $v - 1$ in Formula 2.15 signifies the CC of the vertices in the cliques, and the fraction is the CC of the central vertex.

The existence of a JEB-graph depends on the number of edges, and the possibility to form cliques with those edges. This is not possible for all graphs of 100 vertices. The smallest number of edges, required to construct a JEB-graph, is 150. This JEB-graph has a specific number of wirings, which is shown in Table 2.4. Note that the JEB-graph also has a central vertex, with edges to all 99 other vertices.

Cliques and vertices per clique	Edges per clique	Total edges
48 cliques of 2 vertices	1	48
1 clique of 3 vertices	3	3

Table 2.4: The configuration for a JEB-graph of 150 edges

For the number of edges < 150, another wiring of the edges is needed. Again, the number of vertices with the CC 1 is maximized, but without a central vertex connecting every other vertex. Instead, an *almost-clique* is formed, where one of the vertices of the clique subgraph serves as a bridge between the clique and the rest of the graph. All vertices, except the connecting (bridge) vertex, have the CC 1. The vertex connects the clique to the rest of the graph has a CC $< \frac{v-1}{v}$, typically $\frac{v_c - 1}{v_c}$. An example of such a graph can be seen in Figure 2.16.

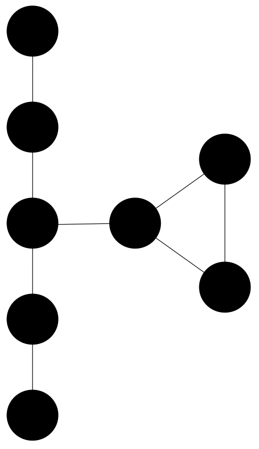


Figure 2.16: A graph of 8 vertices, 8 edges and the maximum CC 0.333

Combining both the JEB-graphs and the almost-clique graphs results in the plots of Figure 2.17, showing the growth of the CC under a growing number of edges.

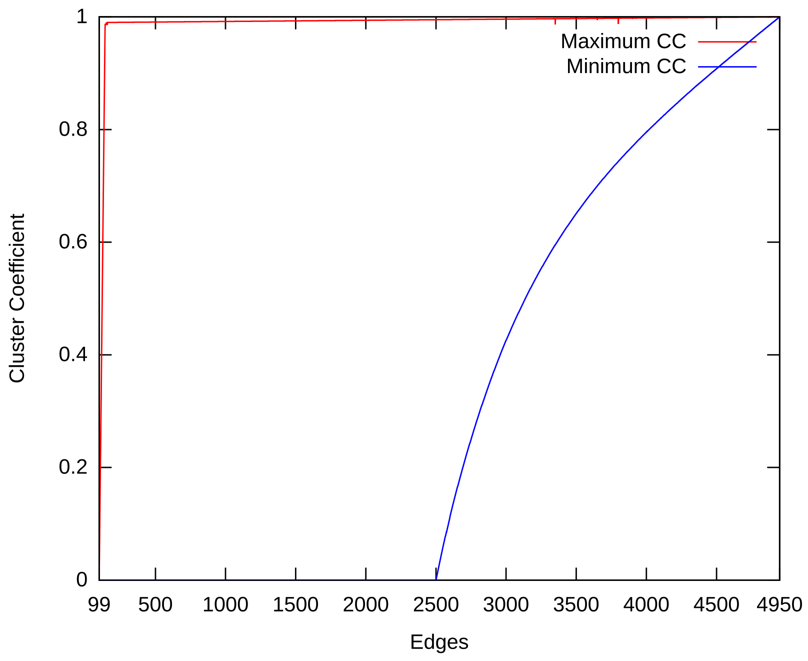


Figure 2.17: The development of the maximal (and minimal) CC with an increasing number of edges

Chapter 3

Statespace

Now that the limits of the statespace are identified, we can determine the number of states in the statespace itself, by determining the number of possible values of the CC and the APL for G.

3.1 Resolution

To determine the total number of possible states, we need to derive the number of possible values for the CC and APL, in G.

3.1.1 Cluster Coefficient

As Definition 1.1.1 states, the CC of a graph G is defined by the CC of its vertices, which are, in turn, defined by the number of existing edges between its neighboring vertices. The degree of a single vertex in a connected graph of v vertices ranges in $[1, v-1]$. This means that the CC of a vertex is always a fraction with a nominator of at least 1, and is at most $\frac{(v-1)(v-2)}{2}$. Thus, for G, with $a_1, a_2, \ldots a_{99}$ being the number of fractions of that particular nominator, its CC is defined as:

$$CC = \frac{1}{100}\left(\frac{a_1}{1} + \frac{a_2}{3} + \frac{a_3}{6} + \ldots + \frac{a_{99}}{4851}\right) \tag{3.1}$$

Finding the lowest common denominator of these 99 fractions is a trivial exercise: it is $\frac{1}{3.49 \cdot 10^{40}}$. The CC of each vertex in G has such a lowest common denominator. This means, that the difference between any two CC values is at least $\frac{1}{100} \cdot \frac{1}{3.49 \cdot 10^{40}} = \frac{1}{3.49 \cdot 10^{42}}$. This value is the *smallest difference* between any two CC values.

Therefore the upper limit for possible, distinct CC values for G is the inverse of the smallest possible difference, i.e. $3.49 \cdot 10^{42}$. This does not mean that there are $3.49 \cdot 10^{42}$ actual graphs with a distinct CC value, but it does place an upper limit on the theoretical number of possible CC values. We denote this number as CC_{values}.

With the known upper and lower bounds of the CC, i.e. CC_{JEB} from Equation 2.15 and CC_{DM} from Equation B.8 in the appendix, the total number of possible CC values can be computed as follows:

Definition 3.1.1 *In total, there are CC_{values} distinct CC values for a graph. Any graph with a predefined number of vertices and edges has a subset CC_{values}. This subset is bounded by the CC of the JEB-graph from above, and the the CC of the DM-graph from below. The difference between these two values define the number of distinct CC values:*

$$CC_{actual} = CC_{values}(CC_{JEB} - CC_{DM})$$

Figure 3.1 shows how the number of distinct CC values changes for a growing number of edges in a graph G.

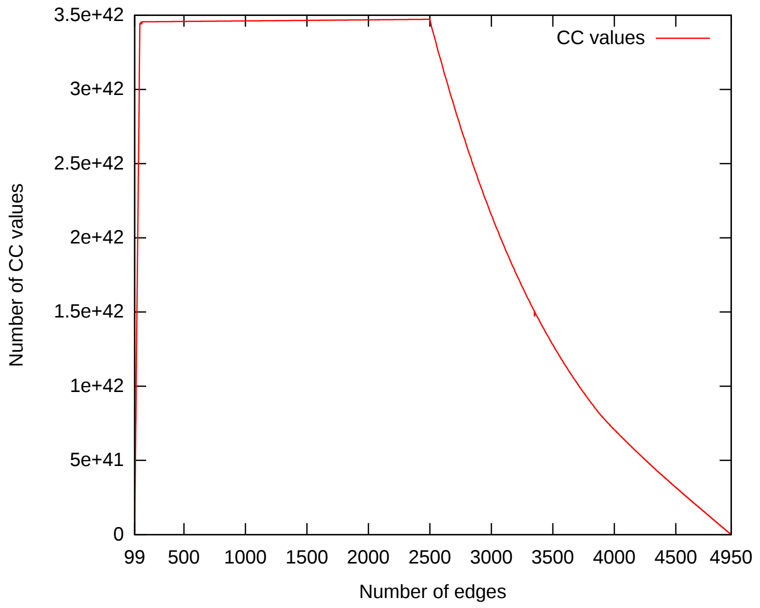


Figure 3.1: The total number of distinct CC values over the number of edges

3.1.2 Average Path Length

The number of distinct APL values can be calculated by the same principle as the number of distinct CC values. By definition, the APL of a graph G

is the sum of the APLs of the individual vertices to each other vertex. The length of a path between any two vertices ranges in $[1, v-1]$, where v is the number of vertices of the graph G.

As the smallest value for a path length between two vertices is 1, and because a single vertex is connected to all other vertices, the smallest APL of a single vertex is $1 \cdot (v-1)$. As the APL of the entire graph G is the sum of the APLs of all vertices, the difference between two APL values of a graph is at least $\frac{1}{v(v-1)}$. However, since the length of the path between any two vertices has the same value, the difference between two APL values is, at least, twice that value, i.e. $\frac{2}{v(v-1)}$.

Thus, for G, the smallest difference in APL is $\frac{2}{100\times 99} = \frac{1}{4950}$. This implies, that for G, between $APL = 2$ and $APL = 3$ there are at most 4950 different APL values. We call this number APL_{values}, and for G, $APL_{values} = 4950$. As with CC_{values}, this is a theoretical upper limit on the total number of distinct APL values.

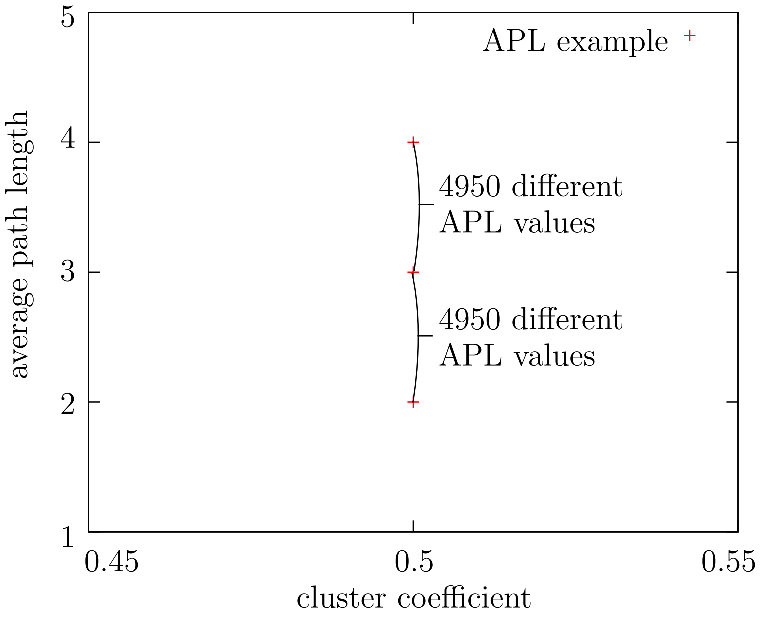


Figure 3.2: Example of the number of different APL values for graphs with an APL 2 and an APL 3.

Equation 2.14 defines the maximum APL as APL_{tail}, and Equation 2.2 defines the minimum APL as APL_{star}. We can calculate the total number of distinct APL values by determining the difference between APL_{tail} and APL_{star}, and multiplying that value with the number of APL_{values}.

Definition 3.1.2 *The total number of distinct APL values is bounded by the the APL of the Tail-graph, APL_{tail}, and the APL of the Star-graph, APL_{star}. The total number of distinct APL values is a superset of the number of APL_{values}, with the difference between APL_{tail} and APL_{star} defining the size of this superset:*

$$APL_{actual} = APL_{values}(APL_{tail} - APL_{star})$$

The total number of possible APL values for different number of edges is shown in Figure 3.3.

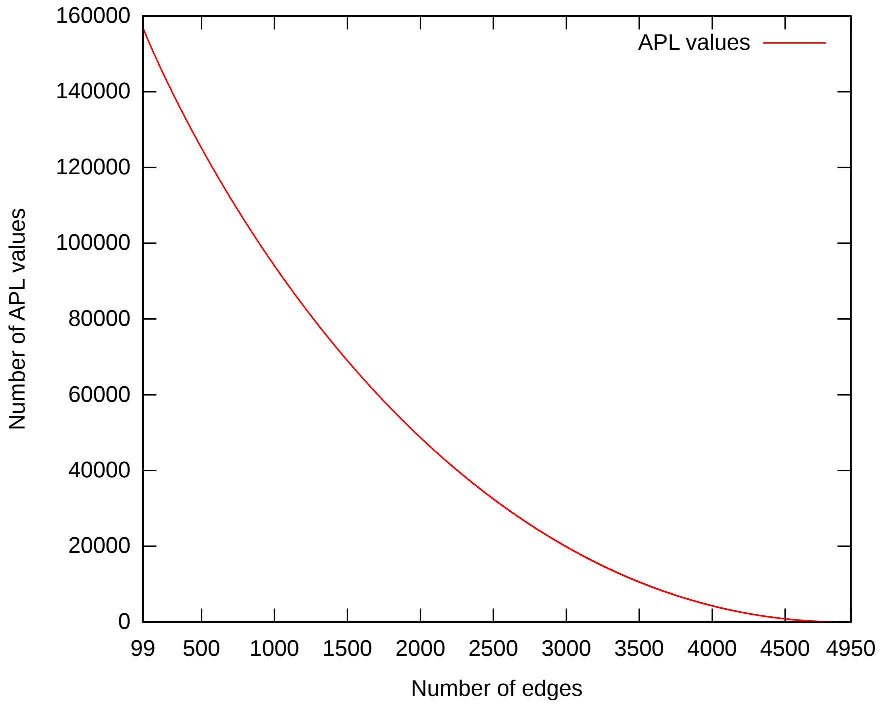


Figure 3.3: Number of distinct APL values for a growing number of edges

3.1.3 Total resolution

With the number of distinct CC values and distinct APL values known from Definition 3.1.1 and Definition 1.1.2, we can also calculate the total number of distinct states a graph G can have.

Definition 3.1.3 *The total number of distinct graph states of CC and APL values is*

$$CC_{actual} \cdot APL_{actual}$$

.

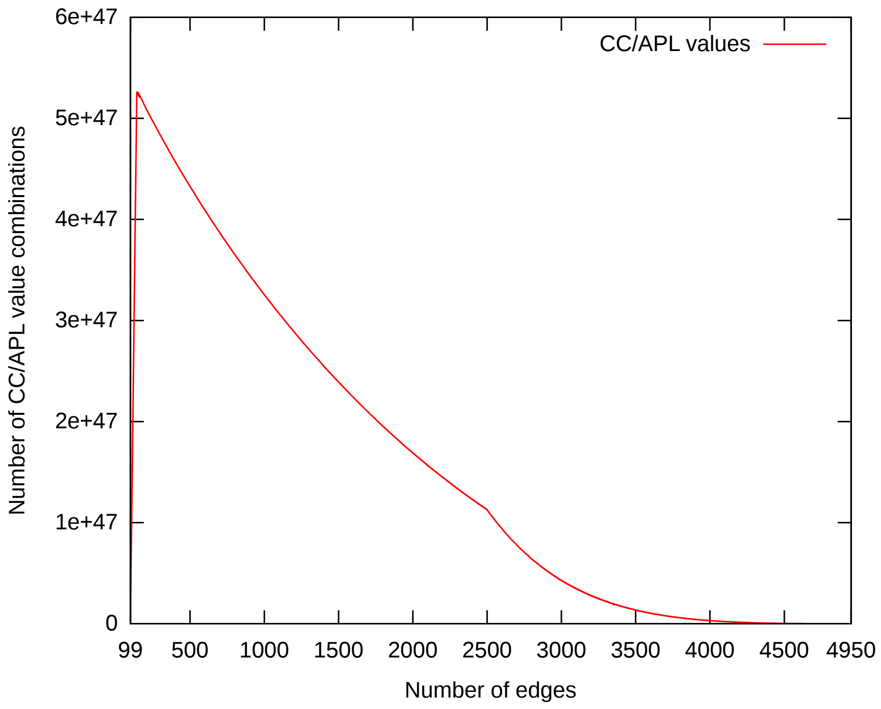


Figure 3.4: Number of CC and APL value combinations for a growing number of edges

Note that there are combinations of values of CC and APL for which there is no graph with the respective configuration. As described in Section 2.2.2, the maximum CC can only exist if the APL is equal to the minimum value. Thus, a graph with the maximum CC value and an APL value greater than the minimum APL value does not exist.

