The small world is not enough. Or is it?

Annamaria Cucinotta
candidate number 42487

BSc Computer Science and Artificial Intelligence 2008
Department of Informatics
Supervisor: Dr Anil Seth

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Acknowledgements

The author wishes to thank A. Seth for the support and advice and L. Barnett and C. Buckley for useful discussion.

SUMMARY

Neural complexity was introduced in 1994 in [31] as a measure of interplay between functional integration and segregation in the brain. Using evolutionary selection it was shown in [18,22] that network topologies giving rise to high complexity had large dense clusters. Random directed networks with the same number of nodes and connections were used for comparison and their complexity was much lower.

Small-world networks are a class of network which have short paths and high clustering. In various papers [16-26] it has been shown that networks with high complexity, either artificial or natural, share the features of small world networks: they have short average path length, like random graphs, and high clustering coefficient, like lattices.

Entropy and integration are related measurements to complexity. Networks optimised for entropy or integration have different topologies and do not show evident small world attributes.

The initial aim of this project was to investigate the relationship between high complexity networks and small-world networks.

It was found that calculation of complexity, entropy and integration was more complicated than expected as even small changes in the kind of normalisation used change the numerical values of these measures for the same network. More importantly parameter variations affected the topology of resulting optimised networks. When using the same parameters to evaluate networks of different size or different number of connections it was not clear how the complexity values were affected by the choice of parameters, which may favour one network because of its size instead of its topology (e.g. some combinations of parameters may reward lower clustering more than others).

Later it was discovered [1,2] that the analytical process used to derive the covariance matrix from the network connection matrix was flawed. So the direction of the project moved to assessing the impact of this error on published research.

Using the correct covariance calculation, networks evolved for high complexity, entropy and integration seem to not resemble those evolved using the previous analytical covariance derivation and do not show small-world attributes.

The covariance matrices obtained using the correct procedure do not resemble the covariance matrices obtained from neurobiological data, so there is some doubt whether the Gaussian multivariate stochastic process model used in the definition of neural complexity is realistic.

The correct covariance matrix calculation is not immune to variations of the same parameters considered above.

Unless a rigorous revision of the neural complexity measure and its applications is undertaken, especially concerning the comparison of networks of different sizes, wider use seems unlikely.

Table of Contents

1 Introduction

This project investigates the relationship between network topology and neural complexity. It also assess the impact of an error recently discovered in the calculation of the complexity value on previous studies.

Tononi, Sporns and Edelman proposed a measure called neural complexity in 1994 [31]. This measure is intended to reflect the interplay between the functional segregation of brain regions and their integration, using methods from information theory. Neural complexity has then been used to examine brain connectivity data and to build simple models of neural networks.

Small-world networks are a class of networks characterized by short path lengths and high clustering. First introduced by Watts and Strogatz in 1998 [33] small-world networks have influenced several fields of research since then and have been widely used in biology, sociology, epidemiology, computer networks and networks in other contexts. Although the actual topology of small-world networks may vary significantly, a common understanding is that if a network is a small-world network it exhibits particular dynamics: short path lengths make any event happening anywhere in the network affect the other nodes quickly, while high clustering should protect the network from disconnections.

Small-world metrics have been introduced in the evaluation of actual and simulated brain networks in [16-26], in addition to neural complexity or on their own. It is generally accepted that brain connectivity exhibits small-world attributes: neurons are densely connected at the local level while long-range connections provide shortcuts between different brain areas. Small-world measures are simpler and faster to calculate than complexity, therefore it is tempting to evaluate the small-world metrics instead of complexity. It seems generally to be the case that a high small-world index is likely to be associated with high complexity.

However a paper by Buckley and Bullock [4] states that small-world networks do not necessarily have high neural complexity.

Furthermore in [1,2] Barnett et al. report that there is a serious error in the way neural complexity is calculated and propose the correct method. They also introduce a measure of approximate complexity, which is much faster to compute.

1.1 Structure of the report

The first chapter of this report will introduce background information on graph theory, small world networks, motifs, information theory and neural complexity.

The second chapter will present the methods and an overview of the software written for the project, although the full details are in the appendices.

In the third chapter previous experiments in this field will be replicated and the results obtained with the erroneous and correct covariance matrix calculation will be compared. These studies are mainly [7,18,19,22] which used evolutionary techniques to evolve high complexity graphs and compared them to various designed or

neurobiological networks.

The fourth chapter will discussed issues concerning various aspects of complexity.

In the fifth chapter relationships between network topologies, especially small-world networks, and complexity will be explored.

Finally conclusions will be drawn.

1.2 Background on graph theory

A network is mathematically defined as a graph. A graph consists of a set of vertices and a set of edges. An edge indicates a connection between two vertices: the two vertices which share the connection are called adjacent or, informally, neighbours. Connections can be represented using an adjacency matrix of size NxN, where N is the number of vertices. If no connection exists between two vertices the corresponding entry in the matrix is zero. Edges can be directed or undirected. Undirected edges are reciprocal so they correspond to two symmetrical entries in the adjacency (or connection) matrix. If there is an edge between two vertices the corresponding entry in the matrix (or both entries for undirected graphs) will be 1. Self-connections may or may not be allowed. A complete graph is one in which every node is connected to every other node. A graph is defined as sparse or dense according to its number of edges compared to the maximum possible. The topology of a graph is its structure as defined by the adjacency matrix. There could be weights associated to the edges, which could be positive or negative numbers. The weight associated to an edge can be stored in a separate matrix or sometimes in the adjacency matrix.

Graphs and vertices are more informally called networks and nodes, while edges may also be called connections or links. Here two graphs will be described as similar if they have the same number of nodes and connections but different topology, e.g. regular vs random. Graphs will be compared according to the various metrics described in this document to verify their similarity.

Several definitions from graph theory are used:

Walks and Paths: a walk is a sequence of contiguous edges which connect two arbitrary nodes. It may contain cycles, i.e. the same node is included more than once. A cycle is a path whose initial and final nodes are the same. A path is a walk with no cycles. The distance between two nodes is the length of the shortest path (there could be more than one) i.e. the path with the fewest edges for unweighted graphs. The diameter of a graph is the maximum distance between any pair of nodes. Often when referring to a path between two nodes the shortest path is intended. The average path length is the average of all the shortest paths between the nodes in the graphs.

Connectedness: a graph is connected if there is a path between each pair of vertices. If the graph is undirected the connectedness of a graph can be assessed fairly easily. For directed graphs the process is more complex. In this project the directed graphs will often be strongly connected, i.e. there will be paths between each pair of nodes in each direction. If a graph is not connected it is made up of different components. Separated components cannot interact with each other.

Degrees of a node: the indegree of a node is the number of incoming or afferent

connections, while the outdegree is the number of outgoing or efferent connections. If the graph is undirected the indegree and outdegree are the same and we just use the term degree. The degree distribution considers the degrees of all nodes. Different topologies of networks have different degree distributions. Of course the actual degree values depend on the number of edges and nodes in the graph.

Random graphs have connections assigned at random. They have short average path lengths and the probability for a node to have a certain degree follows a Poisson distribution. So in a random graph we can expect to find many nodes with about the same degree, but it is still possible to find a few nodes with significantly lower or higher degrees. There are formulas to calculate the average path length or determine with high probability if a random graph is connected on the basis of the number of edges and nodes. Their average shortest path length is fairly low.

Regular graphs have the same degree for all the nodes. They are generally represented as lattices of 1 or more dimensions in which each node is connected to a fixed number of neighbours. Average path length in regular graphs depends on the number of nodes and their degree, and is longer than the average path length in a similar random graph.

1.3 Small world networks

The small-world model proposed by Watts and Strogatz in 1998 [33] refers to networks whose characteristic path length is similar to that of a similar random graph, while the nodes are arranged in clusters. The original small-world model used a regular ring lattice as a starting point. According to a certain probability the connections of the graph are rewired. If the probability is high the resulting network will be a random graph. For medium probability values the resulting network will retain the high clustering of the regular lattice and a similar degree distribution while the new shortcuts will cause a sharp drop of the characteristic path length.

The original meaning of clustering coefficient of a single node is the proportion of direct connections existing between a node's neighbours, or in Watts' own words "in terms of social networks analogy, the clustering coefficient is the degree to which a person's acquaintances are acquainted with each other" [37, p.33]. The value for a graph is calculated as an average over the nodes in the network. The clustering coefficient of a one dimensional lattice with the same number of nodes and edges is used as a model of high clustering. The clustering coefficient is between 0 and 1. We can obtain high clustering when the graph is dense or when it is structured as a "caveman world", in which densely connected subgraphs are sparsely connected among themselves. Note that a cluster is not necessarily segregated from the rest of the network, as for example in the regular lattice.

The characteristic path length is defined as the median of the means of the shortest path lengths connecting each vertex to all other vertices [37, p.29].

Scale-free networks share some attributes with small-world networks. Barabasi and Albert provided a model for scale-free networks in 1999 [36]. Those networks exhibit a peculiar degree distribution. The degree distribution follows a power law, i.e. the probability that a node has degree K is proportional to some negative power of K. This means that in this type of networks the majority of nodes have a very low degree, while a few nodes, called hubs, have a very high degree. The Internet has been shown to be scale-free, as well as many other networks.

The small-world metrics are defined as averages over the graph, so there may be no uniformity in the local structure of the networks. More importantly the original small-world metrics were invented having in mind undirected graphs. The metrics have then been adapted to directed graphs, but the original meaning may have been altered. More recently other small-world metrics have been proposed: a small-world index measuring clustering coefficient and characteristic path length compared to a similar random graph and a community structure property which identifies groups of nodes which are tightly connected to each other while having looser connection to the rest of the network.

1.4 Information theory

Entropy and mutual information are information theory concepts which are used in the definition of neural complexity and other related measures. Information theory mainly applies to signal processing and data compression. Information theory concepts are applied to artificial neural networks since signal transmissions can represent the activity of the brain. Entropy measures the number of bits of information needed to effectively represent some generic entity. If the entropy in a system is less than the sum of the entropies of its components, it means that there are dependencies between them. Entropy is calculated on the basis of the distribution of possible values of a signal or system. There is a well known formula:

$$H(X)=0.5\log(2\pi e)^n|COV(X)|$$

to calculate the entropy of a multivariate normal distribution, that is a system whose variables are individually normally distributed and which are related by a given set of covariances. These covariances are normally given as a matrix, called the covariance matrix, which then represents the interdependency between the elements in the system. The values in the covariance matrix are high if the two elements are strongly correlated and zero if they are independent. The diagonal of the covariance matrix contains the variances of the single elements.

The mutual information:

$$MI(X,Y)=H(X)+H(Y)-H(X,Y)$$

measures the redundancy present in a combined system. Mutual information is always measured between two information sources. If we consider the entropy of two elements in the system and the combined entropy of the combined elements, their overlap is the mutual information between the two elements. To calculate the mutual information of X and Y, we sum their entropies and subtract the entropy of their combination.

The entropy can be calculated for a discrete or continuous distribution. The entropy of a continuous system can be negative, which is not intuitive.

1.5 Neural complexity

The measure of neural complexity $C_N(X)$ was introduced by G.Tononi, O. Sporns and

G. Edelman in 1994 in [31]. It starts from the observation that brain organization combines functional segregation of local areas with integration of their activity without the need of a master area. Neural complexity is a measure which, in the intentions of its authors, should evaluate a system so that integration and segregation are both rewarded, and that to score highly overall a system should possess both.

Although a complete understanding of brain organization has yet to be achieved there is knowledge on brain structure at both local and global level. It is known that some neuronal groups are highly specialised. Each neuron is connected to a small number of other neurons, either, more frequently, in the proximity or over longer distances. The brain exhibits small worlds attributes as it has short paths and high clustering [20].

The functional connectivity patterns are strongly related to the anatomical connectivity: the brain dynamics is determined by topology of its components' interconnections.

Neural complexity is applicable at various scales from neurons to cortical areas, to which we refer as elements. Those elements form a graph and their pattern of connectivity is represented by an adjacency matrix.

The system is assumed to be stationary with no external input. Random noise is injected into each node. Under stationary conditions the entropy of a Gaussian multivariate process can be determined from its covariance matrix using the formula given in 2.3.

The integration measures the system's overall deviation from statistical independence [22]:

$$I(X)=\Sigma_i H(x_i)-H(X)$$

Integration is a non negative value. It is zero when its components are statistically independent.

The neural complexity can be expressed in terms of either mutual information or integration [22]. In both cases all the possible bipartitions of the system are to be taken into account. However actual implementation of this is completely infeasible for systems with more than a few nodes and a sampling approach is used instead.

$$C_N(X)=\Sigma_k(K/2)I(X)-\langle I(X_j^k)\rangle$$
$$C_N(X)=\Sigma_k\langle MI(X_j^k;X-X_j^k)\rangle$$

A simplified version of complexity, C(X), just considers bipartitions consisting of a single element and its complement [22].

$$C(X)=H(X)-\Sigma_i H(x_i|X-x_i)$$
$$C(X)=\Sigma_i MI(x_i;X-x_i)-I(X)$$
$$C(X)=(n-1)I(X)-n\langle I(X-x_i)\rangle$$

Graphically neural complexity and complexity are (from [28]):

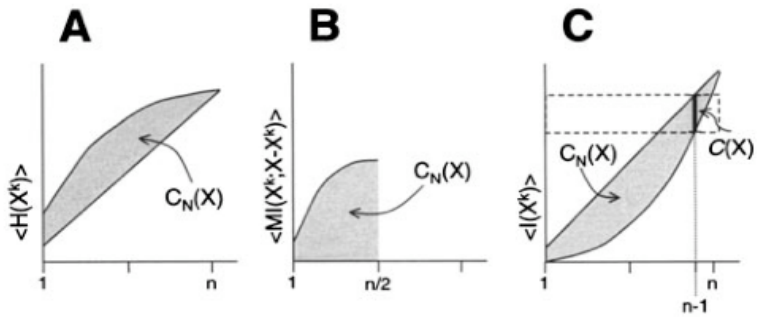

Fig. 1.1

1.5.1 Topology and complexity

According to [18,26,31] high values of neural complexity are found in networks with mostly reciprocal connections and dense clusters.

The authors implemented [17,18,19,22] simulations to explore what type of topologies give rise to high complexity. Other topological measurements are also considered in relation to complexity such as path length, clustering coefficient, cycles of short lengths and motifs.

De Lucia et al. [7] attempted to relate complexity to topology, giving a formula to calculate an approximate neural complexity from the connectivity matrix. The measure applies only to undirected graphs. According to [1,2] there is an error in they way the covariance of subsystems is calculated in [7].

In [2], Barnett et al. propose a measure of approximate neural complexity. This measure is computationally much cheaper than neural complexity as it scales polynomially instead of exponentially with network size. This measure is calculated from the normalised connection matrix:

$$C^{*}=\frac{1}{24}(n+1)\,\Sigma_{i\neq j}(U_{ij})^{2}$$

where *n* is the number of units and *U* is the product of the normalised connection matrix and its transpose.

They also note that the method used to the covariance matrix used in previous studies is incorrect.

In [1] the approximate neural complexity is related to the underlying graph structure. In particular the complexity values depends on the abundance of motifs in Fig. 1.2. The importance of motifs may be affected by the weight normalisation as different connection weights contribute proportionally to complexity.

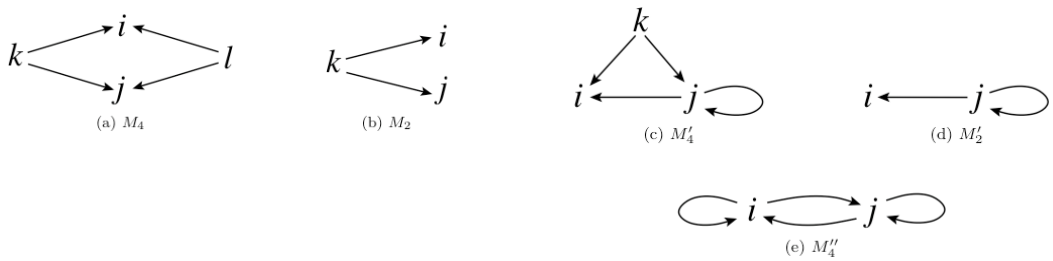

Fig. 1.2 motifs for high complexity

1.5.2 Other measures

The same authors also introduced similar measures based on the same concepts from information theory which are not being considered in this project.

Matching complexity [27,29] measures the change in neural complexity depending on inputs from the environment.
Functional clustering [27,30] identifies strongly interactive brain regions during a particular task.
Degeneracy and redundancy [28] measure how different elements may produce the same output. They are calculated using mutual information between subsets and output.
Φ , capacity to integrate information [32] is defined in terms of effective information between two complementary subsets of a system.

Most of the above measures make use of the analytical derivation of the covariance matrix from the connectivity matrix. Due to the error on this computation those measures are also affected.

1.6 Covariance matrix

The activity of a system is captured by its covariance matrix, which, as described in section 1.4, is used to calculate its entropy and is therefore very important in the present context.

Covariance matrices can be derived from neuroimaging data as described in [30]. Covariance matrices can also be derived from a connection matrices by running a system as a dynamical system [22].

In [17,18,19,22] the covariance matrix is calculated analytically from the connection matrix, assuming that the system is linear and the activity of its components is a Gaussian multidimensional stationary stochastic process. The assumption is that when the components settle under stationary conditions, the vector A of the random variables representing the system has the same values as before.

$$COV = \langle A^T * A \rangle = \langle Q^T * R^T * R * Q \rangle = Q^T * Q$$

$$A = C_{ij} * A + R$$
$$Q = [1 - C_{ij}]^{-1}$$

where C is the adjacency matrix of the system and R is uncorrelated Gaussian noise.

To calculate the covariance matrix from the adjacency matrix this needs to be normalised and preprocessed. The original authors in [18] consider some constraints motivated by properties of real neurobiological networks which need to be used for the normalisation.

The "Saturation constraint" imposes that the maximum total input to each node, considering the absolute value for negative weights, i.e. self connection corresponding to the variance, is always set under 1. In [18] the sum of afferent connections for each node, excluding the self connection, was set to 0.8 for the graph selection process. Since the indegree of each node was 8, the single weight value was 0.1. In the same paper the single weight value was instead set to 0.04 to calculate the complexity of the cat cortex and macaque visual cortex.

The "Activation constraint" requires that small self-inhibitory weights are added to the connection matrix such that the variance, that is the diagonal of the covariance matrix, is equal to a uniform fixed value. The absolute value of the self-connection weight should be less than the other connection weight values. The variance in [18] is set to 0.015 for the evolutionary processes, while it is set to 0.01 for the cortical data. In [22] the variance is set to 0.01. The activation constraint is not mentioned in more recent papers and the standard complexity MatLab toolbox does not include it.

For both the saturation and the activation constraints there are no specific indications on what values are more suitable for different network sizes and/or connection density. The impact of using different values on the complexity is not specified and will be investigated in this report.

In [1,2] the role of normalisation and scaling of the connection matrix is examined and several issues are discussed. To allow the system to be stationary the connection matrix must be scaled so that its maximum eigenvalue is less than 1. The authors propose a damping normalisation of the connection matrix such that the maximum eigenvalue of the matrix is a fixed arbitrary positive number less than 1. They also consider other plausible methods of normalisation such as using different connection weights such that the sum of afferent connections for each node is constant, as proposed in [31], although they question how this method could deal with negative weights.

The last constraint, the "Connectedness constraint" is not directly related to the covariance matrix and imposes that the networks must be strongly connected, although calculating the covariance matrix and complexity of a not strongly connected graph is numerically possible.

In [1,2] was found out that there was an error in the calculation of the covariance matrix from the connection matrix. The error consists in the fact that in a stationary multivariate Gaussian process, the stationarity refers to the distribution of the element values, not to the values themselves. If the connection matrix is symmetrical it possible to derive the correct covariance analytically. For directed graphs it is necessary to perform an iterative process which expands a power series until the required stationarity is reached.

The impact of this error seems to affect heavily the importance of reciprocal

connections for high complexity. A more practical approach on the evaluation of this error is presented in this report.

1.7 Motifs

Motifs are connectivity patterns observed in small groups of connected nodes in a graph. Fig. 1.3 shows typical motifs including 3 or 4 nodes. Motifs have been described as building blocks of complex networks [11, 26]

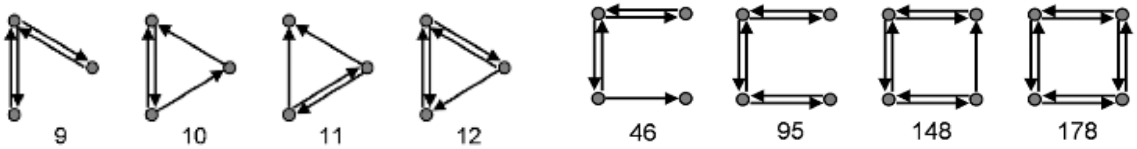

Fig. 1.3

Milo [11] analysed statistically the number of motifs of certain types in biological and artificial networks, suggesting that motifs of certain types are much more common in some types of networks than in others.

Some of these motifs are short re-entrant circuits while other are unidirectional. They also differ in the number of connections and reciprocal connections. Re-entrant motifs are another name for short cycles. Cycles of size 2 are reciprocal connections. Statistical analysis of cycles of different lengths is also included in [18]. High complexity networks and cortex networks have a much higher quantity of short cycles compared to equivalent random graphs.

The way motifs can be counted is somewhat ambiguous: motifs with more nodes and connections include several simpler motifs. Therefore statistical analysis of graph motifs may or may not include simpler motifs in the count or count just the most complex motif. For example if between 3 nodes there is a motif n. 12 we could also count for the same 3 nodes 1 motif n. 9, 1 motif n. 10, 1 motif n. 11 and others. When counting motifs of size 4 they will contain motifs of size 3. So when looking at statistical data we need to pay attention to what was actually being counted. Sporns and Kotter [26] proposed a distinction between structural and functional motifs. A single instance of structural motif may contain several instances of functional motifs. The same paper concludes that networks optimised for complexity are also optimised for functional motif numbers and vice versa, while networks optimised for structural motif numbers show low complexity.

2 Methods

2.1 Professional considerations

This project is mainly research based and the software produced is only intended for use by the author. As a result many of the ethical considerations in the BCS Code of Practice and Code of Conduct seem of limited relevance. However section 15, "You shall not claim any level of competence that you do not possess." does apply. It is true that when I began the project I knew nothing about neural complexity, but I had some experience with MatLab and the basics of graph theory. Before choosing the project I read the papers referred to in the project description and felt confident that my previous experience and knowledge was sufficient to undertake this challenge.

Source code from other sources is acknowledged in section 2.2.

2.2 Requirements analysis and implementation

The software to be written needed to provided a framework for an evolutionary procedure where various parameters could be set independently so that it was possible to replicate and extend the experiments reported in various papers.

A function to reorder the connection matrix such that clusters were readily visible was needed, together with programs to compare complexity in different graphs and wrapper code for various tasks.

The software in the project was to be implemented in MATLAB for compatibility with existing code.

The graphs are strongly connected in nearly all cases. Graphs are usually sparse to resemble biological networks. The graphs are unweighted, although most programs can deal with weighted graphs as well. The numerical values, connection matrices and graphs used in this report usually refer to just one specific case and not as an average, although the same experiments have been repeated numerous times and the results confirmed.

Original code was written for the above functions, while software from the following sources has also been used.

For the computation of complexity and related measurements, MatLab functions provided by O. Sporns (http://www.indiana.edu/%7Ecortex/resources.html) were used. For graph and small-world metrics, including path lengths and clustering coefficients, the same toolbox was used. Cortical data of the cat cortex and macaque visual cortex are from the same source.

From results obtained from the various simulations in this project it is not very clear if the same functions were used in the reference papers [18,22]. In particular the adjacency matrix normalisation has without doubt been done in a different way.

The covariance matrix calculation with variance normalisation was provided by A.

Seth. The correct covariance matrix calculation function and the approximate complexity function were provided by L. Barnett.

2.3 Graph metrics

The measurements used to evaluate networks are:

- diameter: maximum distance between any two nodes.
- characteristic path length: average distance between all pairs of nodes.
- reciprocal connections: proportion over all connections
- clustering coefficient as defined for small world networks
- indegree and outdegree minimum and maximum
- weight normalisation: usually the specified weight is used for all the connections in a graph
- variance normalisation: value on the covariance matrix diagonal, when appropriate

For comparison four different type of graphs are provided: directed random graph (with fixed indegree when appropriate), undirected random graphs, lattices and small world networks (rewired lattices with rewiring rate 0.1).

2.4 Software

The source code written for this project is described in appendix 8.2.

2.4.1 Evolutionary procedure

The evolutionary algorithms implemented are similar to those used in the original papers. More details in appendix 8.2.

2.4.2 Rewiring

Rewiring consists in moving the origin or destination of a connection and it is used for the evolutionary procedure and also for turning a lattice in a small-world network. More details in appendices.

2.4.3 Community structure

A community structure algorithm [10,12,13] is used to order the connection matrices and display the structure of a network. More details in appendices.

2.4.4 Other programs

See appendices.

3 Repeating and extending previous findings

3.1 Evolving high complexity networks

3.1.1 Using original covariance calculation

In [17,18,19] random networks were evolved to achieve high complexity, integration and entropy. The motivation to use an evolutionary algorithm approach was that even for small graphs (both nodes and connections) the variety of topologies was enormous and an exhaustive search in the graph space was not feasible.

The networks to evolve had 32 nodes and 256 connections and the number of nodes and connections did not change during the evolutionary process. The weight for each connection was constant and uniform across all connections. The indegree was fixed to 8 for every node. The starting graphs were random graphs with this indegree value. The fitness function was the complexity measure C(X), the integration I(X) or the entropy H(X).

The population size was 10 and at each iteration the population was replaced in the following way: the individual with the highest fitness score was selected and copied to the new population, while the remaining individuals in the new population were mutated versions of the best individual of the previous population. The mutation consisted in the rewiring of the source of a very limited number of connections (from 1 to 3) while the destination, and hence the indegree of each node, remained unchanged. The number of generations was set to 2000.

The covariance matrices were obtained analytically from the connection matrices. In particular the weights were set to 0.1 so that each node had a constant input of 0.8, while the variance for each node, diagonal terms on the covariance matrices was set to 0.015. The amount of uncorrelated noise, R in the covariance calculation formula was not specified.