Based on the observation that, for some states the corresponding graph doesn't exist, we conclude:

1. One state can correspond to multiple graphs: isomorphic graphs. Isomorphic graphs have the same CC and APL values, even though the vertices might have different labels.

2. There is also a superclass of isomorphic graphs called *isostatic* graphs. These graphs have the same CC and APL values, but are not isomorphic. An example of two isostatic graphs G_1 and G_2 is given in Table 3.1. The example shows two JEB-graphs of 100 vertices and 300 edges. Note that both graphs also have a central vertex, with edges to all 99 other vertices.

	G_1		G_2		
Cliques and vertices per clique	Edges per clique	Total edges	Cliques and vertices per clique	Edges per clique	Total edges
2 cliques of 6 vertices	15	30	1 clique of 14 vertices	91	91
17 cliques of 5 vertices	10	170	5 cliques of 5 vertices	10	50
1 clique of 2 vertices	1	1	20 cliques of 3 vertices	3	60

Table 3.1: The number of configurations for two isostatic JEB graphs of 100 vertices and 300 edges.

We cannot calculate the total number of possible graphs from the number of possible states. Although we can calculate the total number of possible combinations, in which a graph can be wired, this number would include disconnected graphs, and thus overestimate the number of connected graphs. The only measure to find the number of connected graphs, is presented in the paper from 1959 by Erdős and Rényi [14], where they calculated the probability that a graph is connected, when its edges are chosen randomly. However, proving that this formula can also be applied to all possible graphs lies outside of the scope of this thesis.

Chapter 4

Crawler

To search for graphs in the statespaces, a *crawler* was developed. This crawler was deployed through a custom *framework*, which scheduled *jobs* on the DAS-3 [15] and DAS-4 [16] cluster sites. Consicely, each of these jobs had a crawler instance which computed graphs with a specific CC or APL value.

4.1 Basic crawler

The crawler is a "hill climber", searching for a specific range of values from either CC or APL. Actually, the crawler consists of three separate algorithms, that are executed sequencially:

- Dot-algorithm: searches for a specific new dot, or graph with a certain state, in the statespace.
- Line-algorithm: searches for a set of dots in the statespace by using the Dot-algorithm, along a certain axis. We will refer to these sets as lines.
- State-algorithm: Uses the Line-algorithm to create a sequence of lines to construct a statespace.

The crawler saves both the obtained CC and APL values and the mutations needed to achieve the new graph, leading to the capability to retrace all steps and redraw all intermediate graphs.

4.1.1 Dot-algorithm

The Dot-algorithm takes a graph as an input, and performs a *mutation* on the graph by randomly rewiring a single edge. In particular, the Dot-algorithm executes the following steps:

1. A graph is provided as input to the Dot-algorithm, see Figure 4.1a.
2. A random edge is removed from the graph, Figure 4.1b.
3. A random edge is added to the graph, see Figure 4.1c.
4. If the resulting graph is connected, its CC and APL are calculated, see Figure 4.1d.
5. In the current implementation of the algorithm, 100 mutations of the input graph are generated.
6. All generated graphs, that are connected, are then judged for the searched criteria, i.e. either a specific CC or a specific APL value.

For example, a random graph is provided to the Dot-algorithm. 100 mutations are made of this graph, each of which is a random rewiring of a single edge. Among the resulting 100 mutations, the connected graph with the highest CC is selected. It will be a new dot in the statespace, as well as a new input graph for the next execution of the Dot-algorithm.

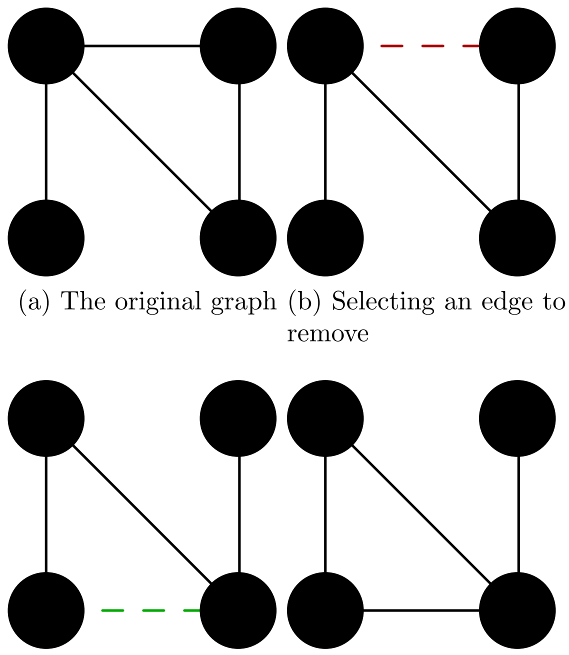
As mentioned before, the Dot-algorithm can also be used with another search criteria, e.g. the specified APL value, (instead of the highest or lowest APL). Such a search criteria is used by the Line-algorithm.

4.1.2 Line-algorithm

With the Line-algorithm a set of dots is searched, by executing the Dot-algorithm 1000 times. The algorithm uses the output from one execution of the Dot-algorithm as the input for the next execution of the Dot-algorithm. By doing this, a set of dots are placed in the statespace, and we call this set of dots a line.

Concisely, the Line-algorithm executes the following steps:

1. The Line-algorithm uses an input graph to search for graphs with a specified CC value, within a deviation of 0.010.
2. The Dot-algorithm is executed until a graph with the specified CC value has been found.



(a) The original graph (b) Selecting an edge to remove

(c) Selecting a new edge to add (d) The resulting graph

Figure 4.1: The Dot-algorithm

3. The search criteria is then changed, to find the graphs with the highest APL. Any discovered graph still has to remain within the deviation of the specified CC value. Graphs discovered in such a way form a line of dots with the same specific value of CC.

4. The Line-algorithm uses the Dot-algorithm for a total of 1000 times to search for the graphs within the deviation of the given CC value.

5. The Line-algorithm produces an output graph as result. This should have the specified CC value, within a deviation of 0.010, and the highest APL. If no graph was found within the deviation, then the graph that has the nearest CC value is used as output graph.

Thus, the algorithm produces either the best result, that is, a graph with the highest APL within the deviation value, or the graph that is closest to the desired CC value.

The following example illustrates the execution procedure of the Line-algorithm for graphs of 300 edges.

1. The JEB-graph is provided as input to the Line-algorithm.

2. Using the Dot-algorithm, a new graph is discovered that is as close as possible to the desired CC value of 0.800. This new graph has a CC of 0.9650.

3. The Dot-algorithm is repeatedly executed until the resulting graph has a CC of $0.8000 \pm 0,010$. In the example, this happens 15 times, the 16^{th} graph has a CC of 0.8039.

4. The Line-algorithm then searches for graphs with the highest APL. Such graphs must have a CC of $0.8000 \pm 0,010$. This set of dots forms a line in the statespace.

5. Upon 984 iterations of the Dot-algorithm, the algorithm discovers a graph with a CC 0.7905 and an APL of 5.1226. This is the output graph, within the given deviation, and the highest APL.

The result of the Line-algorithm is presented in Figure 4.2, the line consists of 1000 dots.

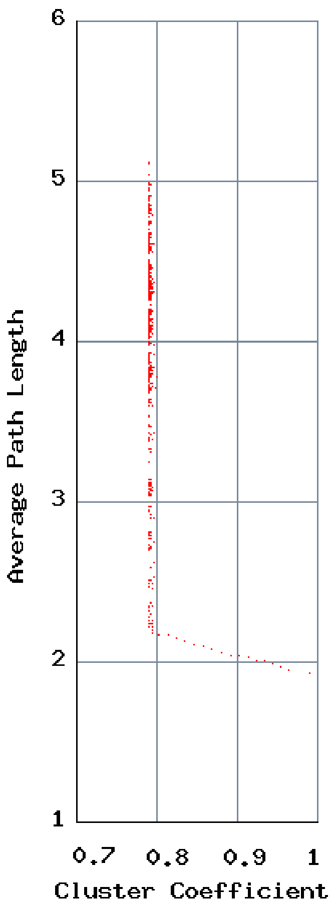


Figure 4.2: Result of the Line-algorithm, searching for a CC value of 0.800.

4.1.3 State-algorithm

The final part of the crawler searches for graphs in specified parts of the statespace. It does this by dividing the statespace in CC value intervals, and using the Line-algorithm four times in each interval. Each interval has a size of 0.001, and these intervals start from the lower bound of the CC up to the upper bound. For example, $0.800, 0.799, 0.801$, and so on. The State-algorithm runs in each interval, and gets an input graph from the user, and the desired CC value to search graphs for. This desired value is the same as the interval value, e.g. the State-algorithm will search for graphs with a CC value 0.800 in the interval 0.800.

The State-algorithm executes in the following steps:

1. The State-algorithm gets an input graph, and the CC interval for which graphs with that CC value must be found.

2. The State-algorithm, with the input graph and the CC value, executes the Line-algorithm.

3. The Line-algorithm searches for graphs with the CC value, by executing the Dot-algorithm 1000 times. If such a graph is discovered, within the error-margin of 0.01, the Line-algorithm will search for graphs with the CC value of the discovered graph and the highest APL.

4. Such an execution of the Line-algorithm is repeated four times, where every output of the previous execution is used as the input for the next execution. The third and fourth execution search for graphs with the CC value and the lowest APL.

Thus, the State-algorithm executes the Line-algorithm four times, resulting in 4000 dots. Each interval therefore consists of 4000 dots. The desired CC value is now increased by the size of the interval, 0.001, and the State-algorithm again uses the Line-algorithm to find graphs in that interval. This continues until the state-algorithm reaches the upper bound of the CC value, rounded up.

To illustrate this part of the crawler as well, an example of the State-algorithm will now follow, searching in the intervals with the CC values of 0.800 and 0.801, with the JEB-graph as the input graph. In the example, the State-algorithm searches for graphs with 100 vertices and 300 edges.

1. The State-algorithm starts searching for graphs with a CC of $0.800 \pm 0,01$, using the JEB-graph as input-graph.

2. The State-algorithm provides the JEB-graph as input to the Line-algorithm.

3. The 16th graph found by the Line-algorithm has a CC of 0.8039. The line-algorithm then searches for graphs with the highest APL, while keeping the CC at $0.800 \pm 0,01$.

4. After finding 1000 dots, the Line-algorithm produces a graph with a CC of 0.7905 and an APL of 5.1226. Figure 4.3a illustrates this.

5. This graph is provided, by the State-algorithm, as input for the second run of the Line-algorithm, which again looks for graphs with a CC 0.8000 ± 0.01, and the highest APL.

6. Once 1000 dots are found, the Line-algorithm produces a graph with a CC of 0.7905, and an APL of 5.8378. Figure 4.3b illustrates this. The newly discovered dots are in blue.

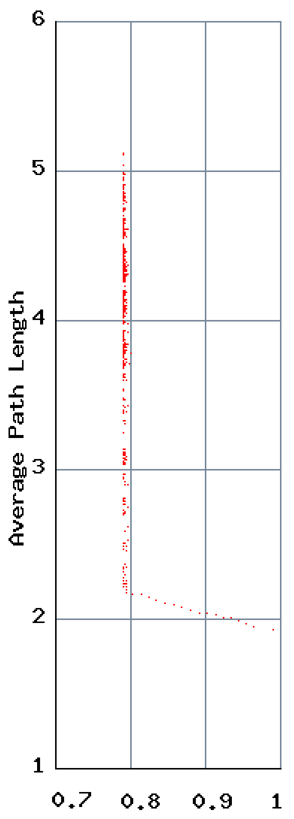
7. This graph is provided as input for the third execution of the Line-algorithm. Again, the algorithm searches for graphs with a CC of 0.8000 ± 0.01, but this time it searches for graphs with the lowest APL.

8. Upon the discovery of another 1000 dots, a graph with a CC of 0.7902 and an APL of 1.9394 was found by the Line-algorithm. Figure 4.3c illustrates this. The newly discovered dots are in blue.

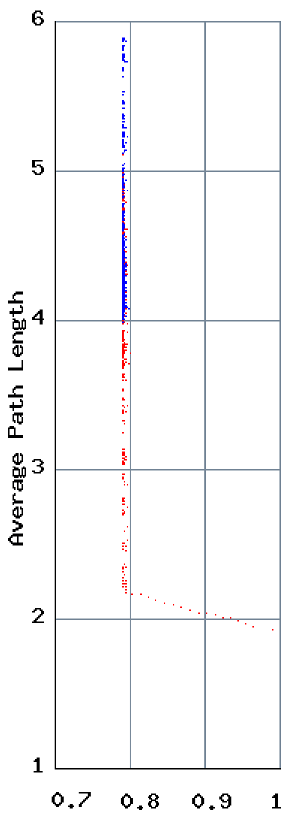
9. With this graph as input, the Line-algorithm is executed for the fourth and final time, to search for graphs with a CC of 0.8000 ± 0.01 and the lowest APL. Figure 4.3d illustrates this. The newly found dots are in blue.

10. For the CC interval of 0.800, the State-algorithm executed the Line-algorithm a total of four times. The State-algorithm then starts searching for graphs in the CC interval of $0.801 \pm 0,01$, reverting back to step 1. Figure 4.3e illustrates this. The 4000 new dots of this interval are in blue.

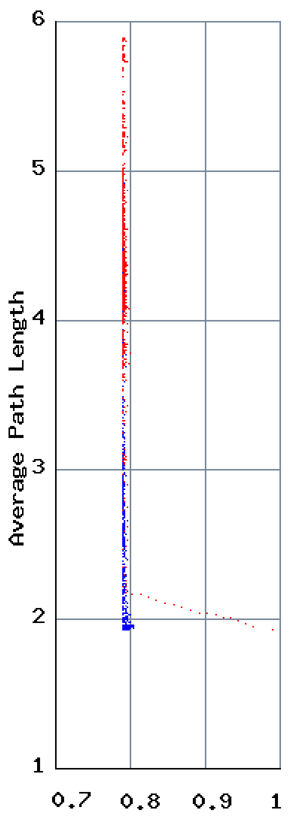
It is important to note that the State-algorithm can also be used to search for specific APL values, and then minimizing and maximizing the CC values.