Replicating the same experiment using R value of 0.1, leads to the following connection matrices:

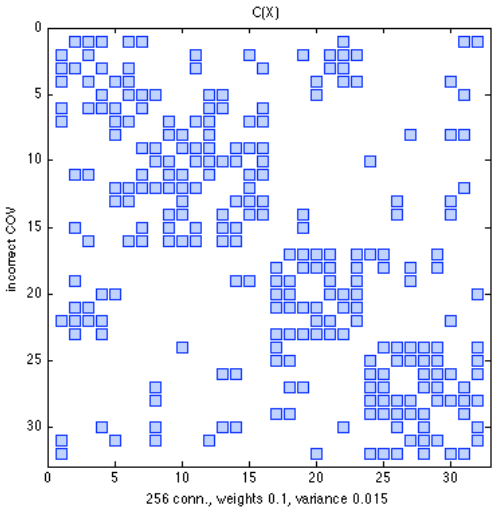

Fig 3.1 C(X)

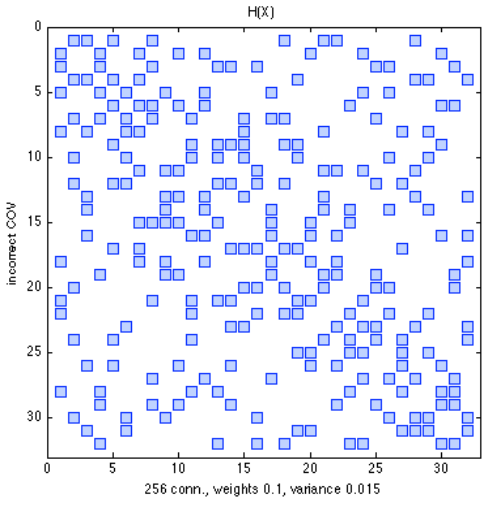

Fig. 3.2 H(X)

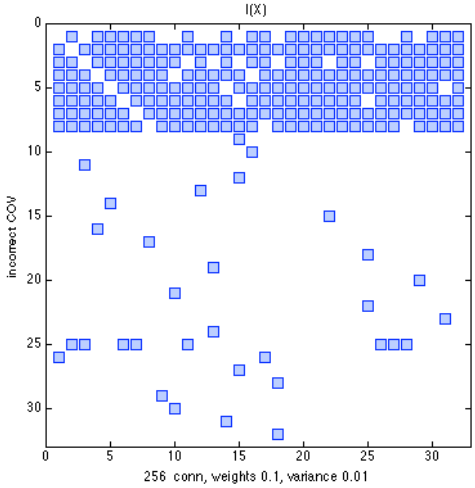

Fig. 3.3 I(X)

The evolved graphs were compared to random graphs, probably with the same indegree as the evolved graphs, although this detail was not specified. In this report undirected random graphs, lattices and small world networks (rewired lattices with rewiring probability rate of 0.1) are included for comparison.

Networks → Values ↓	C(X) max	H(X) max	I(X) max	Random undirected	Random directed	Lattice	Small world
C(X)	0.0858	0.0814	0.0446	0.0805	0.058	0.0846	0.0782
$C_N(X)$	21.05	18.08	13.59	18.95	15.19	22.79	20.58
H(X)	-28.63	-28.19	-29.59	-28.66	-28.61	-29.59	-29.1
I(X)	6.842	6.34	7.783	6.867	6.814	7.798	7.307
Diameter	3	3	7	3	3	8	4
Char.Path	1.98	1.77	3.03	1.83	1.81	2.88	2.08
Recipr. c.	0.99	0.98	0.23	1	0.25	0.95	0.79
Clustering	0.443	0.062	0.649	0.238	0.25	0.702	0.518
Outdegree min/max	7/9	6/9	1/31	3/14	4/13	4/10	3/13
Weight	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Variance	0.015	0.015	0.015	0.015	0.015	0.015	0.015

The above results are in line with the paper results, although the numerical values of complexity seem to be different to those shown in the plots. This could be due to a different approach to variance normalisation. Entropy and integration values are similar to the originals.

Networks optimised for high complexity have high clustering, short paths and nearly all reciprocal connections. Networks optimised for entropy, random undirected graphs and lattices also have a quite high complexity. The only attribute they share is the very high rate of reciprocal connections.

$C_N(X)$ is not always optimised by C(X). The highest value of $C_N(X)$ is found in the lattice, but C(X) is maximum, as it is expected, in the network optimised for

complexity.

The outdegree distribution of networks optimised for integration is quite peculiar, as the majority of nodes have just one efferent connection, while the other nodes have outgoing connections to all the other nodes. The indegree is fixed to 8 for every node. They look like an extreme version of scale free networks.

What is really unexpected is that the lattice has even a higher integration value than that of a network evolved specifically for high integration (after 2000 evolution cycles). The two networks have in common only a high clustering coefficient. Repeating the evolutionary procedures for 10,000 iterations the obtained integration value is 7.83 and the structure is similar to the previous evolved network, although the path length is shorter.

Networks optimised for entropy have extremely low clustering, much lower than random networks and mostly reciprocal connections.

Rewiring a lattice decreases its complexity while the entropy increases.

Although the reference paper claims that 2000 generations are sufficient to stabilise the results, in this report it was found that the fitness values kept increasing very slowly but constantly if the number of generations was bigger. In particular when evolving for complexity the cluster boundaries will become sharper if this number is increased, as shown for 10,000 iterations. in Fig. 3.4.

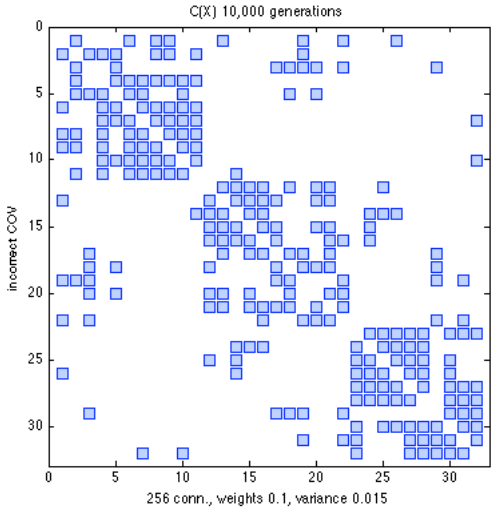

Fig. 3.4

3.1.2 Using correct covariance calculation

Unfortunately the covariance matrix was not calculated in the correct way. Repeating the evolutionary process using the correct covariance calculation yields the following results (graph sizes are the same as above):

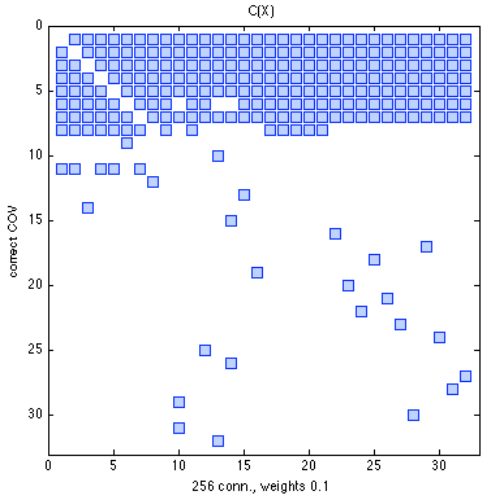

Fig. 3.5 C(X)

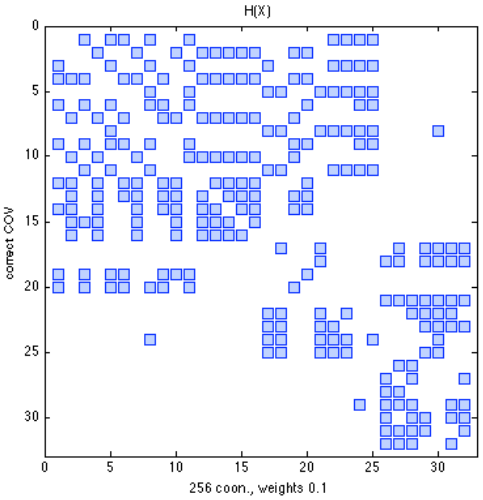

Fig. 3.6 H(X)

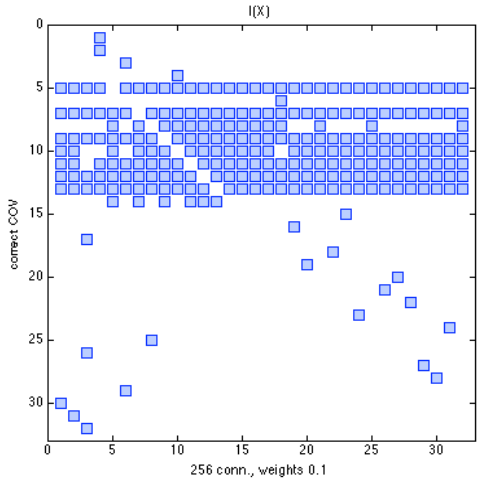

Fig 3.7 I(X)

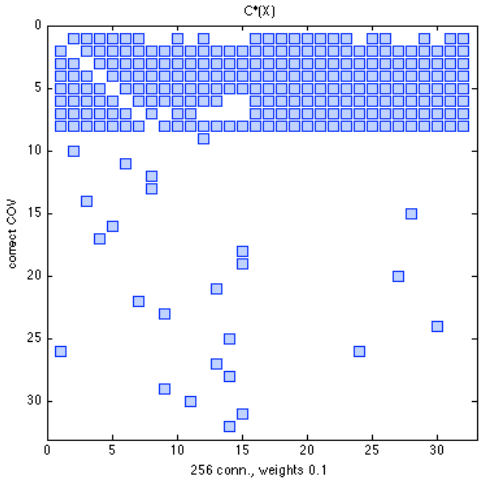

Fig. 3.8 C*(X)

The following table reports the same measures as in section 3.1.1 using the correct covariance calculation, with the addition of the approximate complexity C*(X).

Networks → Values ↓	C(X) max	C*(X) max	H(X) max	I(X) max	Random undirected	Random directed	Lattice	Small world
C(X)	0.0164	0.0159	0.0145	0.0166	0.0103	0.0067	0.0134	0.0112
$C_N(X)$	4.6155	4.4342	3.399	4.9795	2.4704	1.5183	3.0274	2.5363
C*(X)	5.9793	6.0563	1.4058	5.8627	0.7824	0.7309	1.1968	1.0238
H(X)	-27.09	-27.12	-26.54	-27.08	-26.61	-26.8	-26.52	-26.61
I(X)	1.701	1.609	0.903	1.72	0.693	0.381	0.758	0.643
Diameter	12	5	15	14	3	3	8	4
Char.Path	4.17	2.52	4.24	4.98	1.83	1.81	2.88	2.08
Recipr. c.	0.27	0.28	0.42	0.27	1	0.25	0.95	0.79
Clustering	0.673	0.682	0.495	0.664	0.238	0.25	0.702	0.518
Outdegree min/max	1/31	1/31	2/13	1/31	3/14	4/13	4/10	3/13
Weight	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Evolving for complexity or integration leads to the same network topology. None of the evolved networks, except perhaps the one evolved for C*(X), is a small-world network as their path length values are too large, although their clustering coefficients are quite high. Reciprocal connections for high complexity networks is similar to that of random directed graphs.
The relationship between C(X), $C_N(X)$ and C*(X) is examined in section 4.7.

Rewiring the lattice causes the complexity and the other values to fall.

3.2 Biological networks and complexity

The following table compares the complexity values for the cat cortex and macaque visual cortex to those of directed and undirected random networks, lattices and rewired lattices. For the macaque visual cortex the weights are normalised to 0.04 as used in [18]. For the cat cortex the weights are normalised to 0.03, as using 0.04 would have caused the sum of the incoming weights of some nodes to be greater than 1. The first and second complexity values are obtained using the incorrect covariance calculation. The first complexity value uses variance 0.01, thus allowing comparison with [18] while the second value does not use variance normalisation. The third complexity value is obtained using the correct covariance calculation.
The cat cortex connection matrix available with the complexity toolbox is different from the one used in [18] as it has fewer cortical areas (52 instead of 65) and connections (818 instead of 1136).

	Macaque visual cortex	Random undirected	Random direct	Lattice	Small world
Nodes	32	32	32	32	32
Connections	315	314	315	315	315
Diameter	4	3	3	6	3
Charact. Path L.	1.77	1.72	1.71	2.43	1.89
Reciprocal conn.	0.77	1	0.27	0.96	0.81
Clustering	0.554	0.324	0.316	0.729	0.535
Indegree min/max	0/19	5/15	5/16	5/12	5/12
Outdegree min/ max	2/20	5/15	5/16	5/12	6/14
Weight	0.04	0.04	0.04	0.04	0.04
Orginal Complexity C(X) var. 0.01	0.01429	0.0156	0.01038	0.01607	0.0147
Original Complexity C(X) no var.	0.01557	0.01669	0.01081	0.01738	0.01572
Correct COV C(X)	4.69 e-4	2.74 e-4	2.62 e-4	3.97 e-4	3.41 e-4

	Cat cortex	Random und.	Random direct	Lattice	Small world
Nodes	52	52	52	52	52
Connections	818	818	818	818	818
Diameter	4	3	3	6	3
Charact. Path L.	1.81	1.7	1.7	2.49	1.86
Reciprocal conn.	0.74	1	0.31	0.98	0.83
Clustering	0.552	0.311	0.31	0.753	0.566
Indegree min/max	7/32	9/23	7/23	8/18	8/18
Outdegree min/ max	3/34	9/23	9/21	8/18	9/20
Weight	0.03	0.03	0.03	0.03	0.03
Original Complexity C(X) var. 0.01	0.01304	0.01433	0.00982	0.01508	0.01385
Original Complexity C(X) no var.	0.01426	0.01532	0.01028	0.01641	0.01494
Correct COV C(X)	6.28 e-4	3.32 e-4	3.21 e-4	5.84 e-4	4.62 e-4

One thing which is surprising when using the incorrect covariance calculation is that undirected random graphs, as well as lattices and rewired lattices, have higher complexity than the cortical matrices.

Using the correct covariance calculation the biological networks have the highest complexity among the other networks considered for comparison.
However the biological networks do not resemble much the networks evolved for complexity considered in section 3.1.2.

4 Evaluating complexity

Complexity was proposed as a measure to evaluate some properties in a network and was used to compare networks with the same or similar number of nodes and connections. In general the comparison is made with random directed networks of the same size and in [25] a scaled complexity measure was used, defined as C(network)/ C(random). This could be an appropriate way to compare networks of different size if some conditions are met. But other relevant issues seem not have been addressed yet. Those issues are still relevant when using the correct covariance calculation.

4.1 Connection and covariance matrix normalization and complexity

The role of normalisation of the adjacency matrix is treated as marginal by the original authors. It is noted that the weight values must be very small so that the sum of afferent connections for each node is less than 1. Since in [18] the networks used have a fixed indegree value, the weights are stable during the evolutionary process. In older papers the details of the weight normalisation and variance are specified while in more recent papers they are not.

In [1,2] it is noted that the largest eigenvalue of the normalised connection matrix must be less than 1 to guarantee a stationary process, otherwise the system will not settle. The authors also note that the numerical values of complexity depends on the scaling of the connection matrix and therefore complexity values are only comparable if the normalisation method is equivalent. To do this they propose to normalise the connection matrix in such a way that its maximum eigenvalue is a fixed value less than 1. In this way complexity values should be comparable among different networks.

The standard complexity function provided in the toolbox sets the weights to 0.01, which could not be appropriate for large networks. The weight values are different from those used in [18], where weights were set to 0.1 and 0.04. Also in various papers it is said that the diagonal of the covariance matrices is to be set to a constant value, e.g. 0.015 in [18] for graph selection and 0.01 in [18] for neural data (cat and macaque cortex), 0.01 in [22] for graph selection.

Using the complexity toolbox it is not possible to replicate the graph selection process as described in section 3.1.1 and [18].

The covariance calculation function in the complexity toolbox does not add inhibitory self connections such that the variance, the diagonal of the covariance matrix, is equal to a specific value for all the elements. The covariance calculation function used in this project was provided by A. Seth. The process to normalise the covariance diagonal iteratively adjusts the negative self-connections until the desired variance is achieved. It is non-trivial and certainly deserves to be included in the complexity toolbox if it is considered important. The numerical values of complexity do not seem much different whether the variance normalisation is used or not, so this could be the reason for its exclusion.

One problem with the variance normalisation is that depending on the particular matrix it cannot be guaranteed that the inhibitory self-connections are always smaller

than the connection weights, which seems to be a requirement in [18]. It could actually happen that the self-connections weights need to be positive to make the covariance matrix diagonal have the specified value.

In [22], Sporns and Tononi published results which are in contradiction with the ones published in [18]. In [22] are reported results obtained evolving networks for high complexity, entropy and integration. The connection matrices are (from [22]):

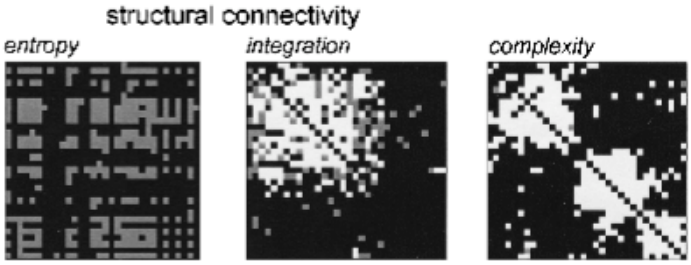

Fig. 4.1

White indicates reciprocal connections, grey unidirectional connections and black no connections. The above connections for entropy and integration are very different form those published in [18] and repeated in 3.1.1:

entropy *integration* *complexity*

Fig. 4.2

In [18] networks evolved for entropy had 99% reciprocal connections, while in [22] they have none. Networks evolved for integration look completely different, the number of reciprocal connections has increased dramatically form [18] to [22]. Networks evolved for complexity have changed, as the clusters are more segregated in [22], although structurally (reciprocal connections and clustering) they are more alike.

The main difference seems to be that in [18] the networks had 256 connections while in [22] they had 320. The number of nodes was 32 in both cases. The weights were set to 0.1 in [18] and to a constant value "such that the amount of input for each node is less than 1" in [22]. Variance was 0.015 in [18] and 0.01 in [22]. It is not clear if in [22] the number of incoming connections is uniform as in [18], as the initial graphs are just described as random and looking at the connection matrix for integration there are definitely columns with very few connections.

To find an explanation for the above various combinations of weight and variance values have been tried to replicate the same results. The following connection matrices are obtained using connection weights of 0.01 while the variance is 0.01 and evolving for complexity, entropy and integration.

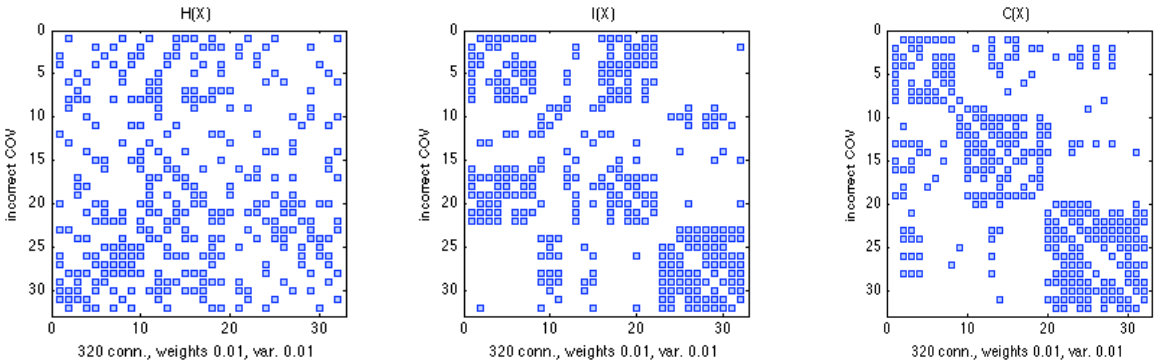

Fig. 4.3

The networks in Fig. 4.3 are more similar to those in Fig. 4.2. Examining the values on the self-connections for the variance normalisation of the three evolved networks it was found that all the self-connections values were negative and their absolute value was smaller than the normalised connection weights (about -0.003 while connection weights were 0.01). If the variance had been set to 0.015 the same networks would have had a positive value as self-connection. The network evolved for integration in 3.1.1, when normalised using weights 0.1 and variance 0.015, has a few self-connections (which are all negative) whose absolute value is greater than 0.1.

Repeating the same using the correct covariance does not change much the results obtained in 3.1.2:

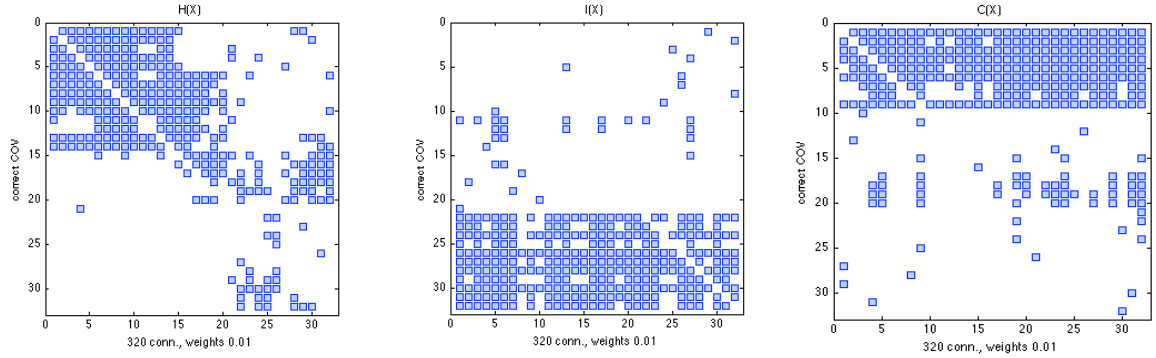

Fig. 4.4

From a practical approach the normalisation affects the numerical values of the complexity but also its dependency on the network structure. Changing the normalisation method changes the results of the evolutionary processes. Therefore the ranking of networks according to their complexity values depends on the way the normalisation is done as has just been shown.

The evolutionary process seems more stable when evolving for complexity, which is the most significant measure in this context.

The rate of growth of complexity and other measures during the evolutionary process varies depending on the various parameter settings. Therefore a dramatic change in the network topology may be the result of just a modest change in the fitness score.

The choice of parameters can also affect the time required for converging to a stable

solution, as it may make the fitness landscape rougher. For example we know that in certain cases the system will converge to a solution with only reciprocal connections, but this process can take a few hundred generations in the best case or several thousand in the worst case depending on the scaling imposed. Another example is optimal networks which have high reciprocal connections and high clustering. Varying the parameters it is possible to have systems which evolve both these characteristics at the same time or we could have a system which first evolves reciprocal connections and then increases its clustering.

It is nearly impossible to perform an exhaustive number of simulations to explore a wide range of parameters, as the evolutionary processes take a very long time.

Even when the final results, the converged solutions, look the same, except of course for the numerical values of complexity which depend on the parameters used, one may wonder how the differences in parameters may affect the numerical values of complexity when complexity is just used for evaluating a few networks and determining which one has the highest complexity.

With the correct covariance calculation setting the diagonal of the covariance matrix to a constant is not considered. An enhanced version which combines this function with the function used to normalise the diagonal was implemented. It was quite slow, as it uses two nested iterative processes. In the end this version was discarded for three reasons. The first is that using the correct covariance calculation the values in the covariance matrix are smaller and therefore the variance values used previously, 0.015 and 0.01, would actually require the self-connections to be positive. The second is that without the covariance normalisation the diagonal terms are already very similar and it seemed that the role of self-connections was just to guarantee uniform values on the covariance matrix diagonal. The third, and most important, reason is that self-connections play a role in the complexity value, as illustrated in Fig. 1.2, therefore their introduction does not seem recommended.

The approximate complexity C* does not make use of a covariance matrix so it is still unclear if the inhibitory self connections are needed. It could be possible to calculate the covariance anyway to determine the self connection values and use the adjusted connection matrix as input for the complexity calculation.

It is not clear if increasing the number of evolution cycles may lead to a better convergence when varying parameters. It is also unclear what approach should be taken with regard to weighted networks or networks with negative weights. Ultimately the topology of a network giving rise to high complexity should be consistent independently of the size of network and its number of connections. If weights vary it is very possible that network topology which optimises complexity will not be unique but rather will depend on the connection weights.

4.2 Reciprocal connections and complexity

In various papers [17,18,19,22,31] the importance of reciprocal connections has been stressed because of neurobiological implications.

The cat cortex and macaque visual cortex have 74% and 77% reciprocal connections respectively. The link the authors seem to fail to make is that complexity depends mostly on reciprocal connections: a network cannot have high complexity unless it has mostly reciprocal connections. Undirected random networks (i.e. with only

reciprocal connections) have fairly high complexity, higher than the cortical matrices (see section 3.2).

The problem seems to be that the erroneous covariance calculations assign an excessive importance to reciprocal connections [1,2]. The correct covariance calculation does not require reciprocal connections any more: networks optimised for complexity have low reciprocal connections, about the same as random directed graphs, thus being quite different from actual neural data.

4.3 Null hypothesis for complexity

When comparing networks of different sizes it is not trivial to evaluate which network has the highest complexity, even when using the same normalisation.
Using directed random networks for comparison does not prove very useful as it is already known that the biological networks we try to emulate are definitely not random. Also in the case of the complexity calculated using wrong covariance we have established that reciprocal connections are crucial for high complexity and any network with few reciprocal connections is doomed to low complexity. It therefore seems a bit unfair to compare supposed high complexity networks to networks with a low amount of reciprocal connections to show that the former have high complexity. It would be better to use as a null hypothesis a random network with the same amount of reciprocal connections. It also would be interesting to use random networks which have the same indegree and outdegree distributions as the supposed high complexity networks, as the one used in [11].
The random networks used for comparison in this report have the same fixed indegree as the evolved high complexity networks where possible.

4.4 Comparing networks

Complexity varies with the number of nodes and connections. It is nearly impossible to compare networks with different numbers of nodes and connections. Using the same scaling factor for all networks makes a fully connected network have higher complexity than an evolved sparser network.
For the following comparisons the weights have been set to 0.02, which should avoid the sum of incoming connections for a node being greater than 1, while the networks considered are random and directed.
In Fig. 4.5 and 4.6 it is shown how complexity varies for a network with 32 nodes when the connections vary between 128 and 350. In Fig. 4.5 the complexity used is the original C(X) with variance 0.015, while in Fig.4.6 C(X) is calculated using the correct covariance matrix.