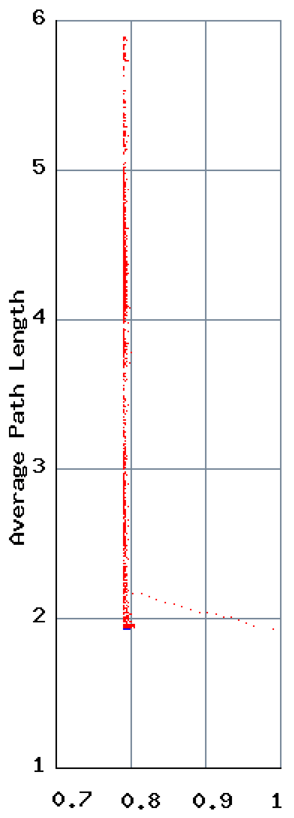
(a) First line-algorithm



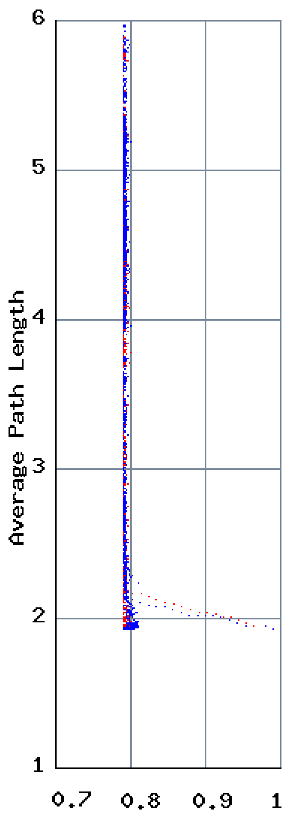
(b) Second line-algorithm



(c) Third line-algorithm



(d) Fourth line-algorithm



(e) Next interval

Figure 4.3: Example showing 2 iterations of the State-algorithm.

4.2 Grid framework

In order to use the crawler on the DAS-3 and DAS-4 sites, a framework has been developed. It consists of: 1) a master server keeping track of the calculation jobs, which are provided to the crawler instances, and 2) client nodes, deployed on the DAS sites, to start the crawler instances and report the job progress to the master server. We will first describe the jobs.

4.2.1 Jobs

A job is the information that the crawler needs to execute the state-algorithm. In particular, a job consists of: 1) an ID for communication and bookkeeping purposes, 2) the number of edges of the graphs to search for, and 3) the initial graph that is used by the Line-algorithm. It specifies if the crawler should look for graphs with a certain CC or APL value. Moreover, the input also includes the range of values for the search, for example:

```
id=442187
edges=100
type=cpl
typestart=jeb
start=8.790
finish=8.790
```

4.2.2 Nodes

The crawler has been deployed on the DAS-3 and DAS-4 sites, grid computer sites belonging to various academic institutions in the Netherlands. The DAS, Distributed ASCI Supercomputer, sites are meant for research purposes, and are now on their fourth generation, DAS-4, while DAS-3 remains in (limited) operation.

The DAS-3 sites [15], launched in late 2006, are located at 5 Dutch institutions:

- Vrije Universiteit Amsterdam: 85 dual-CPU dual-core 2.4 Ghz AMD Opteron DP 280 compute nodes. 10 nodes have been used for this master project, for 40 crawler instances.
- Leiden University: 32 dual-CPU 2.6 Ghz AMD Opteron DP 252 compute nodes. 10 nodes have been used, for 20 crawler instances.

- University of Amsterdam: 41 dual-CPU dual-core 2.2 Ghz AMD Opteron DP 275 compute nodes. 10 nodes have been used, for 40 crawler instances.
- Delft University of Technology: 68 dual-CPU 2.4 Ghz AMD Opteron DP 250 compute nodes. 10 nodes have been used, for 20 crawler instances.
- The MultimediaN Consortium: 46 dual-CPU 2.4 Ghz AMD Opteron DP 250 compute nodes. 10 nodes have been used, for 20 crawler instances.

Overall, the DAS-3 sites contributed 140 crawler instances to the project. Each crawler could run for 128 hours per week, for a total of 17,920 compute hours per week for all crawlers. The DAS-3 crawlers ran for a total of 18 weeks.

The DAS-4 sites [16], launched in late 2010, are located 6 Dutch academic institutions:

- Vrije Universiteit Amsterdam: 74 dual-CPU quad-core 2.4 Ghz compute nodes. 15 nodes have been used for this master project, for a total of 120 crawler instances.
- Leiden University: 16 dual-CPU quad-core 2.4 Ghz compute nodes. 5 nodes have been used, for 40 crawler instances.
- University of Amsterdam: 16 dual-CPU quad-core 2.4 Ghz compute nodes. 10 nodes have been used, for 80 crawler instances.
- Delft University of Technology: 32 dual-CPU quad-core 2.4 Ghz compute nodes. 10 nodes have been used, for 80 crawler instances.
- The MultimediaN Consortium: 36 dual-CPU quad-core 2.4 Ghz compute nodes. 10 nodes have been used, for 80 crawler instances.
- ASTRON: 24 dual-CPU quad-core 2.4 Ghz compute nodes. 10 nodes have been used, for 80 crawler instances.

The DAS-4 sites contributed 480 crawler instances to the project. Each crawler could run for 108 hours per week, for a total of 51,840 compute hours per week for all crawlers. The DAS-4 crawlers ran for a total of 32 weeks.

In total, 620 crawler instances have calculated approximately 2,085,120 hours in total.

4.2.3 Server side

The master server, using MySQL to store the job data, keeps track of the jobs that need to be calculated, and assigns them to crawler instances which contact it. All crawler instances are started in a *reservation* on one of the DAS clusters, and get the requested number of nodes. To start the reservation, the framework will contact the server and request a new time-slot to reserve on the DAS cluster. After the reservation is granted at the DAS-site, all instances will be started and all instances will contact the master-server. For an instance to get a job, it has to contact the master server, which (using a white-list) checks if the domain of the instance is allowed to request a job. During the reservation, instances contact the master server to request a new job, update the progress of the current job, and announce the end of a job. Jobs can be assigned to a designated DAS cluster, or be assigned on a first come, first serve basis.

Since the job data is kept in a MySQL database, a job will take a minimum of 4 MySQL queries: checking the white-list, starting, updating and finalizing a job.

4.2.4 Client side

An instance, running at one of the DAS nodes, proceeds to request a new job, after finishing its currently assigned job. All instances have a designated directory in which they store the results, and keep a copy of the crawler executable. Due to the reservation policy on the DAS cluster and, thus, limited execution time, all execution results (including intermediate ones) are stored in files. In this way, only results of executions upon the expiration of a reserved time on DAS cluster are lost.

All instances are started consecutively, with 10 second intervals, to prevent the master-server of being overloaded with job-requests. The first started instance, graph0, collects all results and sends them by *scp* to the *kits* student server at the VU. It does so after completing its assigned job, at most every 15 minutes, to ensure the that the *kits* server is not flooded with SSH requests.

4.2.5 Flaws

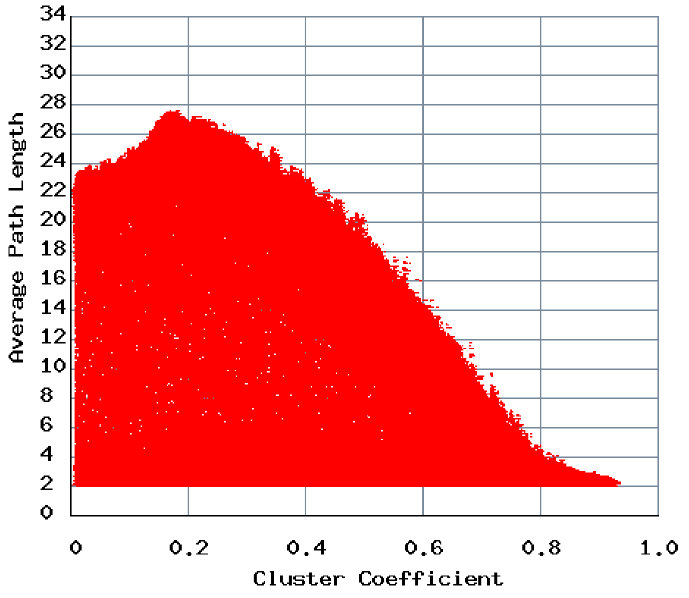
Two major flaws associated with the framework have been identified and corrected during the project. First, all instances would copy their results back to the *kits* server by *scp*, prior to termination. This lead to a flood of SSH requests, causing the SSH daemon at the *kits* server to become irresponsive, thereby preventing other users from logging in. This issue has been fixed by ensuring that only a single instance at a site is allowed to copy results back.

Second, if the duration of the jobs was small enough, the master server can potentially be swamped by the number of connection requests, and by queries from its MySQL daemon. This is practically a DDoS attack by 620 instances on a single master server. If the master server is down, no jobs can be assigned, leading to crawler instances stopping their work, wasting the time of the DAS-reservation. This flaw has been fixed by identifying small jobs[1] and distributing them to a limited number of sites.

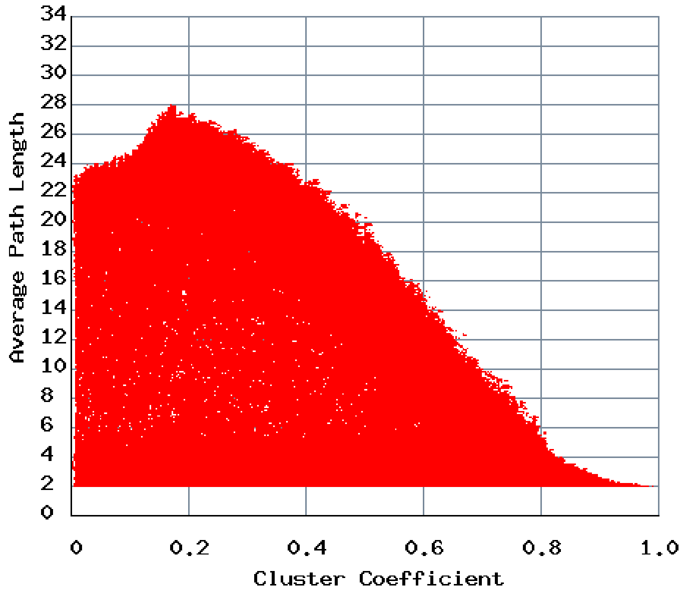
4.3 Scatterplot generation

In order to produce the scatterplots in the next section, all statespaces for a particular number of edges were calculated a total of six times. Three of those were done with the Line-algorithm looking for CC values (Figures 4.4a, 4.4b and 4.4c), and the other three with the Line-algorithm looking for APL values (Figures 4.4d, 4.4e and 4.4f). For both sets of scatterplots, there were three different starting points: (1) a randomly generated graph (Figures 4.4a and 4.4d), (2) the JEB-graph (Figures 4.4b and 4.4e) and (3) the Tail-graph (Figures 4.4c and 4.4f). An example of the combined result is shown in Figure 4.5.

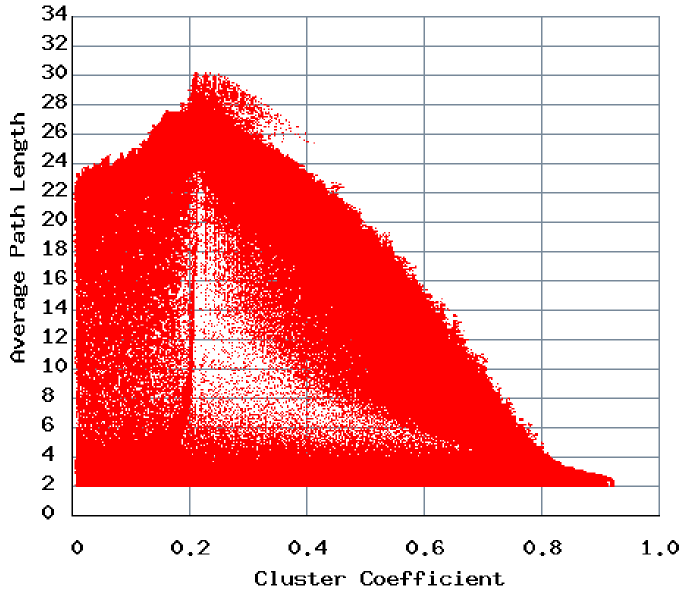
[1]Generally jobs with a high number of edges and looking for graphs with CC values



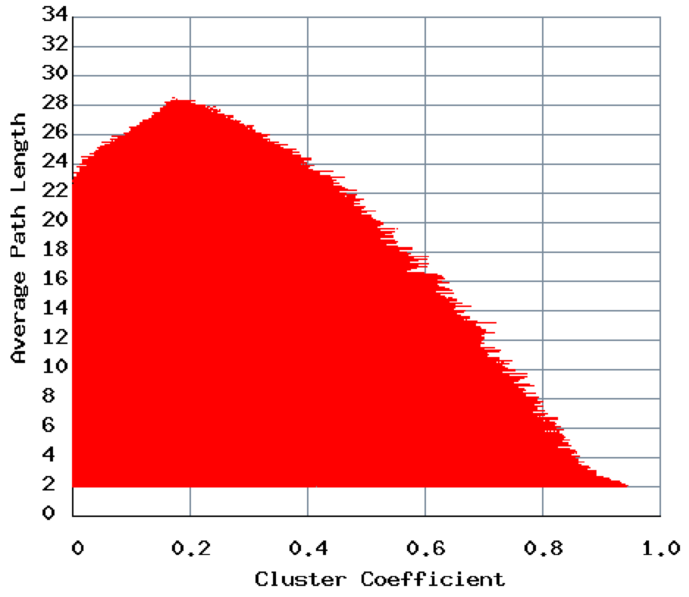
(a) CC starting with random



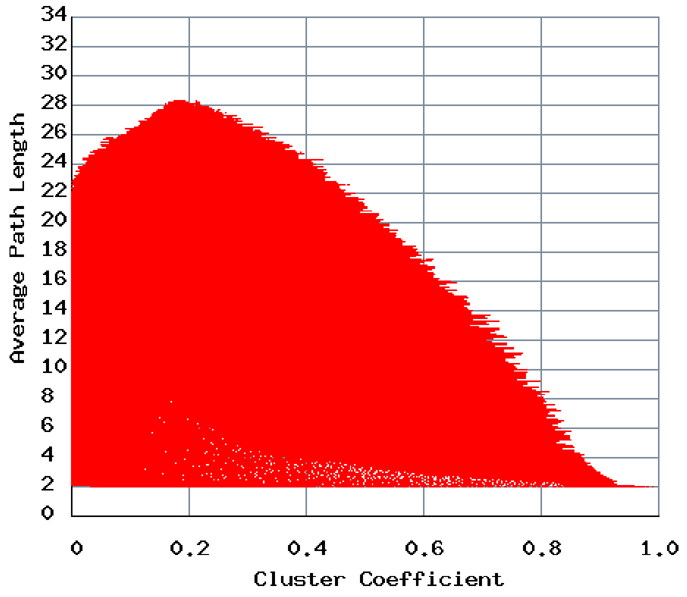
(b) CC starting with JEB



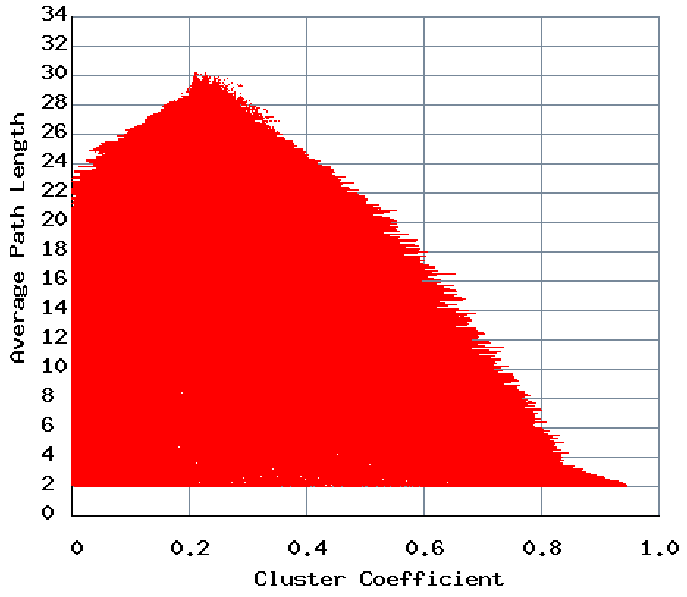
(c) CC starting with Tail



(d) APL starting with random



(e) APL starting with JEB



(f) APL starting with Tail

Figure 4.4: Examples of the 6 different generated scatterplots for the statespace of 300 edges

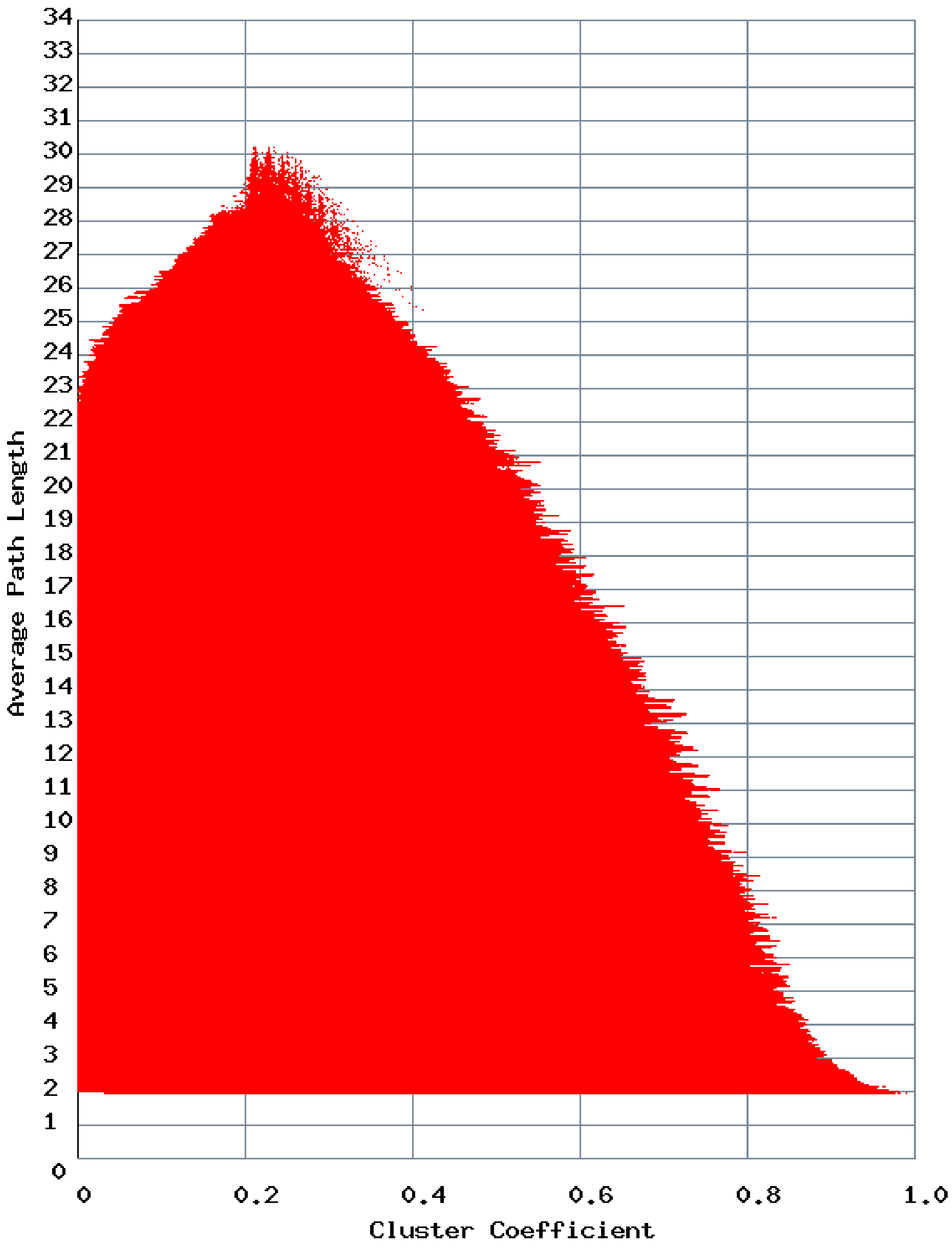


Figure 4.5: Combination of the 6 previous scatterplots, showing the statespace of a graph of 300 edges.

Chapter 5

Crawler results

The crawler deployed on the DAS-3 and DAS-4 cluster sites, searched for graphs with 100 vertices and a predefined number of edges: from the interval $[100, 4950]$ with a step of 50 edges. As mentioned in Section 4.3, the crawler searched for graphs with specific CC and APL values, from three different initial points: JEB-graphs, Tail-graphs and randomly generated graphs.

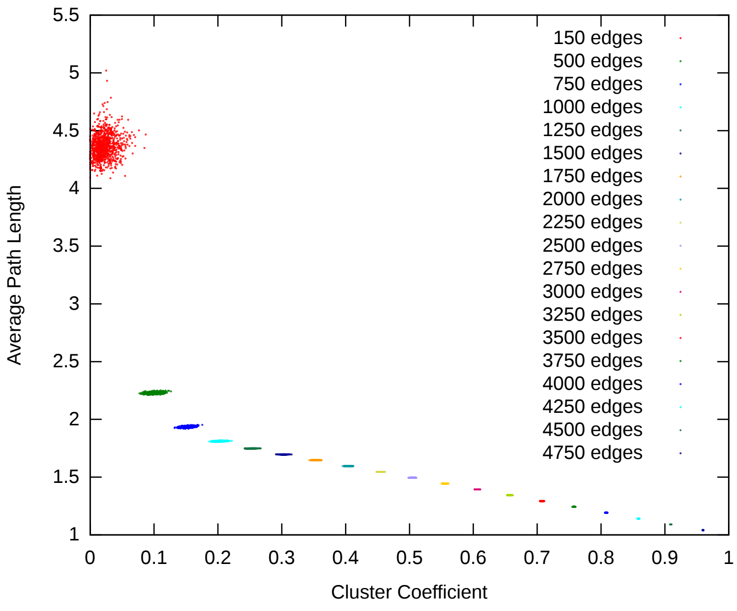


Figure 5.1: The scatterplot of the random graphs with different number of edges. Only connected graphs are selected from 10 million graphs, generated by the crawler.

5.1 Random graphs

One of the initial points of the crawler is a randomly generated graph. In order to visualize such a graph, we plotted ten sets, each with a different random seed, of 10 million random graphs for each of the statespaces searched for. We recorded the number of disconnected graphs, to compare it with the formula of Erdős and Rényi [14][1]. We show these results for $100, 500, 750, 1000, \ldots, 4750$ edges in Figure 5.1. The larger versions of these scatterplots for the edges $[150, 500]$ with a step of 50 edges, can be found in Appendix A.

Edges	Erdős and Rényi	Generated by crawler
100	0.00%	0%
150	0.69%	0.33%
200	16.02%	17.24%
250	50.98%	55.98%
300	78.05%	82.28%
350	91.28%	93.66%
400	96.70%	97.84%
450	98.77%	99.28%
500	99.55%	99.76%
550	99.83%	99.92%
600	99.94%	99.98%
650	99.98%	99.99%
700	99.99%	99.99%
750 and more	100.00%	100.00%

Table 5.1: Comparison between predicted values by Erdős and those found experimentally.

As Table 5.1 shows, there is a difference between the values predicted by Erdős and Rényi, and those found by generating ten sets of 10 million graphs, each set with a different random seed. We speculate that this is due to the formula of Erdős and Rényi being described as being for graphs with any number of vertices, and also for graphs with a number of vertices greater than 100. We further discuss this in Appendix C.

[1]These graphs were only generated, and did not use the Dot-algorithm.

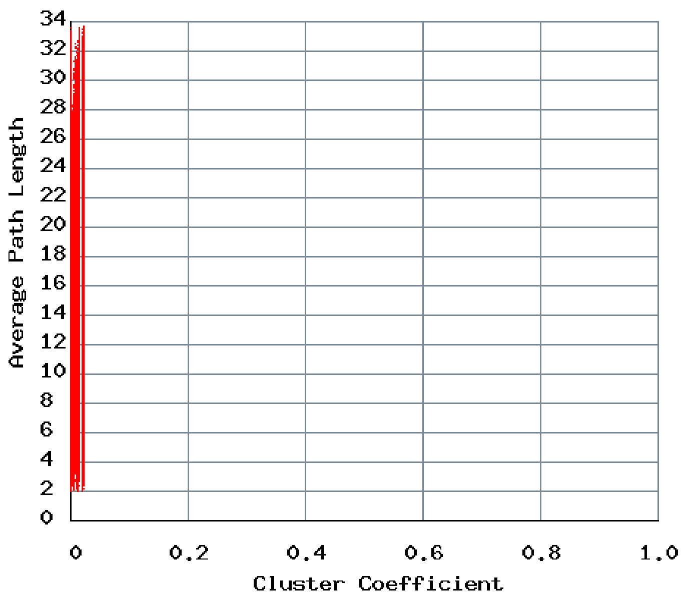
5.2 Results

In the remainder of this section, we will show the results obtained by the crawler. All generated graphs are plotted according to their CC and APL values. The combined scatterplots are shown, for the graphs of 100, 250, 500, 750 . . ., 4750 and 4950 edges. Resulting scatterplots can be found at `http://www.phoib.net/uni/graphs`

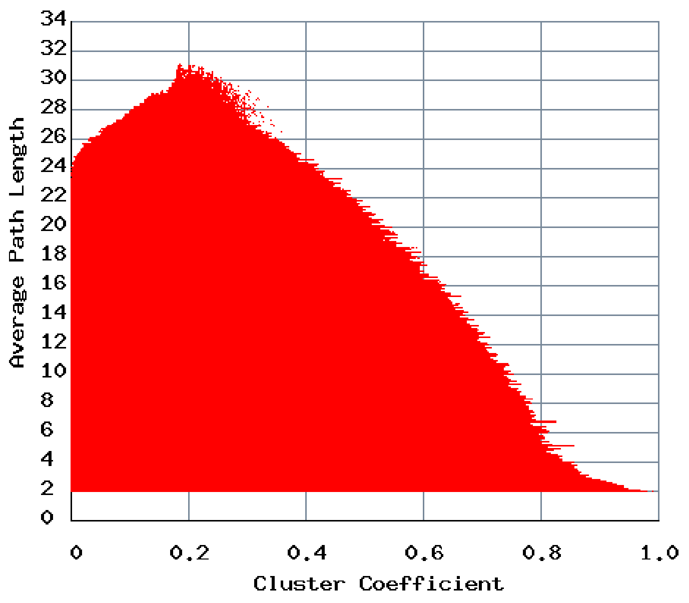
The crawler generated graphs, searching for graphs with specific CC values. The lower bound of these specific CC values was the minimal CC, CC_{DM}. The higher bound of these specific CC values was the maximal CC, CC_{JEB}. The lower and upper bound were seperated by CC intervals, each 0.001 wide. For each interval, the crawler searched for graphs. This was also done for the APL, with the lower bound of the specific APL values being the minimal APL, APL_{star}, and the higher bound the maximal APL, APL_{tail}. The lower and upper bound of the APL were seperated with APL intervals of 0.01 wide.

None of the generated graphs exceeded the bounds of the CC and APL, which were described in Chapter 2. The results in Figures 5.2b-5.2f show that the shape of the statespace shifts to the right, in the same way as the CC of the Tail-graph shifts to the right when more edges are added to it. The results also show (e.g. in Figure 5.3d) that graphs with an APL higher than a randomly generated graph and a CC 0 become less likely when more edges are added to the graph. Also, Figure 5.3e show, that while graphs of 2500 edges with a CC 0 are theoretically possible, but were not found by the crawler..

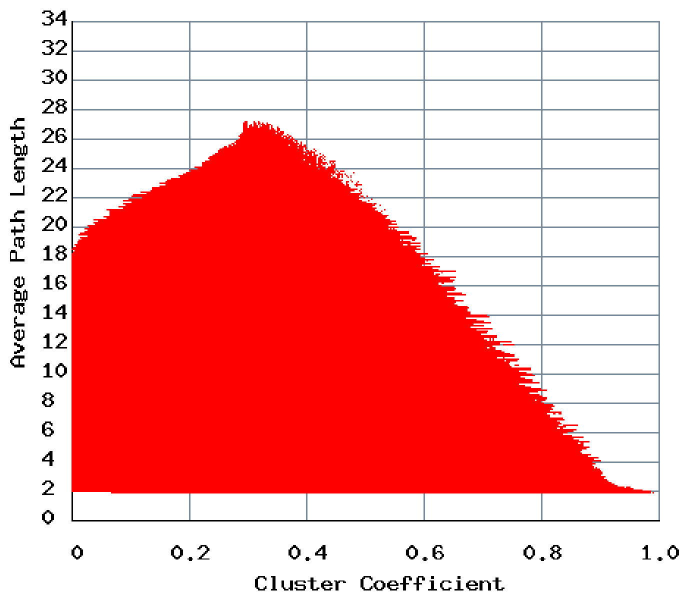
The same applies for the JEB-graphs. They were never found by the crawler, when graphs were generated with an input graph which was not a JEB-graph itself. Tail-graphs were also never found, when graphs were generated with an input graph, other than a Tail-graph. This gives an insight into the distribution of graphs within the statespace: that such specialized graphs are very rare, compared to the generated graphs.