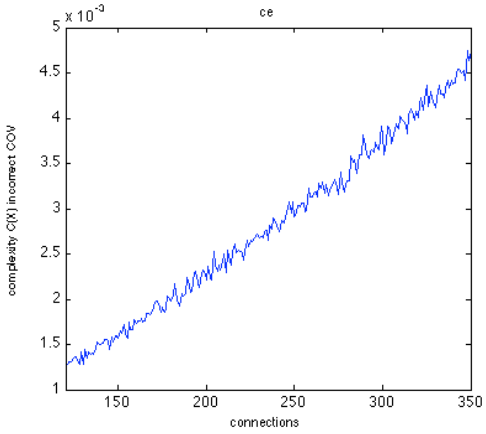

Fig.4.5 incorrect COV

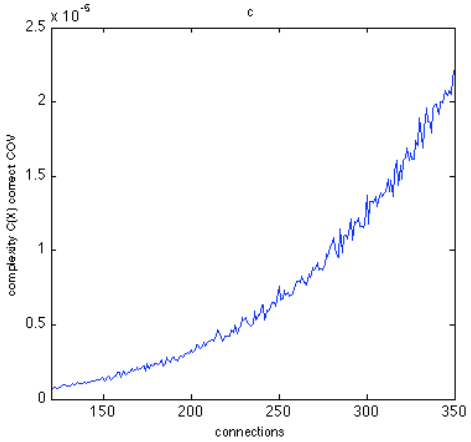

Fig. 4.6 correct COV

While the complexity increases linearly in Fig. 4.5, this is not the case in Fig. 4.6. If using $C_N(X)$ the results show a similar trend to the corresponding C(X). Also C*(X) is similar to the corresponding C(X) and $C_N(X)$.

The following results are obtained by keeping the number of connections fixed to 300, varying the number of nodes from 18 to 35 and setting the weight values to 0.02. The first two figures concern the complexity C(X) and $C_N(X)$ using the incorrect covariance calculation and variance 0.015. The behaviour of $C_N(X)$ is peculiar and was confirmed using different numbers of nodes, connections and connection weights.

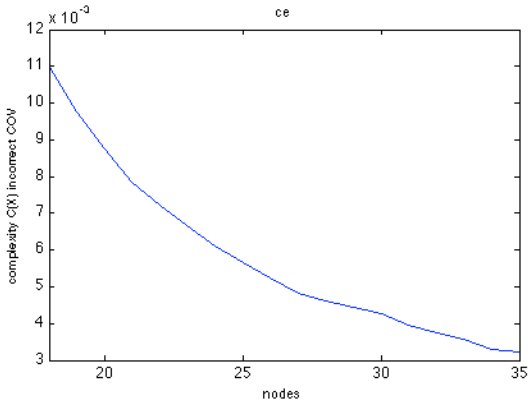

Fig. 4.7 C(X)

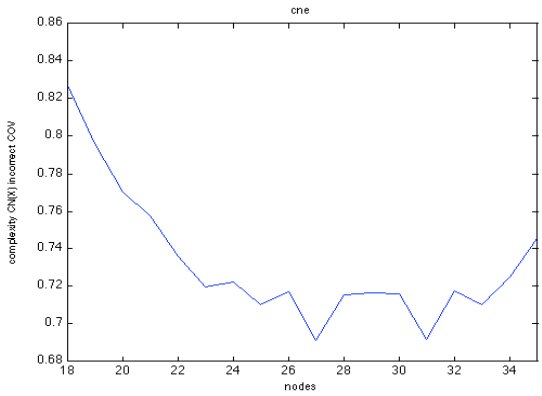

Fig. 4.8 $C_N(X)$

The following figures are obtained from C(X), $C_N(X)$ and C*(X), using the correct covariance. In this case the results are consistent.

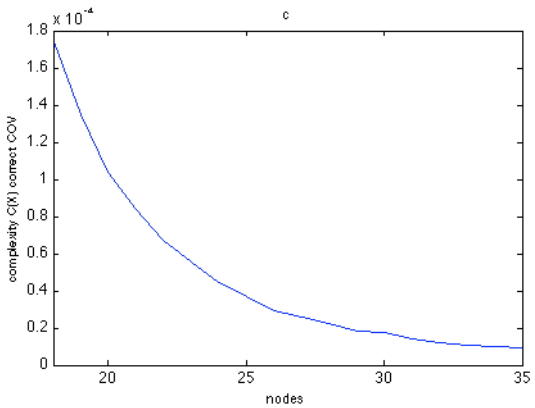

Fig. 4.9 C(X)

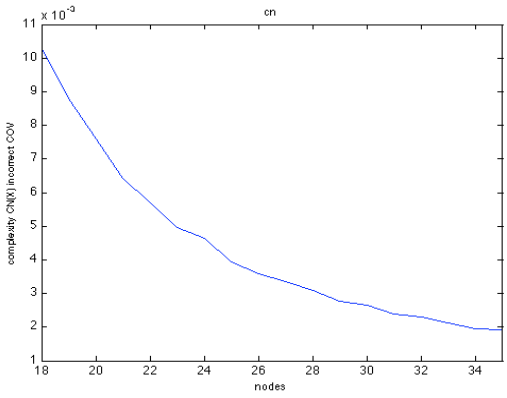

Fig. 4.10 $C_N(X)$

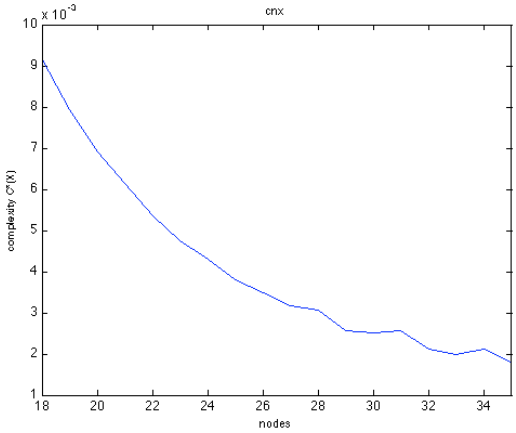

Fig. 4.11 C*(X)

The following figures show how scaled complexity changes for similar cases to those in Fig. 4.5 and 4.6 (32 nodes, 128-650 connections, weight 0.2, variance 0.015). The scaled complexity is calculated comparing a lattice to a directed random graph of the same size.

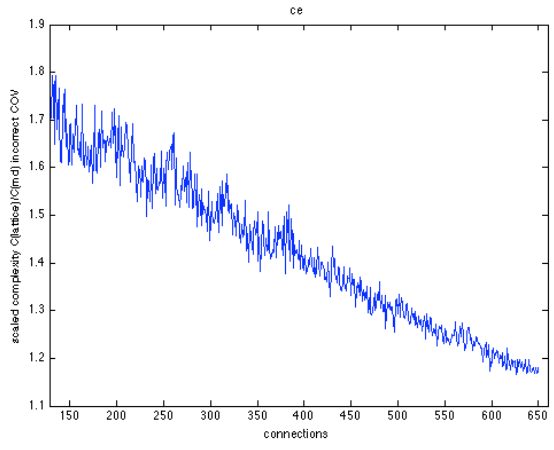

Fig. 4.12 scaled complexity incorrect COV

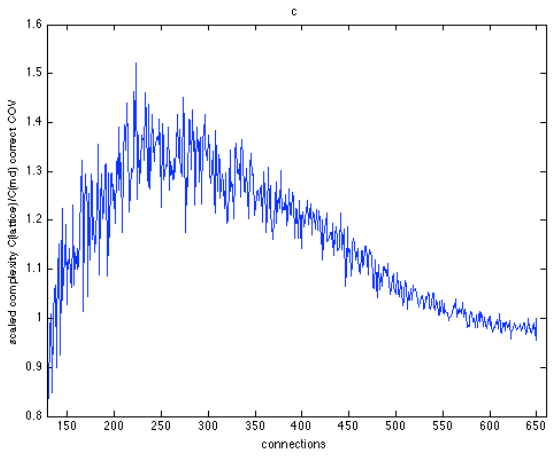

Fig. 4.13 scaled complexity correct COV

The scaled complexity changes as the number of connections increases although the trends are different, at least with the parameters used. Using the incorrect covariance calculation the scaled complexity deceases constantly and slowly. Using the correct covariance the scaled complexity shows an interesting behaviour and for the first time it seems to show complexity arising in medium density networks, as the inventors hoped.

C*(X) behaves in the same way as C(X) (with correct covariance).

Similar results are found when the number of connections is fixed (260) and the number of nodes varies (18-68):

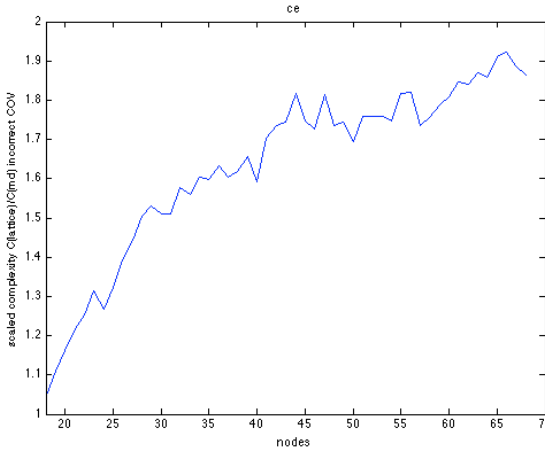

Fig. 4.14 scaled complexity incorrect COV

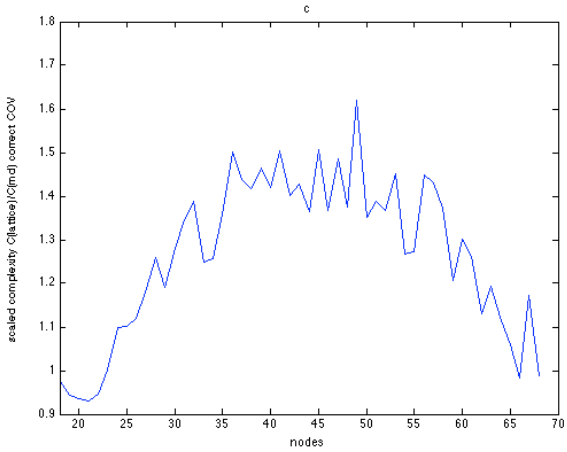

Fig. 4.15 scaled complexity correct COV

It seems that, for a lattice with 32 nodes the optimal number of connections for high complexity (using the correct covariance) is in the range 250-300.

From these initial findings it seems that comparing networks of different sizes using the scaled complexity is not feasible.

Different results for all the cases discussed in this section may be obtained using different weights or non-uniform weights.

4.5 Maximum complexity

One thing the authors fail to investigate in more detail is why and how segregated areas emerge during the evolutionary process. The number and size of densely connected subgraphs varies in various simulations [17,18,19,22]. It is observed that the size and numbers of clusters depend on the number of edges and connections. This is true but it is not the whole story. The number and size of the clusters depends also on the constraints imposed on the network. In all the various simulations there are always constraints on the number or total weights of the afferent connections in the initialization of the network. Even when the initial network is completely random its degree distribution will rarely include nodes with just very few efferent connections or a high number of afferent connections. The number of afferent connections does

not change during the evolutionary process, as the rewiring affects only the source of the connection. Therefore the case in which one node has just one incoming connection will be impossible or extremely rare, as it will be for a node to have incoming connections from all the other nodes.

Although some constraints seem reasonable from a biological plausibility point of view, the fact that other network topologies could even have higher values of complexity is not considered. Networks with high complexity are compared to random networks but not with each other.

To explore the graph space in more depth, a more generalised evolutionary algorithm was implemented. This version allowed greater freedom in the rewiring process, while still allowing control on the weight allowed (incoming and outgoing) for each connection. Rewiring can affect any of the following: source of connection, destination of connection, direction of the connection or both source and destination and also can be extended to a reciprocal connection if there is one.

Scaling/normalising when the indegree is variable may need a different approach as the weight for the same connection may vary depending on the number of incoming connections for the same node.

The evolutionary procedure evolved graphs with 32 nodes and 256 connections. The weights were set to 0.03, so even in the worst case the sum of afferent connections for a single node was less than 1. It is also to be considered if the weight scaling should depend also on the maximum number of efferent connections for a node.
The variance for Fig. 4.16 was 0.01.

In the absence of constraints, the maximum complexity is achieved by networks of the following structure:

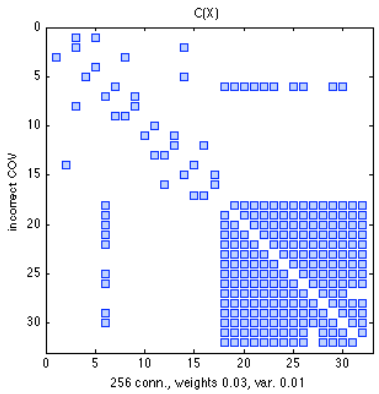

Fig. 4.16 incorrect COV

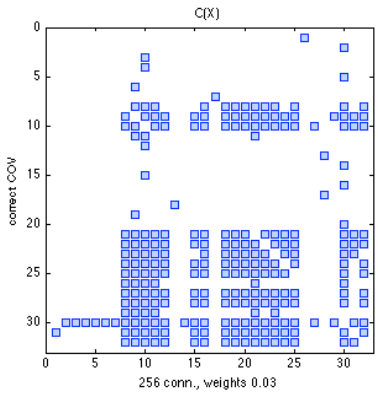

Fig. 4.17 correct COV

The network in Fig. 4.16 has path length of 5.73, clustering 0.47 and 98% reciprocal connections. The network in Fig. 4.17 has path length 2.11, clustering of 0.53 and 50% reciprocal connections. In contradiction with the networks in sections 3.1.1 and 3.1.2, this time evolving using the incorrect covariance does not lead to a small-world network, while the network evolved using the correct covariance is a small-world.