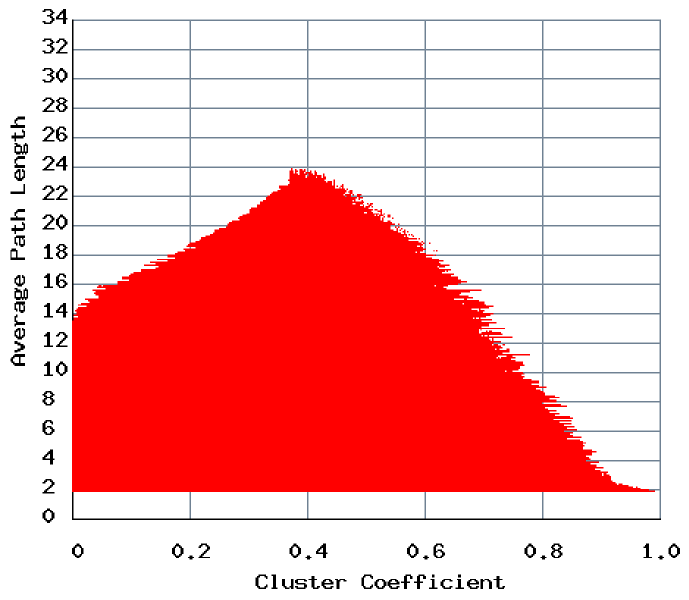
(a) 100 edges



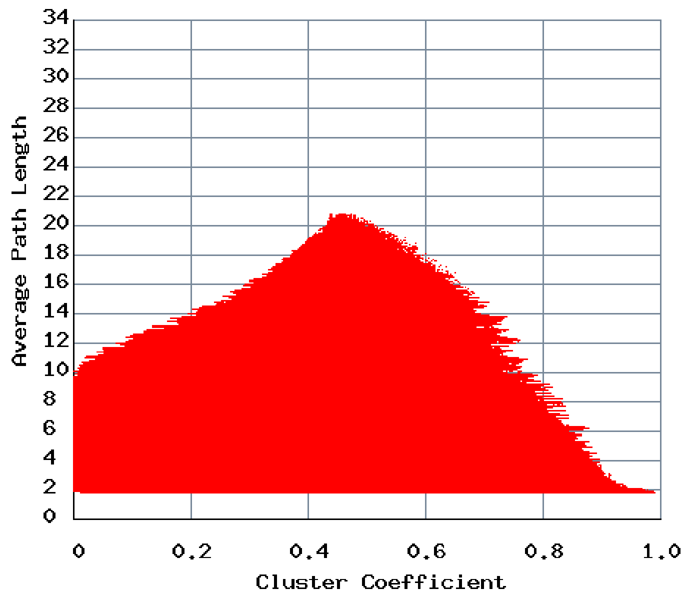
(b) 250 edges



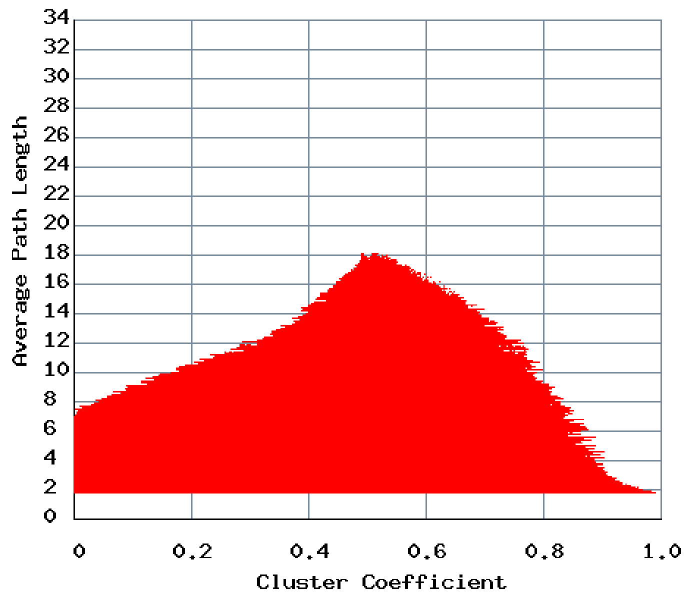
(c) 500 edges



(d) 750 edges

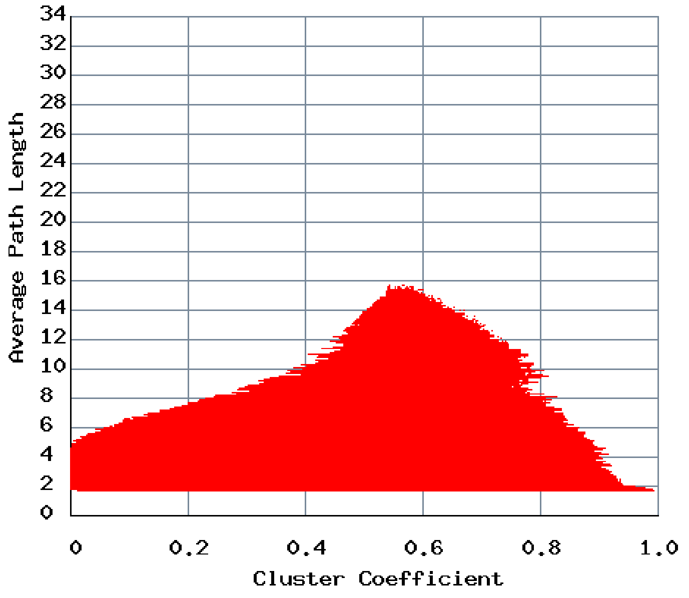


(e) 1000 edges

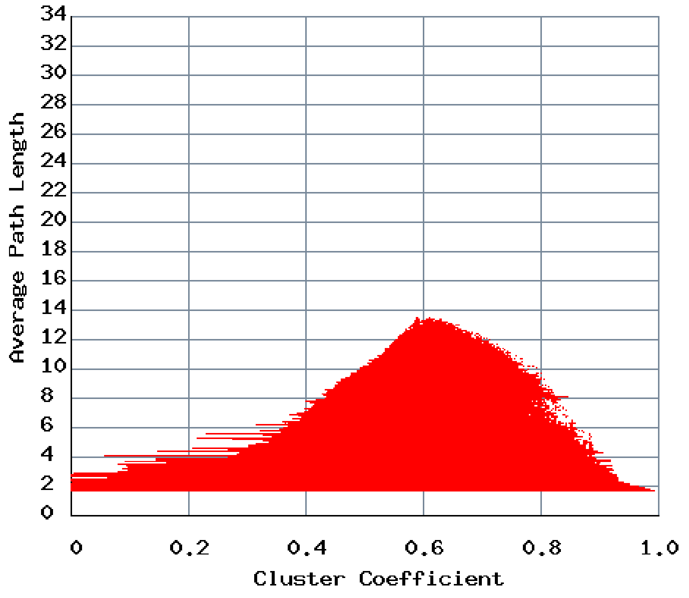


(f) 1250 edges

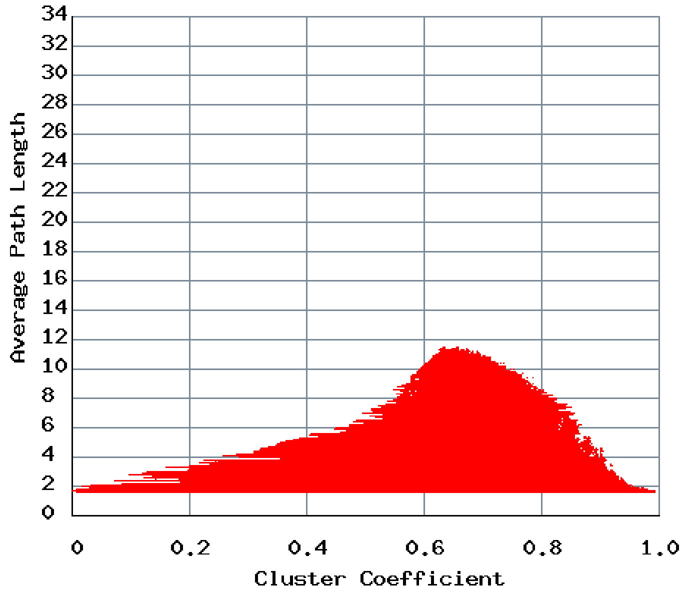
Figure 5.2: Evolution of the statespaces between 100 and 1250 edges



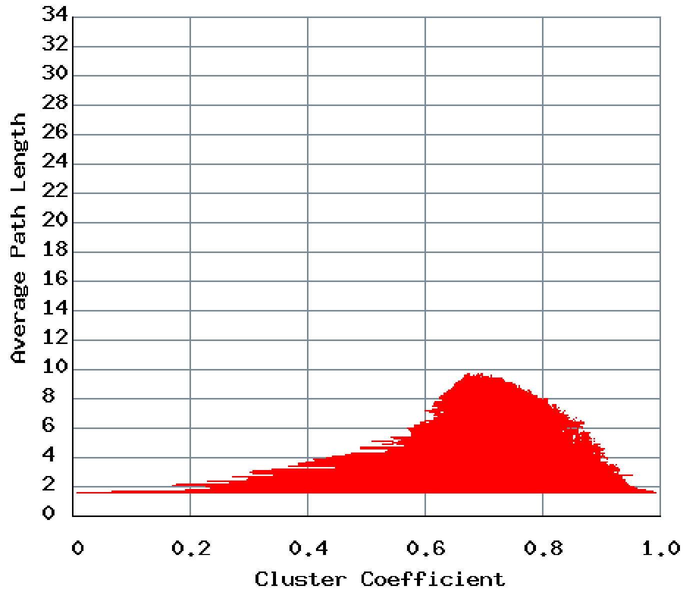
(a) 1500 edges



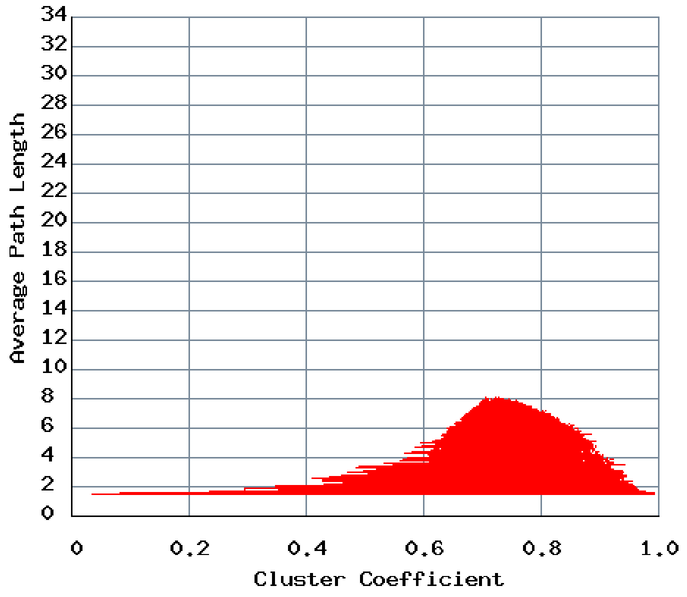
(b) 1750 edges



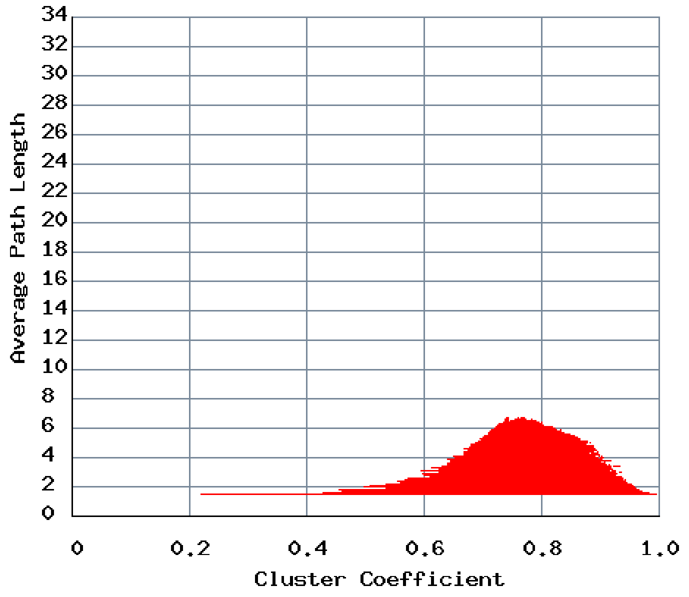
(c) 2000 edges



(d) 2250 edges

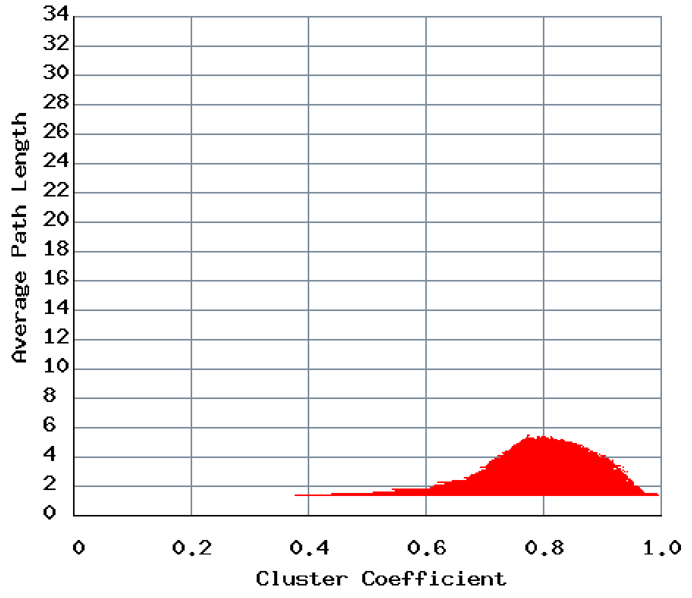


(e) 2500 edges

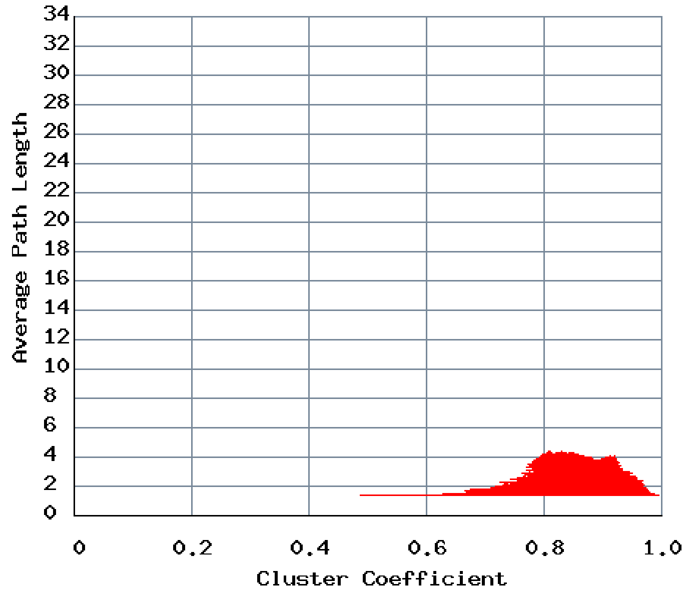


(f) 2750 edges

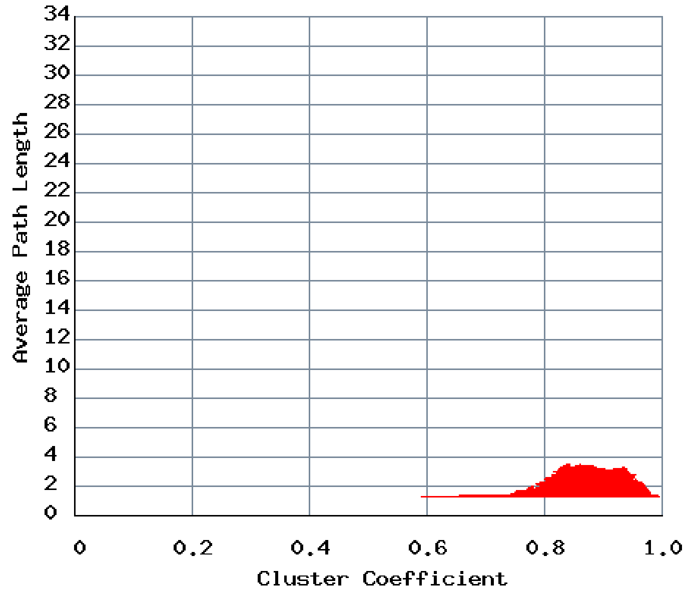
Figure 5.3: Evolution of the statespaces between 1500 and 2750 edges



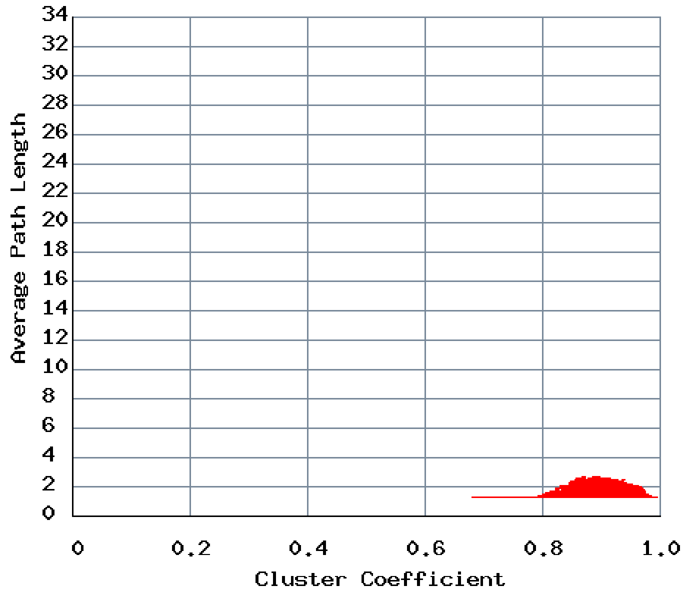
(a) 3000 edges



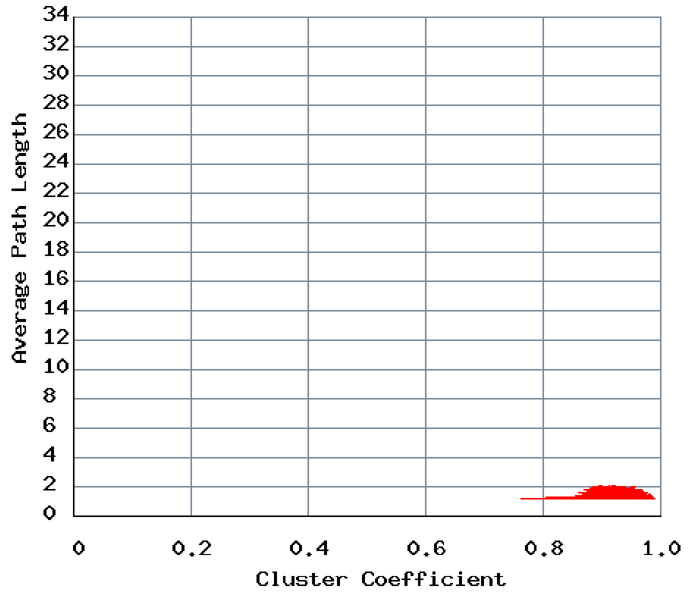
(b) 3250 edges



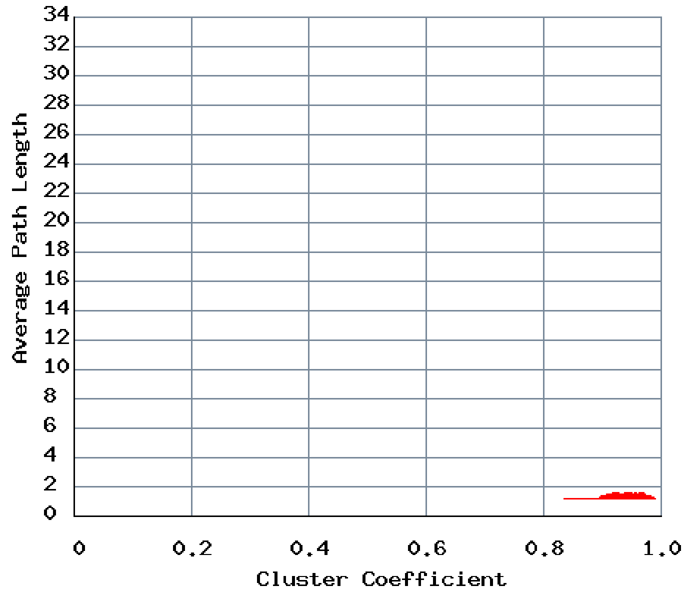
(c) 3500 edges



(d) 3750 edges

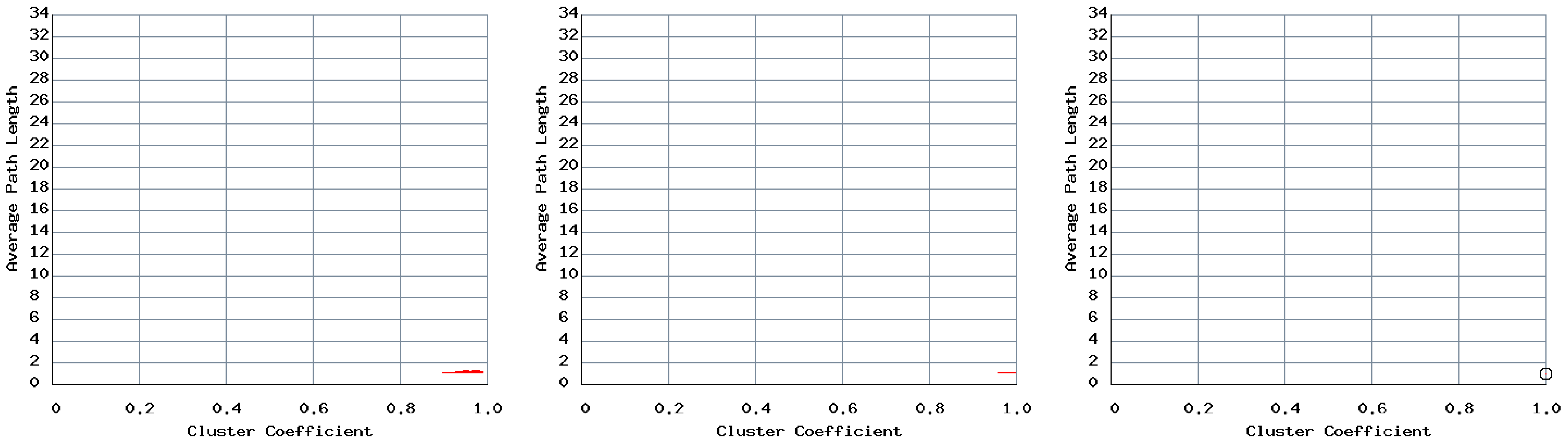


(e) 4000 edges



(f) 4250 edges

Figure 5.4: Evolution of the statespaces between 3000 and 4250 edges



(a) 4500 edges

(b) 4750 edges

(c) 4950 edges. Note a single point inside the circle.

Figure 5.5: Evolution of the statespaces between 4500 and 4950 edges

Chapter 6

Conclusion

In this thesis, we have explored the space of simple, connected graphs of 100 vertices. We expressed a statespace of these graphs as the combination of their CCs and APLs. Three different research questions were addressed: (1) How does a statespace develop under a growing number of edges, (2) can the lower and upper bounds of the statespace be identified, and (3) can the number of states for graphs with a certain number of edges be described? We can now answer these three questions.

Development of the statespace: This thesis produced an insight into the development of the statespace of a graph of 100 vertices. Even with a step of 50 edges, one can observe how the average shape of the statespace develops, gradually shifting to the right under a growing number of edges. This is in line with the randomly generated graphs, which CC also shifts towards the right under a growing number of edges.

We can also expect the generated graphs to be a random sample by the way the crawler generates its graphs. It performs a random walk through the statespace, and even though the direction of the walk is dictated by the crawler, its results still reflect the nature of a random walk. Combined with the fact that neither a JEB or a Tail-graph was found by the crawler, this would imply that there are very few, random paths through the statespace to either a JEB or a Tail-graph.

Bounds of the statespace: We have conclusively shown the upper and lower bounds of the statespace, and formulated the conditions under which these bounds can be achieved. Both CC and APL have distinct upper and lower bounds, in particular, the DM-graph and JEB graph for the CC, and the Star-graph and Tail-graph for the APL.

We have shown that the bounds for the APL as well as the upper bound for the CC converge to a single point; though the convergence is more gradual in case of the APL than of the CC. Except the upper bound of the CC, all three bounds grow continuously, by adding single edges.

Size of the statespace: In general, the total number of simple, connected graphs with 100 vertices in the entire statespace remains too large to be calculated. We have shown the number of possible combinations of the CC and APL, but we cannot calculate how many graphs each state has.

Future work: The graphs obtained from the crawler need to be studied with respect to their degree distribution. This will introduce a new dimension in which graphs can vary.

Furthermore, a new crawler has to be developed, which can not only search for graphs with a specific value, but also quantify how many graphs there are, with such a state. Such a crawler will give an insight into how graphs are distributed inside a statespace.

Finally, theoretical research is needed into the number of graphs. Using the formula from Erdős and Rényi, combined with Cayley's formula[17] on the number of trees, it should be able to formulate the number of graphs for a predefined number of vertices and edges.

Bibliography

[1] M. E. J. Newman, "The structure and function of complex networks," *SIAM REVIEW*, vol. 45, pp. 167–256, 2003.

[2] J. Saramäki and K. Kaski, "Scale-free networks generated by random walkers," *Physica A: Statistical Mechanics and its Applications*, vol. 341, pp. 80–86, Oct. 2004.

[3] L. da Costa, F. Rodrigues, G. Travieso, and P. Boas, "Characterization of complex networks: A survey of measurements," *Advances in Physics*, vol. 56, pp. 167–242, 2007.

[4] W. M. Tam, F. C. M. Lau, and C. K. Tse, "Construction of scale-free networks with adjustable clustering," 2008.

[5] N. Takahashi, "On clustering coefficients of graphs with the fixed numbers of vertices and edges," in *Proc. European Conference on Circuit Theory and Design*, pp. 814–817, 2009.

[6] R. Southwell and J. Huang, "Complex networks from simple rewrite systems," *CoRR*, vol. abs/1205.0596, 2012.

[7] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity," *J. Graph Algorithms Appl.*, vol. 9, no. 2, pp. 265–275, 2005.

[8] E. Coupechoux and M. Lelarge, "Impact of clustering on diffusions and contagions in random networks," in *Proc. Conf. on NETwork Games, COntrol and OPtimization (NetGCooP)*, pp. 1–7, IEEE, 2011.

[9] K. Soramäki, M. L. Bech, J. Arnold, R. J. Glass, and W. E. Beyeler, "The topology of interbank payment flows," *Physica A: Statistical Mechanics and its Applications*, vol. 379, pp. 317–333, June 2007.

[10] Z. Dong, W. Wu, X. Ma, K. Xie, and F. Jin, "Mining the structure and evolution of the airport network of china over the past twenty years," in *Proc. Conf. on Advanced Data Mining and Applications (ADMA)*, vol. 5678 of *LNCS*, (Berlin, Heidelberg), pp. 104–115, Springer-Verlag, 2009.

[11] J. Badham and R. Stocker, "A spatial approach to network generation for three properties: Degree distribution, clustering coefficient and degree assortativity," *Journal of Artificial Societies and Social Simulation*, vol. 13, no. 1, p. 11, 2010.

[12] "NetworkX Python package." `http://networkx.lanl.gov/`.

[13] W. S. Lovejoy and C. H. Loch, "Minimal and maximal characteristic path lengths in connected sociomatrices," *Social Networks*, vol. 25, no. 4, pp. 333 – 347, 2003.

[14] P. Erdös and A. Rényi, "On Random Graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[15] "Distributed ASCI Supercomputer 3." `http://www.cs.vu.nl/das3`.

[16] "Distributed ASCI Supercomputer 4." `http://www.cs.vu.nl/das4`.

[17] A. Cayley, "A theorem on trees," *Quart. J. Math*, no. 23, pp. 376 – 378, 1889.

Appendix A

Random starting points

This appendix presents scaled-up version of scatterplots of random starting points, previously mentioned in Section 5.1. Each scatterplot shows connected graphs, selected out of 10 million generated graphs. The scatterplots are shown for $(150, 200, \ldots, 500)$ edges. The scatterplots for more than 500 edges can be found online at
`http://www.phoib.net/uni/graphs`

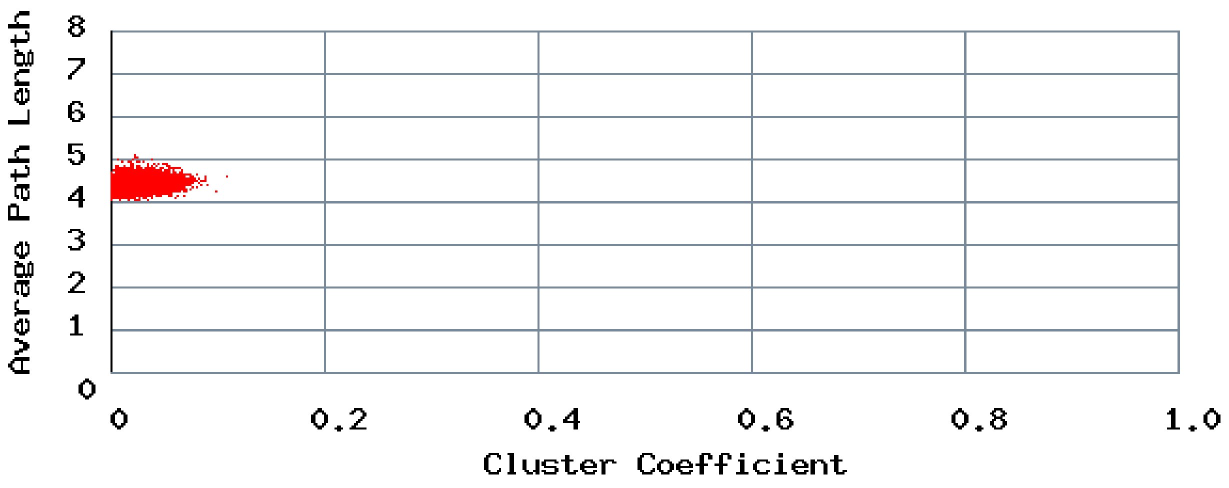


Figure A.1: Example of 10 million randomly chosen graphs at 150 edges.