The reasons driving the formation of the clusters observed in 3.1.1 and 4.1 can now be explained. In the absence of constraints the highest complexity is achieved by a single cluster of the maximum size allowed by the number of connections.
When constraints are imposed a unique cluster cannot be formed and the evolutionary algorithm will grow multiple smaller clusters. These clusters look like squares along

the matrix diagonal (in the reordered graph) as high complexity graphs have nearly all reciprocal connections, so the afferent and efferent connections of a nodes are reciprocal. The size of the clusters is limited to the indegree values, fixed or random.

The structure of the network evolved using the correct covariance, Fig. 4.16, is not well defined. Perhaps a different reordering algorithm could show its structure more clearly.

4.6 C(X) vs $C_N(X)$

Evolving using $C_N(X)$ and C(X) may lead to slightly different results, as the rank of networks is affected, as shown in section 3.1.1.

Ultimately, evolving using $C_N(X)$ is not feasible for several reasons. The first is that $C_N(X)$ is much slower to compute than C(X). The second one is that even for smallish networks the calculation is not done on all the subsets of X but on a sample and therefore the numerical values vary slightly for the same network and we need a reliable value to use as fitness score as in later stages of the evolutionary process even a very small difference matters. Using the complexity toolbox, the maximum network size for which the complete neural complexity $C_N(X)$ calculation is possible is 12.

4.7 C(X) vs $C_N(X)$ vs C(X)*

Approximate complexity and neural complexity were used in 3.1.2 to compare evolved graphs to random networks and lattices. The results leave us with many doubts. $C_N(X)$ is an approximate measure, as it is calculated only on a sample, therefore it may be subject to imprecision and inconstancy. There seems to be a rank problem between C*(X) and $C_N(X)$ similar to the one noticed in 4.6. The rate of growth of C(X), C*(X) and $C_N(X)$ is unconvincing and deserves more attention, which is outside the scope of this report.
A network evolved to optimise C*(X) was presented in 3.1.2. It was similar to the network evolved for C(X) but with a much shorter average path.

5 Relating complexity to network topology

5.1 Small world networks and complexity

It has been shown in section 3.1.1 that networks evolved for high complexity (using the erroneous covariance calculation) show small world attributes. Their characteristic path length is short, like that of a similar random graph, while the clustering coefficient is high, like a lattice with the same size. Comparisons with small world networks (Watts and Strogatz model, rewired lattice) have also been made and the small world attributes compared, which matched.

Networks evolved for complexity using the correct covariance calculation have high clustering, while the average path length is very long for C(X) and moderately long for C*(X) , about half way between a lattice and a rewired lattice, so it seem that they do not possess fully small world characteristics (see section 3.1.2).

It has also been shown that complexity in lattices falls if the lattice is rewired (3.1.1 and 3.1.2).

In [4] it was shown that rewiring lattices makes the complexity fall monotonically. Similarly in [18] it was shown that rewiring cortical matrices decreased complexity proportionally to the rewiring rate.

Repeating the same experiment, using both the incorrect and correct covariance calculations, confirm the above findings. But while complexity keeps falling when using the incorrect covariance, using the correct covariance the complexity flattens out if the rewiring rate is greater than 50%. The connection weights are set to 0.3, while the variance for the networks in Fig. 5.1 is set to 0.01.

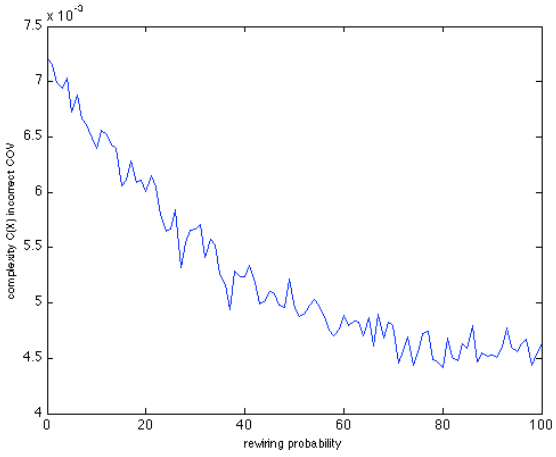

Fig. 5.1

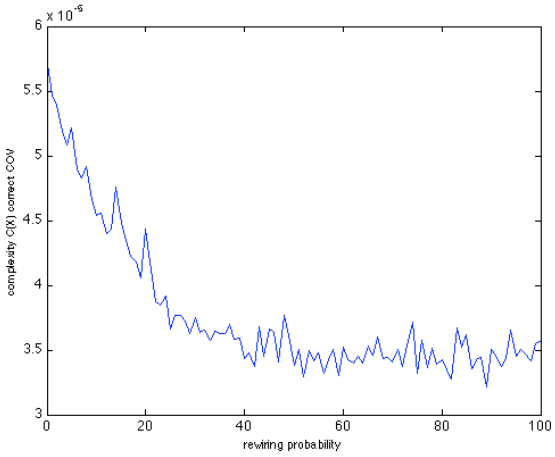

Fig. 5.2

In the small world network model the initial lattice has high clustering and highish path length, depending on the size of the network and the degree of each node. The purpose of the rewiring is to maintain the clustering while reducing the characteristic path length by introducing a few long range connections. So the clustering is that of the original lattice and can only decrease with random rewiring.

One important thing is that complexity, calculated with wrong covariance relies

enormously on reciprocal connections: networks evolved for high complexity have nearly 100% reciprocal connections. Therefore the rewiring process will heavily affect the reciprocal connections, breaking them. Secondarily rewiring affects the clustered structure of the network as described above.

The clustering coefficient is defined as average therefore it does not say much about the variety of structure in a network. It also does not capture the structure of high complex networks which present dense segregated areas. It is also questionable if the way the clustering coefficient is calculated for directed graphs is appropriate in the present context.

Alternative clustering measures exist and could be considered in further studies.

5.2 Complexity and motifs

In [16,17,18,19,20,26] it was observed that networks with high complexity have a high number of short re-entrant circuits/motifs. Unsurprisingly these networks have also a high number of motifs with reciprocal connections.

Motif frequency analysis also gives information about the clustering coefficient of a graph when considering motifs of size 3: the number of triangles is related to the clustering coefficient. It also relates to the amount of cycles of size 3 present in a graph. Networks evolved for entropy in [18] have few cycles of size 3 and a low clustering coefficient.

[1,2] also relate neural complexity, calculated using the correct covariance, to the abundance of motifs of certain types, as illustrated in section 1.5.1.

5.3 Models of high complexity graphs

Using the results obtained in 4.5 it was possible to create a model for high complexity networks according to C(X) and the wrong covariance calculation. In this graph there is a fully connected subgraph of the maximum size allowed by the total number of connections, while the nodes not in the cluster are connected loosely. (Fig. 5.4) Similarly a graph structure with multiple clusters can be considered to be compared to evolved networks with fixed indegree.

A model for a high complexity graph, using the correct covariance calculation, is mostly bipartite and has very low clustering (0.02) (Fig 5.4).

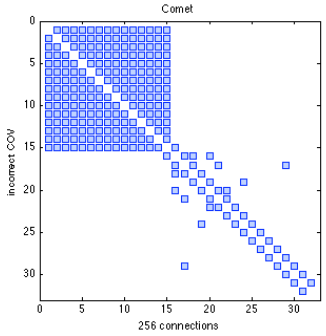

Fig. 5.3

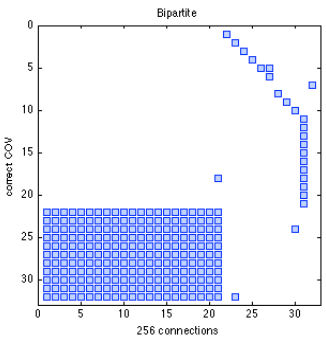

Fig. 5.4

6 Conclusion

The aim of this project has changed during its implementation for two reasons. The first was that from initial findings it was discovered that weight and variance normalisation had a strong impact on the results of the evolutionary procedures. The second reason was the discovery half-way through the year that current research had found an error in the way complexity was calculated rendering all past results moot.

The role of normalisation is still unclear and even the results published in [18] and [22] seem to have been affected. Preliminary results using the correct covariance calculation seem to suggest that the complexity calculation is more robust as the evolved networks do not change structurally when changing the normalisation.

But the numerical values of complexity change according to the normalisation used. Therefore it seems infeasible to compare complexity values obtained using different normalisation methods. It is still unclear if and how networks of different sizes (nodes and/or connections) could be compared, as it would seem that some measure of scaled complexity should be used. It is also not clear if it makes sense to compare networks of different sizes.

As far as the original aim is concerned, it has been shown that although small-world networks built using the Watts-Strogatz model (rewired lattices) have higher complexity compared to random directed networks, the original non-rewired lattices have higher complexity. This is valid for both the erroneous and correct covariance calculation.

High complexity networks may or may not show small-world attributes depending on the normalisation used and constraints imposed. Also the definition of clustering coefficient for directed graphs used in this report is questionable and other possibilities should be considered.

The constraints, such as the maximum number of incoming connections, which are often used in papers, may be justifiable by biological reasons but restrict the range of networks which can be evolved. Networks not corresponding to these specifications can have higher complexity as has been shown.

7 References and Bibliography

7.1 References

1. Barnett L., Buckley C.L. and Bullock S., 2008, On a graph theoretic interpretation of neural complexity, in preparation
2. Barnett L., Buckley C.L. and Bullock S., 2008, An approximate measure for neural complexity, in preparation
3. Boccaletti, S. et al, 2005, Complex networks: Structure and dynamics, Physics Reports
4. Buckley, C. and Bullock, S., 2007, Spatial embedding and complexity: The small-world is not enough.In Proceedings of the 9th European Conference on Artificial Life (ed. F. Almeida e Costa), pp. 986-995. Lisbon: Springer.
5. Christensen C and Albert R., 2006, Using graph concepts to understand the organization of complex systems, eprint arXiv:q-bio/0609036
6. da F. Costa, L., 2007, Characterization of Complex Networks: A Survey of measurements,
7. De Lucia, M. et al, 2005, Topological approach to neural complexity, Physical Review.
8. Girvan, M. and Newman, M.E.J., 2001, Community structure in social and biological networks, PNAS
9. Latera, V., Marchiori, M., 2001, Efficient Behaviour of Small-World Networks,
10. Leicht, EA, Newman, MEJ, 2008, Community structure in directed networks,Physical Review Letters, vol. 100, Issue 11
11. Milo, R., 2002, Network Motifs: Simple Building Blocks of Complex Networks, Science 298
12. Newman, M.E.J., 2003, The Structure and Function of Complex Networks, SIAM Review
13. Newman, M.E.J., 2006, Modularity and commnunity structure in networks, PNAS vol. 103 | no. 23 | 8577-8582
14. Seth, A.K. and Edelman, G.M. ,2004,. Theoretical neuroanatomy: Analyzing the structure and dynamics of networks in the brain.In /Complex Networks. /E. Ben-Naim, H. Fraunfelder, & Z. Toroczkai (eds).p.483-511, Springer-Verlag, Berlin.
15. Seth, A.K. (in press). Causal networks in simulated neural systems./ Cognitive Neurodynamics
16. Sporns O, Chialvo DR, Kaiser M, Hilgetag CC. Organization, development and function of complex brain networks. Trends Cogn Sci. 2004 Sep;8(9):418-25.
17. Sporns,O. et al.,2002, Theoretical neuroanatomy and connectivity of the cerebral cortex Behavioural Brain Research 135, 69-74
18. Sporns, O., Tononi, G. & Edelman, G. M., 2000, Theoretical neuroanatomy: Relating anatomical and functional connectivity in graphs and cortical connection matrices. Cerebral Cortex 10, 127-141
19. Sporns, O., 2000, Connectivity and Complexity: the relationship between

neuroanatomy and brain dynamics, Neural Networks
20. Sporns, O., and Zwi,J.,2004, The small world of the cerebral cortex.
21. Sporns, O. (2004) Complex neural dynamics. In: Coordination Dynamics: Issues and Trends, Jirsa, V.K. and Kelso, J.A.S., (eds.),pp. 197-215, Springer-Verlag, Berlin. Neuroinformatics 2, 145-162.
22. Sporns, O., Tononi, G., 2001, Classes of Network Connectivity and Dynamics, Complexity
23. Sporns, O., 2003, Network Analysis, Complexity and Brain Function, Complexity
24. Sporns, O., Graph Theory Methods for the Analysis of Neural Connectivity Patterns, Sporns' web site
25. Sporns, O., 2006, Small-world connectivity, motif composition and complexity of fractal neuronal connections, BioSystems
26. Sporns, O. 2004B Motifs in Brain networks, PLoS Biology
27. Tononi, G et al, 1998, Complexity and coherency: integrating information in the brain, Trends in Cognitive Sciences
28. Tononi, G. et al, 1999, Measures of degeneracy and redundancy in biological networks, Proc. Natl. Acad. Sci USA
29. Tononi, G. et al, 1996, A complexity measure for selective matching of signals by the brain, Proc. Natl. Acad. Sci USA
30. Tononi, G. et al, 1998, Functional Clustering: Identifying Strongly Interactive Brain Regions in Neuroimaging Data, Neuroimage
31. Tononi, G. et al, 1994, A measure for brain complexity: Relating functional segregation and integration in the nervous system, Proc. Natl. Acad. Sci USA
32. Tononi, G., 2003, Measuring information integration, BMC Neuroscience
33. Watts DJ, Strogatz SH,1998, Collective dynamics of 'small-world' networks. Nature 393:440-442

7.2 Bibliography

34. Six Degrees: The New Science of Networks by Duncan J. Watts
35. The Structure and Dynamics of Networks by M Newman
36. Small Worlds The Dynamics of Networks between Order and Randomness by DJ Watts
37. Small World: Uncovering Nature's Hidden Networks by Mark Buchanan
38. Information Theory, Inference and Learning Algorithms by David J. C. MacKay
39. Graph Theory, An Introductory Course, Bela Bollobas
40. Random Graphs, Bela Bollobas
41. Past project: Mikkel Vestergaard (2006-07)

8 APPENDICES

8.1 Project Log

Autumn term
The autumn term was spent mostly doing background reading. The MATLAB toolbox by Sporns was tested and a first model of high complexity network was implemented (weeks 9 and 10).
Meetings with supervisor in weeks 2, 4 and 9.