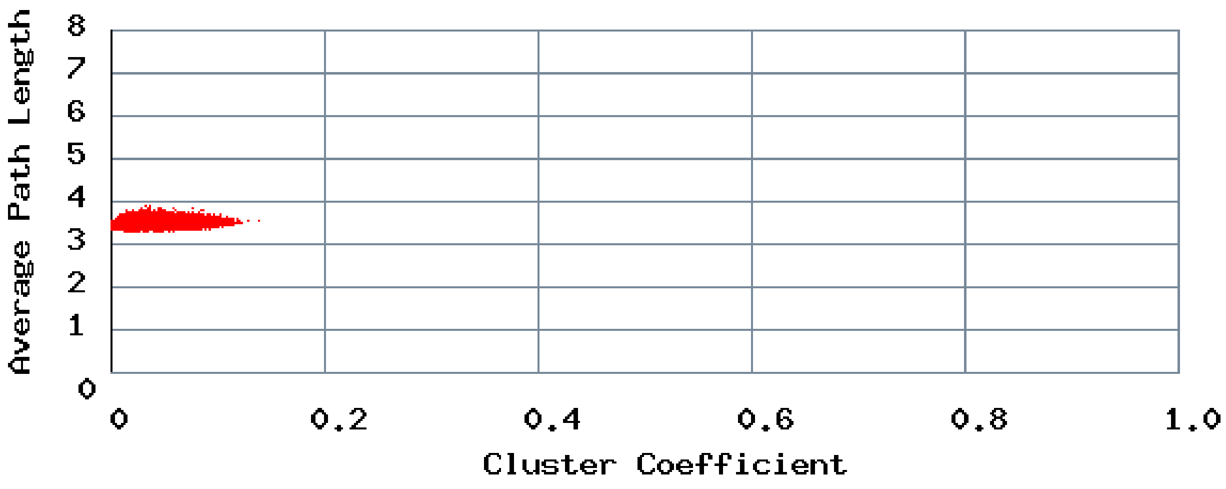


Figure A.2: Example of 10 million randomly chosen graphs at 200 edges.

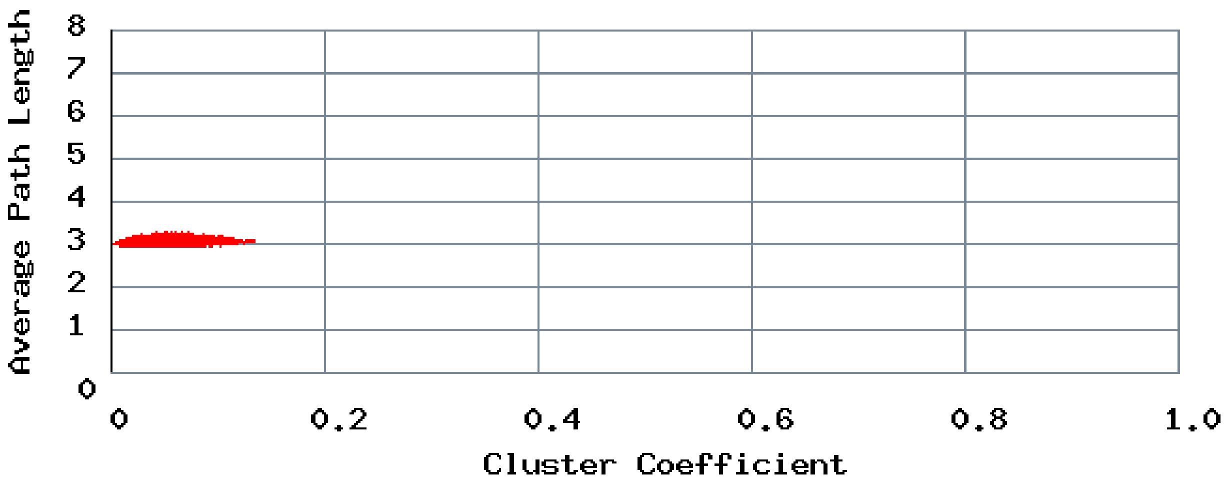


Figure A.3: Example of randomly chosen graphs at 250 edges.

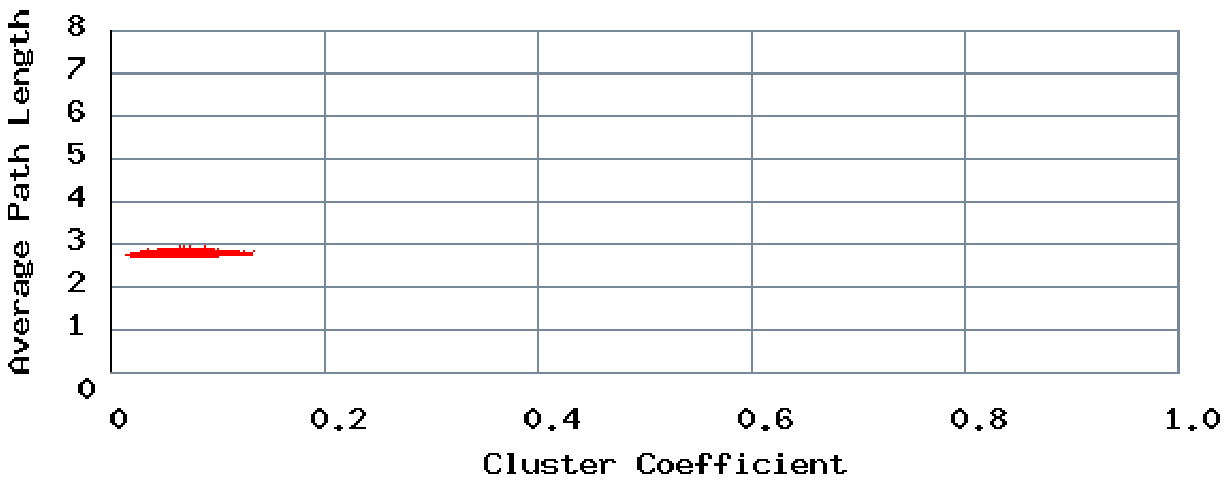


Figure A.4: Example of randomly chosen graphs at 300 edges.

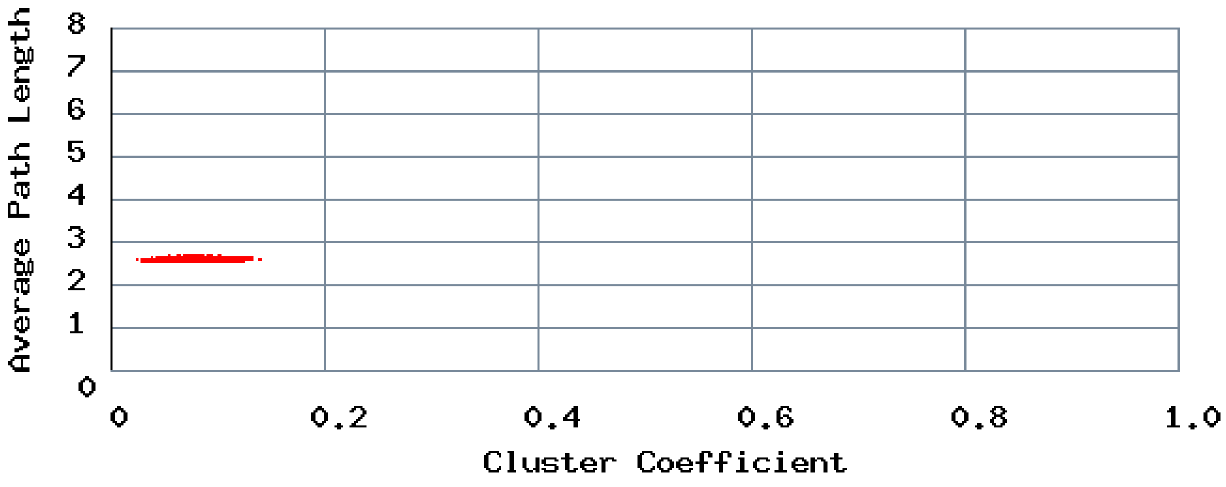


Figure A.5: Example of randomly chosen graphs at 350 edges.

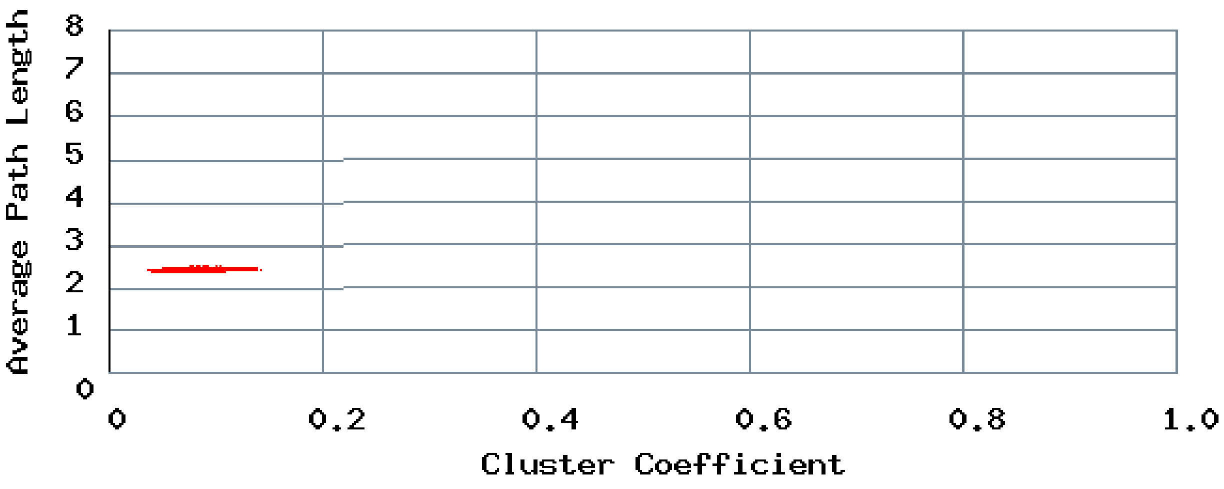


Figure A.6: Example of randomly chosen graphs at 400 edges.

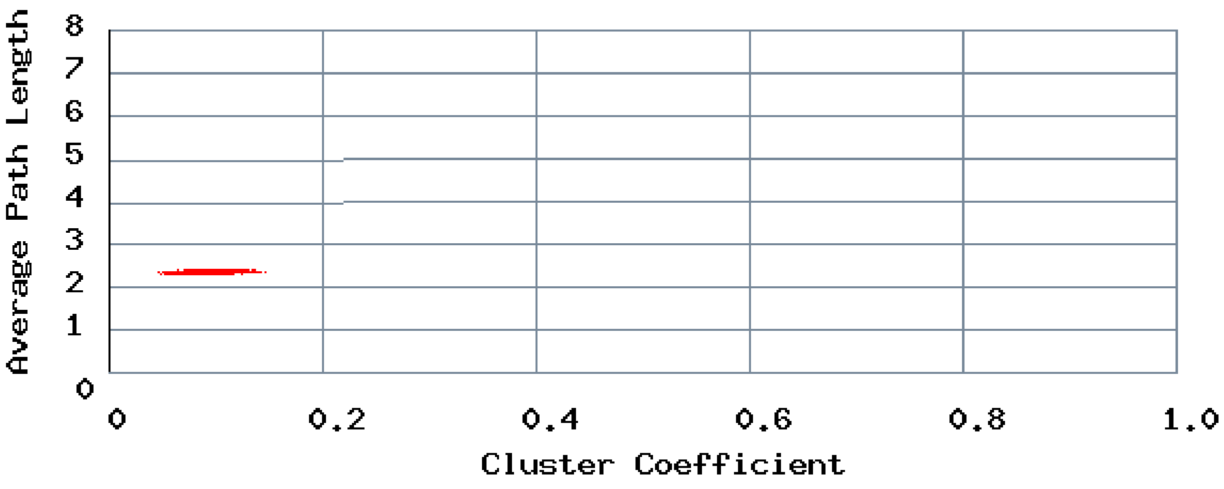


Figure A.7: Example of randomly chosen graphs at 450 edges.

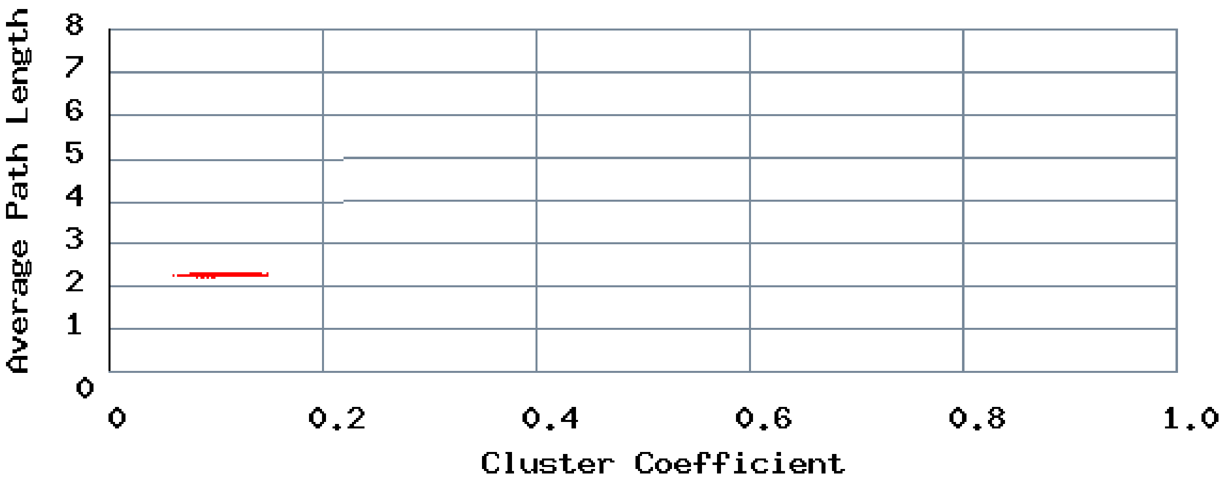


Figure A.8: Example of randomly chosen graphs at 500 edges.

Appendix B

Distributed Map Graph Formula

To explain the formula for the CC of the DM-graph, we introduce notations for several sets of vertices and edges. Recall from Section 2.2.1, a DM-graph is divided in two equal sized sets of vertices, L and R. Actually, both subsets are of equal size when the number of vertices, v, are even, and are of unequal size, otherwise. Also recall that, for a DM-graph of $\frac{1}{4}v_2$ edges or more, no edge can be added without increasing the CC. To minimize this increase of the CC, we increase the degree of a single vertex to the maximum degree $v - 1$. Because every vertex in L is connected to all the vertices in R, and vice versa, adding an edge to a vertex in one set will change the CC of all vertices in the other set. If only the vertices in L would get their degree maximized, the CC of the set R would grow faster than if the vertex which degree is maximized is alternated between sets L and R.

This method of alternating introduces several kinds of vertices and edges, which all need to be explained. To aid in the explanation, we introduce two new notations. Let Λ be a set of edges.

Definition B.0.1 $\Lambda(A, B)$ *is the set of edges, with one vertex from set A and another vertex from set B. By definition, all vertices in set A are connected to all vertices in B. The cardinality of this set is therefore $|A| \cdot |B|$.*

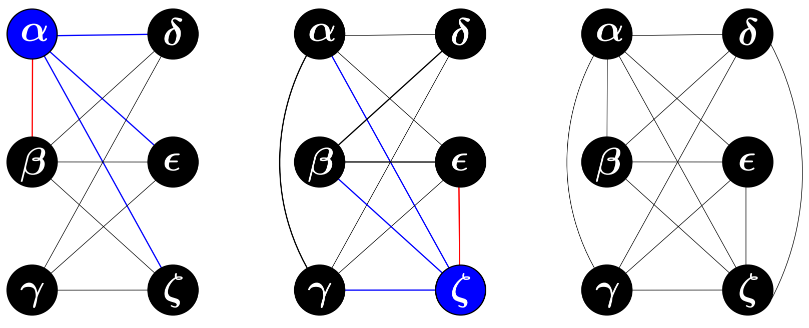
Let Θ be a set of vertices.

Definition B.0.2 $\Theta(A)$ *is the set of vertices which are adjacent to the vertices in set A.*

Also, for notation purposes, we will refer to the vertex, of which the degree is maximized, as a set. (It therefore gets an upper case Latin letter as designation, and not a lower case Greek letter.)

Set	Cardinality	Description	Label on the example
V	v	Vertices	all vertices in Figure B.1a
E	e	Existing edges	all edges in Figure B.1a
L	l	Vertices on the left side of the DM-graph	α, β and γ in Figure B.1a
R	r	Vertices on the right side of the DM-graph	δ, ϵ and ζ in Figure B.1b
E_L	e_l	$\Lambda(L, L)$	$\alpha\beta$ in Figure B.1a
E_R	e_r	$\Lambda(R, R)$	$\zeta\epsilon$ in Figure B.1b
V_{il}	1	Vertex in L to which edges are added	α in Figure B.1a
V_{ir}	1	Vertex in R to which edges are added	ζ in Figure B.1b
V_L	v_l	$\Lambda(V_{il}, L) \cup$ $\Lambda(V_{il}, R)$	$\alpha\delta, \alpha\epsilon, \alpha\zeta$ (blue) $\alpha\beta$ (red) in Figure B.1a
V_R	v_r	$\Lambda(V_{ir}, L) \cup$ $\Lambda(V_{ir}, R)$	$\alpha\zeta, \beta\zeta, \gamma\zeta$ (blue) $\epsilon\zeta$(red) in Figure B.1b
$\hat{V_L}$	$\hat{v_l}$ $= \frac{1}{2}v_l(v_l - 1)$	Possible edges to V_{il}	$\alpha\gamma$ (not shown) in Figure B.1a
$\hat{V_R}$	$\hat{v_r}$ $= \frac{1}{2}v_r(v_r - 1)$	Possible edges to V_{ir}	$\delta\zeta$ (not shown) in Figure B.1b
M	m	Vertices with maximum degree	α and ζ in Figure B.1c
M_L	m_l	$\forall\lambda, \lambda \in L \cup M$	α in Figure B.1c
M_R	m_r	$\forall\lambda, \lambda \in R \cup M$	ζ in Figure B.1c
L_c	l_c	$\forall\lambda, \lambda \in (\Theta(V_{il}) \cup L) \setminus M_L$	β in Figure B.1a
R_c	r_c	$\forall\lambda, \lambda \in (\Theta(V_{ir}) \cup R) \setminus M_R$	ϵ in Figure B.1b
L_n	l_n	$L \setminus (L_c \cup M_L)$	γ in Figure B.1a
R_n	r_n	$R \setminus (R_c \cup M_R)$	δ in Figure B.1a

Table B.1: Notations for the sets



(a) The edges for V_{il} (b) The edges for V_{ir} (c) M, M_L and M_R

Figure B.1: Examples to illustrate the sets

We will now describe, for every set, the method to calculate the CC of a vertex in that set. We will do this by

1. Identifying the neighbors of the vertex.
2. Determining the total number of possible edges between the neighbors of the vertex.
3. Calculating the number of edges between those neighbors.
4. Determining the formula for the CC for such a vertex.

- For a vertex in M:
    1. All other vertices in the graph are the neighbors of a vertex in M. There are $v-1$ neighbors in total.
    2. The number of possible edges for the neighbors of such a vertex is the maximum number of edges between those neighbors: $\frac{(v-1)(v-2)}{2}$.
    3. These $v-1$ neighbors require $v-1$ edges to connect to a vertex in M. All other edges in the graph are edges between these neighbors: $e-(v-1)$.
    4. The CC for a vertex in M is:

$$\frac{e-(v-1)}{\frac{(v-1)(v-2)}{2}} = \frac{2\big(e-(v-1)\big)}{(v-1)(v-2)} \tag{B.1}$$

- For the vertex V_{il}:
    1. It has v_l neighbors, that can be divided into three sets:
        - r vertices in R;
        - l_c vertices in L_c;
        - m_l vertices in M_L.
    2. For the number of possible edges, we have a special set: $\hat{v}_l$. See Table B.1.
    3. All of these neighbors of V_{il} have edges between each other.
        - All vertices in R have e_r edges between them, for a total of e_r edges.
        - All vertices in L_c are connected to all vertices in R: this gives a total of $r \cdot l_c$ edges.

  - All vertices in M_L are connected to all vertices in R: this gives a total of $r \cdot m_l$ edges.
  - All vertices in M_L are also connected to the vertices in L_C, for a total of $m_l \cdot l_c$ edges.
  - All vertices in M_L are also connected with each other, for a total of $\dfrac{m_l(m_l-1)}{2}$ edges.

4. The formula for the CC for V_{il} is:

$$\frac{1}{\hat{v_l}}\left(e_r + r \cdot l_c + r \cdot m_l + \frac{m_l(m_l-1)}{2} + m_l \cdot l_c\right) \tag{B.2}$$

- The CC of the vertex V_{ir} can be calculated similar to the vertex V_{il}, but by using the opposite sets: where for the vertex Vil sets were used from set L, sets will now be used from set R:

$$\frac{1}{\hat{v_r}}\left(e_l + l \cdot r_c + l \cdot m_r + \frac{m_r(m_r-1)}{2} + m_r \cdot r_c\right) \tag{B.3}$$

- For the vertices in the L_c set:

1. There are three types of neighbors for a vertex in L_c:
   - r vertices in R.
   - m_l vertices in the set M_L.
   - one V_{il} vertex.
2. The number of possible edges between all these neighbors is, according to Equation 1.1,

$$\frac{(r+m_l+1)(r+m_l)}{2}$$

3. All these neighbors have edges between them:
   - All vertices in R have e_r edges between them: e_r edges.
   - Between all m_l vertices in M_L there are $\dfrac{m_l(m_l-1)}{2}$ edges.
   - The m_l vertices also each have r edges towards R: $r \cdot m_l$ edges.
   - The V_{il} vertex has r edges towards R: r edges.

   for a total number of edges of:

$$e_r + \frac{m_l(m_l-1)}{2} + r \cdot m_l + r = e_r + r(m_l+1) + \frac{m_l(m_l-1)}{2}$$

4. The CC for a vertex in L_C is:

$$\left(e_r + r(m_l+1) + \frac{m_l(m_l-1)}{2}\right)\left(\frac{2}{(m_l+1+r)(m_l+r)}\right) \tag{B.4}$$

- Likewise, for vertices in the set R_c, the CC is:

$$\left(e_l + l(m_r+1) + \frac{m_r(m_r-1)}{2}\right)\left(\frac{2}{(m_r+1+l)(m_r+l)}\right) \tag{B.5}$$

- Vertices in the set L_n are the same as the vertices in the set L_c, with one exception: they are not connected to the V_{il} vertex. Thus:

  1. The neighbors for a vertex in the set L_n are thus r vertices of R, and m_l vertices of M_L.
  2. The total number of possible edges between the neighbors of a vertex in L_n are:
  $$\frac{(r+m_l)(r+m_l-1)}{2}$$
  3. The total number of edges between the neighbors of a vertex in L_n is the same as between the number of neighbors for a vertex in L_c, minus the edges of the V_{il} vertex towards the set R:
  $$e_r + r \cdot m_l + \frac{m_l(m_l-1)}{2}$$
  4. This makes the CC of a vertex in the set L_n:
  $$\left(e_r + \frac{m_l(m_l-1)}{2} + r \cdot m_l\right)\left(\frac{2}{(m_l+r)(m_l+r-1)}\right) \tag{B.6}$$

- Again, R_n differs only in all vertices being from the opposite set; thus, the CC is:

$$\left(e_l + \frac{m_r(m_r-1)}{2} + l \cdot m_r\right)\left(\frac{2}{(m_r+l)(m_r+l-1)}\right) \tag{B.7}$$

Hence, putting the Equations B.1, B.2, B.3, B.4, B.5, B.6 and B.7 together, the Equation for the CC of a DM graph is then:

$$
\begin{aligned}
CC_{Min} \quad = \quad & \frac{1}{v}\Bigg(m\bigg(\frac{2\big(e-(v-1)\big)}{(v-1)(v-2)}\bigg) + \\
& \frac{1}{\hat{v_l}}\bigg(e_r + r \cdot l_c + r \cdot m_l + \frac{m_l(m_l-1)}{2} + m_l \cdot l_c \bigg) + \\
& \frac{1}{\hat{v_r}}\bigg(e_l + l \cdot r_c + l \cdot m_r + \frac{m_r(m_r-1)}{2} + m_r \cdot r_c \bigg) + \\
& l_c\bigg(e_r + r(m_l+1) + \frac{m_l(m_l-1)}{2} \bigg)\bigg(\frac{2}{(m_l+1+r)(m_l+r)} \bigg) + \\
& r_c\bigg(e_l + l(m_r+1) + \frac{m_r(m_r-1)}{2} \bigg)\bigg(\frac{2}{(m_r+1+l)(m_r+l)} \bigg) + \\
& l_n\bigg(e_r + r \cdot m_l + \frac{m_l(m_l-1)}{2} \bigg)\bigg(\frac{2}{(m_l+r)(m_l+r-1)} \bigg) + \\
& r_n\bigg(e_l + l \cdot m_r + \frac{m_r(m_r-1)}{2} \bigg)\bigg(\frac{2}{(m_r+l)(m_r+l-1)} \bigg)\Bigg)
\end{aligned} \tag{B.8}
$$

Appendix C

Difference in randomly generated graphs

As mentioned in Section 5.1, there is a noticeable difference between the number of connected graphs as found by the crawler, and the number of connected graphs as predicted by Erdős and Rényi. We speculate that this is caused by the fact that we take a random sample from a finite set of graphs, and that the formula, as described by Erdős and Rényi, describes an average. Therefore, when the size of the set of graphs increases, any random sampling will become closer to the number predicted. Thus, as the number of vertices increases, we speculate that the number of connected graphs, as generated by the crawler, will approach the number predicted by Erdős and Rényi.

To test this, we generated sets of graphs, for different number of vertices and edges. We generated sets of graphs of 50, 100, 200, 500 and 1000 vertices. Each set consists of a number of subsets with the same number of vertices, but a different number of edges. Each subset consists of ten subsubsets of $10,000$ graphs, each with the same number of vertices and edges. Each of these ten subsubsets used a different random seed for its pseudo-random number generator.

To eliminate the possibility that the difference is caused by the pseudo-random number generator itself, we used a different pseudo-random number generator than the one used by the crawler. The crawler uses the standard C random number generator, while the series of graphs in this appendix were generated by a Java implementation of the Mersienne Twister. This accounts for the minute difference in the set for 100 vertices.

Edges	Generated in Java	Erdős and Rényi	Difference
50	0.00%	0.12%	-0.12%
70	3.34%	4.78%	-1.45%
90	29.29%	25.51%	3.78%
110	62.15%	54.13%	**8.02**%
130	83.01%	75.89%	7.12%
150	92.94%	88.34%	4.59%
170	97.09%	94.58%	2.51%
190	98.96%	97.53%	1.43%
210	99.57%	98.88%	0.69%
230	99.83%	99.50%	0.33%
250	99.95%	99.77%	0.18%
270	99.99%	99.90%	0.09%
290	99.99%	99.95%	0.04%
310	100.00%	99.98%	0.02%
330	100.00%	99.99%	0.01%
350	100.00%	100.00%	0.00%

Table C.1: Difference between connectedness for graphs of 50 vertices

Edges	Generated in Java	Erdős and Rényi	Difference
100	0.00%	0.00%	0.00%
150	0.33%	0.69%	-0.36%
200	17.19%	16.02%	1.17%
250	55.98%	50.98%	**5.00**%
300	82.16%	78.05%	4.11%
350	93.65%	91.28%	2.37%
400	97.81%	96.7%	1.11%
450	99.25%	98.77%	0.48%
500	99.74%	99.55%	0.19%
550	99.92%	99.83%	0.09%
600	99.98%	99.94%	0.04%
650	100.00%	99.98%	0.02%
700	100.00%	99.99%	0.01%
750	100.00%	100.00%	0.00%

Table C.2: Difference between connectedness for graphs of 100 vertices

Edges	Generated in Java	Erdős and Rényi	Difference
200	0.00%	0.00%	0.00%
250	0.00%	0.00%	0.00%
300	0.00%	0.00%	0.00%
350	0.13%	0.24%	-0.11%
400	2.26%	2.57%	-0.31%
450	11.22%	10.84%	0.38%
500	27.90%	25.99%	1.91%
550	47.43%	44.16%	**3.27**%
600	63.94%	60.91%	3.03%
650	76.79%	74.03%	2.76%
700	85.34%	83.33%	2.01%
750	91.28%	89.53%	1.75%
800	94.87%	93.51%	1.35%
850	96.82%	96.01%	0.80%
900	98.09%	97.56%	0.53%
950	98.88%	98.51%	0.37%
1000	99.34%	99.10%	0.24%

Table C.3: Difference between connectedness for graphs of 200 vertices

Edges	Generated in Java	Erdős and Rényi	Difference
500	0.00%	0.00%	0.00%
750	0.00%	0.00%	0.00%
1000	0.00%	0.01%	-0.01%
1250	3.37%	3.44%	-0.07%
1500	30.15%	28.96%	1.19%
1750	64.89%	63.39%	**1.50**%
2000	85.65%	84.56%	1.09%
2250	94.53%	94.02%	0.51%
2500	97.99%	97.76%	0.23%
2750	99.27%	99.17%	0.10%
3000	99.72%	99.69%	0.03%
3250	99.89%	99.89%	0.00%
3500	99.96%	99.96%	0.00%
3750	99.99%	99.98%	0.01%
4000	99.99%	99.99%	0.00%

Table C.4: Difference between connectedness for graphs of 500 vertices

Edges	Generated in Java	Erdős and Rényi	Difference
1000	0.00%	0.00%	0.00%
1500	0.00%	0.00%	0.00%
2000	0.00%	0.00%	0.00%
2500	0.10%	0.12%	-0.02%
3000	8.59%	8.38%	0.21%
3500	41.60%	40.18%	**1.42**%
4000	72.36%	71.50%	0.86%
4500	88.85%	88.39%	0.46%
5000	95.82%	95.56%	0.26%
5500	98.46%	98.34%	0.12%
6000	99.32%	99.39%	-0.07%
6500	99.79%	99.77%	0.02%
7000	99.93%	99.92%	0.01%
7500	99.97%	99.97%	0.00%
8000	100.00%	99.99%	0.01%
8500	99.99%	100.00%	-0.01%

Table C.5: Difference between connectedness for graphs of 1000 vertices

As the tables show, the difference between the number of connected graphs as generated by Java, and the number of connected graphs as predicted by Erdős and Rényi, decreases when the number of vertices increases. We consider a formal proof of this outside the scope of this thesis.