Christmas break
Replicated Buckley and Bullock findings.
Completed experiments with high complexity networks.
Implemented evolutionary algorithms.
Reorganised bibliography.

Spring term
Extensive search of high complexity networks through evolutionary algorithm using various parameters.
Related complexity to small-word index and vice-versa.
Tried evolutionary algorithm to improve SW index and evaluate variation in complexity.
Repeated all the above using the correct covariance calculation.
Wrote draft final report.
Meeting with supervisor in weeks 2,5 and 9. Also with L. Barnett in weeks 5,9 and C. Buckley in week 9.

Easter break
Collected statistics and data. Ran evolutionary procedures.
Checked for new papers on the topic.
Wrote final report.

Summer term
Completed final report.

8.2 Source code description

8.2.1 Evolutionary procedure

The evolutionary algorithms implemented are similar to those used in the original papers.

The number of nodes and connections does not change during the evolutionary process which therefore involves only the placement of connections.

The initial population can be any of random graphs, random graphs with a specific number of indegree or copy of a graph passed as input. The latter is useful to evolve a specific graph, suitably designed for optimality or obtained in other ways, e.g. previous evolutionary procedure or cortical matrix.

The property to optimise, complexity, neural complexity, integration or entropy is specified as a parameter, which also allows to specify if the measure is to be calculated using the erroneous or correct covariance.

The population size is specified as a parameter, although its importance is marginal as the reproduction process does not use crossover, which is unsuitable in this case since we want the connection and node numbers to remain the same.

At each iteration the population is replaced in the following way. The individual with the highest fitness score is selected and copied to the new population, while two thirds of the remaining individuals in the new population are mutated versions of the best individual and one third are mutated versions of a random element of the population. The mutation consists in the rewiring of a very limited number of connections. The amount of rewiring depends on a parameter, the rewiring rate. Each connection is considered in turn and if a generated random number is less than the rewiring rate the edge is rewired. For the same rewiring rate the number of connections rewired will depend on the number of total edges in the graph. For example if the graph has 200 connections and the rewiring rate is 0.01, it is expect that 2 edges will be rewired. In the paper considered the number of rewired edges is low, from 1 to 3.

The number of generations is usually 2000.

The evolutionary procedure is composed of the following programs:

8.2.1.1 evolvePop

The main evolutionary algorithm is called evolvePop and takes as parameters:

1. the initial population
2. the number of generations
3. the rewiring probability
4. the complexity type

5. the maximum weight(or degree) for each node
6. the type of rewiring
7. the weight to be assigned to connections for normalisation
8. the variance value for the covariance matrix

The parameters 4,7 and 8 are used for to calculate the complexity (or entropy or integration) value, which constitutes the fitness function.
The procedure returns the best element of the last generation.

8.2.1.2 evolveClJ

Takes as input a connection matrix as well as the parameters 2-8 described above and initialises the population with the specified number of copies of the input connection matrix before calling the main evolutionary procedure. It is used to start the evolutionary procedure from a non-random network, e.g. a network previously evolved or a lattice.

8.2.1.3 evolveRand

Takes as input a number of nodes and connections as well as the parameters 2-8 specified above. The population is formed of random directed graphs which are checked to be strongly connected. The program then invokes the main evolutionary produre.

8.2.1.4 evolveRandInmax

As evolveRand, but it uses random directed graphs with fixed indegree.

8.2.2 Rewiring

Rewiring consists in moving the origin or destination of a connection and it is used for the evolutionary procedure and also for turning a lattice in a small-world network.

The rewiring can take different forms, depending on a parameter passed to the function. In general it involves one edge of a connection. In the original small world model, graphs were undirected hence it did not matter which edge of a connection was rewired. In our case the network is directed, so it does matter which end of a connection is rewired. In [18,22] the source of the connection was rewired, thus preserving the nodes' indegree.

To explore in more depth the graph space it is possible a more generalised rewiring process is necessary. This version allows greater freedom in the rewiring process, while still allowing control on the weight allowed (incoming and outgoing) for each connection. Rewiring can affect any of the following: direction of the connection, source of connection, destination of connection, direction of the connection, both source and destination and also can be extended to a reciprocal connections if there is one.

For each edge in the network a random number is generated. If this number is less

than the rewiring rate the edge is rewired. The type of rewiring is specified by a parameter. If the rewiring type allows multiple possibilities the exact type will be chosen at random among the alternatives listed above.

After an edge is rewired the resulting graph must be still strongly connected. If this is not the case the change is discarded. To simplify the choice of the rewiring rate it is also possible to specify a minimum and maximum of edges to be rewired. Using a minimum of one rewired edge is important when using a low rewiring rate or using particularly stringent constraints. If the requested number of rewired edges is not achieved after considering all the edges, the rewiring process will restart from the beginning. The maximum rewiring amount will make the rewiring process end as soon as the required number of edges is rewired.

To optimise the randomness of the rewiring process the initial edge, that is the row and column of the connection matrix, is chosen at random. The other edges will be considered sequentially and when the end of the matrix is reached it will start again from row 1, column 1, until the whole matrix is considered.

The same rewiring function, using a different, usually much higher rewiring rate is used to obtain small world networks from lattices.

8.2.2.1 rewire12

Takes as input:
1. connection matrix
2. rewiring probability
3. minimum number of connections to be rewired
4. maximum number of connections to be rewired
5. maximum indegree/weight
6. maximum outdegree/weight
7. rewiring type

8.2.2.2 rewireS

Simplified version of rewire 12. Rewires the incoming connection according to the specified rewiring rate.

8.2.2.3 complexSW

Rewires the input network using rewiring rates from 0.01 to 1 and plots complexity.

8.2.3 Complexity calculation

The functions to calculate the complexity, entropy and integration values were provided by external sources.
A generalised interface which takes as inputs the connection matrix, the complexity type, the weight and variance normalisation values is called **ccomplex.** The connection matrix weights are normalised, according to the value of a parameter to a fixed value, to 0.8 divided by the number of connections or to 0.8 divided by the

number of the incoming connections for that node. According to the type of complexity and the variance normalisation the covariance matrix is calculated using one of four different external functions. Then the required complexity or entropy or integration value is calculated using different functions.

The complexity types are:

1. **ce** for C(X) using the original, erroneous covariance calculation
2. **c** for C(X) with correct covariance calculation
3. **he** for H(X), wrong covariance
4. **h** for H(X), correct covariance
5. **ie** for I(X), wrong covariance
6. **i** for I(X), correct covariance
7. **cne** for $C_N(X)$ wrong covariance
8. **cn** for $C_N(X)$ correct covariance
9. **cnx** for C*(X) approximate complexity, correct covariance

8.2.4 Community structure

The connection matrix figures in this report have been obtained reordering the connection matrix using a community structure algorithm described in [10,12,13]. Nodes belonging to the same community share many more connections among themselves than with the rest of the network. The algorithm recursively partitions the graph or subgraph into two groups until no more convenient way of separation is possible or until the subgroup is under a certain size.

A matrix of the same size of the adjacency matrix is initialised with:

$$B_{ij} = A_{ij} - \frac{indegree_i outdegree_j}{m}$$

The matrix B is added to its transpose to get a symmetric matrix and then the eigenvalues and eigenvectors of the resulting matrix are calculated. The original graph or subgraph is partitioned according to the sign of the corresponding element of the eigenvector of greatest eigenvalue. The process is repeated recursively on each partition until no more convenient partitions can be found.

The function **dispComm** takes in input a connection matrix, orders it using the community structure algorithm and the function **sortCIJv** and displays the connection matrix. The function **sortCIJv** reorders the input matrix according to the values contained in an input vector.

8.2.5 Network models

8.2.5.1 makeRandInmax

Creates a random directed network with the specified indegree

8.2.5.2 makeSunxxx and makecometCIJ

Create networks with the largest possible fully connected component.

8.2.5.3 makeBip

Create a (nearly) bipartite graph of the specified size

8.2.6 Other programs

8.2.6.1 measureCIJ and measure2

Receives a connection matrix and calls various graph theoretical measure functions (path length, clustering etc.) and displays the results. The function **measure2** in addition prints complexity measures.

8.2.6.2 compareCIJ and compareCIJr

Calculate the complexity for a range of networks varying the number of node while keeping constant the number of connections, or vice versa, and plot results. CompareCIJr sclaes the complexity to that of a random network.

8.3 Source code

8.3.1 Evolutionary procedures

8.3.1.1 evolvePop

```
function fitnessScore = fitnessF(CIJ, fitnessType, wei, vari)

% calculate the fitness score for the individual (connection matrix) using
% the specified fitness function

    fitnessScore = ccomplex(CIJ, fitnessType, wei, vari);
end


function CIJ = evolvePop(population,generations,prob, complexType,maxWeight,both, wei,
vari)

% evolve a poplation of popsize networks with N nodes and K edges for
% generations generations

N=size(population,2);
K=sum(sum(population(:,:,1)));
popsize= size(population,3);

% initialize population fitness
fitness = zeros (popsize,1);
for i=1:popsize
        fitness(i) = fitnessF(population (:,:,i),complexType, wei, vari);
end

disp '======'
fitness
disp '======'

prevc=0;
prevp=zeros(N);
for (e=1:generations)

    [maxcompl,indmax] = max(fitness);
    population (:,:,1) = population (:,:,indmax);
    fitness(1) = maxcompl;
    for (i=2:ceil(popsize/3*2))
        network = rewire12(population (:,:,1),prob,1,9,maxWeight,maxWeight,both);
        population (:,:,i) = network(:,:);
        fitness(i) = fitnessF(network,complexType, wei, vari);
    end
    indrand = ceil(rand*ceil(popsize/3*2));
    randnet = population (:,:,indrand);
    for (i=ceil(popsize/3*2)+1:popsize)
        network = rewire12(randnet,prob,1,9,maxWeight,maxWeight,both);
        population (:,:,i) = network(:,:);
        fitness(i) = fitnessF(network,complexType, wei, vari);
    end
     [maxcompl,indmax] = max(fitness);
    if (mod(e,200) == 0)

        e
        maxcompl
        indmax
        fitness

        if (maxcompl > prevc + 0.00000000000001 || maxcompl < 0)
            prevc = maxcompl;

            dispComm(population (:,:,1));
            measureCIJ(population (:,:,1))
        end
```

```
    end
end
[maxcompl,indmax] = max(fitness);
maxcompl

CIJ = population (:,:,indmax);
```

8.3.1.2 evolveCIJ

```
function CIJ = evolveCIJ(CIJ,popsize,generations,prob,
       complexType,maxWeight,both,wei,vari)

% evolve a poplation of popsize networks with N nodes and K edges for
% generations generations

% initialize population
N=length(CIJ);
K=sum(sum(CIJ));
population = zeros(N,N,popsize);

for i=1:popsize
    population(:,:,i) = CIJ;
end
CIJ=evolvePop(population, generations, prob, complexType,maxWeight,both,wei,vari);
```

8.3.1.3 evolveRand

```
function CIJ = evolveRand(N,K,popsize,generations,prob, complexType, maxWeight,both, wei,
vari)

% evolve a poplation of popsize networks with N nodes and K edges for
% generations generations

% initialize population
population = zeros(N,N,popsize);
maxAttempts = 1000;
i=1;
attempts=0;
while (i<=popsize && attempts < maxAttempts)
    network = makerandCIJ(N,K);

    % discard disconnected networks
    [R,D] = breadthdist(network);
    if (sum(sum(R)) == N*N)
        population (:,:,i) = network(:,:);
        i=i+1;
    end
    attempts = attempts + 1;
end
if (attempts < maxAttempts)
    CIJ=evolvePop(population, generations, prob, complexType,maxWeight,both,
              wei, vari);
else
    CIJ=zeros(N,N);
end
```

8.3.1.4 evolveRandInmax

```
function CIJ = evolveRandInmax(N,K,popsize,epochs,prob, complexType,
                      maxWeight,both, wei, vari)
% evolve a poplation of popsize networks with N nodes and K edges for
% epochs epochs

% initialize population
population = zeros(N,N,popsize);
maxAttempts = 1000;
i=1;
attempts=0;

while (i<=popsize && attempts < maxAttempts)
```

```
    network = zeros(N);
    for z=1:N
        knode = 0;
        while (knode < maxWeight)
            node = ceil(rand*N);
            if (node ~= z && network(node,z) == 0)
                network(node,z) = 1;
                knode = knode + 1;
            end

        end
    end

    % discard disconnected networks
    [R,D] = breadthdist(network);
    if (sum(sum(R)) == N*N)
        population (:,:,i) = network(:,:);
        i=i+1;
        network;
    end
    attempts = attempts + 1;
end
if (attempts < maxAttempts)
    population;
    CIJ=evolvePop(population, epochs, prob, complexType,9999,both, wei, vari);
else
    CIJ=zeros(N,N);
end
```

8.3.2 Rewiring

8.3.2.1 rewire12

```
function [CIJ] = rewire12(CIJ,p,minrew,maxrew,maxWeightIn,maxWeightOut,both)

% inputs:
%           CIJ      connection matrix
%           p        rewiring probability
%           minrew   minimum number of rewired edges
%           maxrew   maximum number of rewired edges
%           maxWeightNode maximum value for incoming connections
%           both
%                    =0 rewire (i,j) to (x,j) 1 edge,
%                    =1 rewire 1 edge,
%                    =2 rewire reciprocal edges (if they exist)
%                    =3 probability 0.5 of rewiring both edges
% outputs:
%           CIJ      connection matrix with rewired edges
%
% Annamaria Cucinotta
% ================================================================================

krew = 0;
if (minrew <= 0)
    minrew = -1;
end
startRow = floor(rand * size(CIJ,1)) + 1;
startCol = floor(rand * size(CIJ,1)) + 1;
cycles = 0;

while ((minrew < 0 || krew < minrew) && cycles < 100)
    cycles = cycles + 1;

    i = startRow;
    krow = 0;

    while (krow < size(CIJ,1))
            krow= krow + 1;
        if (i > size(CIJ,1))
            i = 1;
        end
        j = startCol;
```

```
        kcol = 0;
        while (kcol < size(CIJ,1))
            kcol= kcol + 1;
            if (j > size(CIJ,1))
                j = 1;
            end
% rewire if the node is not already connected to everything else.
% but if a node is already connected to everything else how can we possibly
% rewire the connection?????/
% if connection reciprocal then we may rewire indirectly through the
% reciprocal connection
% the graph is strongly connected, so at least one reciprocal connection
% exists (considering the node connected to everything else in one
% direction) we can just hope tht the reciprocal connection will be rewired
% at some point but this will make rewire less effective/probable for this
% node

            weightInJ=sum(CIJ(:,j));
            weightOutI = sum(CIJ(i,:));

            if  (i~= j && CIJ(i,j) ~= 0 && (rand <= p || weightInJ > ...
                maxWeightIn || weightOutI > maxWeightOut))


                oldCIJ=CIJ;
                randN=rand;
                if (both == 0)
                    % rewire (i,j) to (x,j)
                    knode = 0;
                    for inode=1:length(CIJ)
                        if (inode ~= i && inode ~= j && CIJ(inode,j) == 0 ...
                            && sum(CIJ(inode,:)) + CIJ(i,j) <= maxWeightOut)
                            knode = knode + 1;
                            listNodes(knode) = inode;
                        end
                    end
                    if (knode > 0) % possible source nodes found
                        node = listNodes(ceil(rand * knode));
                        CIJ(node,j) = CIJ(i,j);
                        CIJ(i,j) = 0;
                    end

                else
                    if (weightInJ > maxWeightIn || randN < 0.3) % rewire (i,j) to (i,x)

                        knode = 0;
                        for inode=1:length(CIJ)
                            if (inode ~= i && inode ~= j && CIJ(i,inode) == 0 ...
                                    && sum(CIJ(:,inode,:)) + CIJ(i,j) <= maxWeightIn)
                                knode = knode + 1;
                                listNodes(knode) = inode;
                            end
                        end
                        if (knode > 0) % possible target nodes found
                            node = listNodes(ceil(rand * knode));
                            CIJ(i,node) = CIJ(i,j);
                            CIJ(i,j) = 0;
                             % check reciprocal connection
                             % rewire reciprocal only if it is not alreaady there
                            bothc = 'ix one';
                            if  ((both==2 || (both==3 && rand > 0.5)) && ...
                                    CIJ(j,i) ~= 0 && CIJ(node,i) == 0 && ...
                                    sum(CIJ(node,:)) + CIJ(j,i) <= maxWeightOut)

                                CIJ(node,i) = CIJ(j,i);
                                CIJ(j,i) = 0;
                                bothc = 'ix two';
                            else
                            end

                        end
                    else
                        if (randN < 0.6 || weightOutI > maxWeightOut)
                    % rewire (i,j) to (x,j)
                            knode = 0;
```

```
                for inode=1:length(CIJ)
                    if (inode ~= i && inode ~= j && ...
                            CIJ(inode,j) == 0 && ...
                            sum(CIJ(inode,:)) + CIJ(i,j) <= maxWeightOut)
                        knode = knode + 1;
                        listNodes(knode) = inode;
                    end
                end
                if (knode > 0) % possible source nodes found
                    node = listNodes(ceil(rand * knode));

                    CIJ(node,j) = CIJ(i,j);
                    CIJ(i,j) = 0;
                     % check reciprocal connection
                     % rewire reciprocal only if it is not alreaady there

                    bothc = 'xj one';
                    if  ((both==2 || (both==3 && rand > 0.5)) && CIJ(j,i) ...
                            ~= 0 && CIJ(j,node) == 0 && sum(CIJ(:,node)) ...
                            + CIJ(j,i) <= maxWeightIn)

                         CIJ(j,node) = CIJ(j,i);
                         CIJ(j,i) = 0;
                         bothc = 'xj two';
                     else
                     end

                end


            else
                if (randN < 0.8)
                    % rewire (i,j) to (x,y)
                    knode = 0;
                    for inode=1:length(CIJ)
                        if (inode ~= i && inode ~= j && CIJ(i,inode) ...
                                == 0 && sum(CIJ(:,inode,:)) + CIJ(i,j) <=
                                maxWeightIn)
                            knode = knode + 1;
                            listNodes(knode) = inode;
                        end
                    end
                    if (knode > 0) % possible target nodes found
                        node = listNodes(ceil(rand * knode));
                    else
                        node = j;
                    end
                    knode = 0;
                    for inode=1:length(CIJ)
                        if (inode ~= i && inode ~= j && inode ~= node && ...
                                CIJ(inode,node) == 0 && sum(CIJ(inode,:)) ...
                                + CIJ(i,j) <= maxWeightOut)
                            knode = knode + 1;
                            listNodes(knode) = inode;
                        end
                    end

                    if (knode > 0) % possible source nodes found
                        node2 = listNodes(ceil(rand * knode));
                        cijnodenode2= CIJ(node2,node);
                        CIJ(node2,node) = CIJ(i,j);
                        CIJ(i,j) = 0;
                        % check reciprocal connection
                        % rewire reciprocal only if it is not alreaady there
                        bothc = 'xy one';
                        if  ((both==2 || (both==3 && rand > 0.5)) &&
                                CIJ(j,i) ...
                                ~= 0 && CIJ(node,node2) == 0 &&
sum(CIJ(node,:))
                                + CIJ(j,i) <= maxWeightOut && (node == j || ...
                                sum(CIJ(:,node2)) + CIJ(j,i) <= maxWeightIn))

                            CIJ(node,node2) = CIJ(j,i);
                            CIJ(j,i) = 0;
                            bothc = 'xy two';
```

```
                                    else
                                    end

                                %end


                                end
                            else
                                % rewire (i,j) to (j,i)
                                if (CIJ(j,i) == 0 && sum(CIJ(j,:)) + CIJ(i,j) <= ...
                                        maxWeightOut && sum(CIJ(:,i)) + CIJ(i,j) <=
                                            maxWeightIn)

                                    CIJ(j,i) = CIJ(i,j);
                                    CIJ(i,j) = 0;
                                end

                            end
                        end
                    end
                end
                [R,D] = breadthdist(CIJ);
                if (sum(sum(R)) < size(CIJ,1)*size(CIJ,1))
 %disp 'disconnected graph '
                    CIJ=oldCIJ;

                else

                    krew = krew + 1;
                    if (krew >= maxrew)
                         return
                    end
%                   else
%                       disp 'cannot rewire'
                end

            end
            j = j + 1;

        end
        i = i + 1;

    end
    if (minrew < 0)
        minrew = 0;
    end
end
```

8.3.2.2 rewireS

```
function [CIJ,krew] = rewireS(CIJ,p)

% rewire the input network accoding to the specified rewiring rate
% inputs:
%           CIJ      connection matrix
%           p        rewiring probability

% output:
%           CIJ      connection matrix with rewired edges
%           krew     number of rewired connections
%
% ============================================================================


initCIJ = CIJ;
for i = 1:length(CIJ)
    for j=1:length(CIJ)
        rrand = rand;
        if  (CIJ(i,j) ~= 0)
        knod = knod + 1;
        end
        if  (initCIJ(i,j) ~= 0 && CIJ(i,j) ~= 0 && rrand <= p)

            oldCIJ=CIJ;
```

```
            % rewire (i,j) to (x,j)
            inr = 0;
            for nr=1:length(CIJ)
                if (nr ~= i && nr ~= j && CIJ(nr,j) == 0)
                    inr = inr + 1;
                    listNodes(inr) = nr;
                end
            end

            if (inr > 0) % target node found
                node = listNodes(ceil(rand * inr));
                CIJ(node,j) = CIJ(i,j);
                CIJ(i,j) = 0;

            end
            [R,D] = breadthdist(CIJ);
            if (sum(sum(R)) < size(CIJ,1)*size(CIJ,1))
                % disp 'disconnected graph '
                CIJ=oldCIJ;
            else
                krew = krew + 1;
            end
        end

    end
end
```

8.3.2.3 complexSW

```
function complexSW(CIJ, complexType, wei, vari)
%
% rewire the input networks using rewiring rate 0-100 and plot complexity

    compval= zeros(101,1);
    rCIJ=CIJ;
    for i=1:101
        compval(i) = ccomplex(rCIJ, complexType, wei, vari);
        if i < 101
            [rCIJ,krew]  = rewireS(CIJ, 0.01 * i);
        end
%         if (mod(i,10)==0)
%             dispComm(rCIJ);
%         end
    end
    figure
    plot(0:100,compval)
end
```

8.3.3 Complexity

```
function complexity = ccomplex(CIJ, complexType, wei, vari)
%-------------------------------------------------------
% Calculate complexity, entropy or integration.
% Parameters:
%   CIJ connection matrix
%
%   complexType complexity type:
%       ce complexity wrong COV
%       cne neural complexity wrong COV
%       he entropy wrong COV
%       ie integration wrong COV
%       c complexity correct COV
%       cn neural complexity correct COV
%       cnx neural complexity C* correct COV
%       h entropy correct COV
%       i integration correct COV
%
%   wei weight normalisation
%       if > 0 wei value as constant weight
```

```
%       if = -1 weight = 0.8/incomig connections
%       if = -2 weight = 0.8/N
%-------------------------------------------------------

N = length(CIJ);
R = 0.1*eye(N);

if (wei == -1)
    CIJn = CIJ.*wei;
    weights=ones(1,N);
    weights = weights*0.8;
    weights = weights./(sum(CIJ>0));
    for inx=1:N

       CIJn(:,inx) = CIJ(:,inx).*weights(1,inx);
    end

else
    if (wei == -2)
         CIJn = CIJ.* 0.8/N;
    else

         CIJn = CIJ.*wei;
    end
end
if (strcmp(complexType,'cnx'))
    complexity = calcCNapprox(CIJn, true);
    return
end
if (strcmp(complexType,'ce') || strcmp(complexType,'cne') || ...
    strcmp(complexType,'ie') || strcmp(complexType,'he') )
    if (vari == 0)

        [COV,COR] = calcCOV00(CIJn,R);
    else
        [COV,its] = seth_calcCOV3(CIJn,vari);
    end
else
    if (vari == 0)
        [COV,COR] = calcCOVcorrect(CIJn,R,1000,false);
    else
        [COV,its] = calcCOVcorrectVD(CIJn,vari);
    end
end

if (strcmp(complexType, 'c') || strcmp(complexType,'ce'))
     C_alt = calcC_det(COV);
     complexity = C_alt;
else
    if (strcmp(complexType, 'cn') || strcmp(complexType,'cne'))
        [C_N,I_lvl,I_lvl_max] = calcC_Nint(COV,1000);
        complexity = C_N;
    else
        if (strcmp(complexType,'i') || strcmp(complexType,'ie'))
            C_i = calcI_det(COV);
            complexity = C_i;

        else
            C_h = calcH_det(COV);
            complexity = C_h;
        end
    end
end
end
```

8.3.4 Community structure

```
function [CIJcomm,nnn,nodelist] = communityCIJ(CIJ,partsize,nnn,nodelist,thisnodes)
% community structure algorithm from Leicht and Newman (2007)
%
% divide recursevely the input connection matrix into two partitions
 %
```

```
%   CIJ connection matrix
%   partsize minimum partition size
%   nnn
%   nodelist
%   thisnodes

    CIJcomm=CIJ;
    if (size(nodelist) == [1 1]) % initialise
        nodelist = zeros(3,length(CIJ));
        nodelist(1,1:end) = 1:length(CIJ);
        nodelist(2,1:end) = 1;
        nodelist(3,1:end) = 1:length(CIJ);
        thisnodes = 1:length(CIJ);
    end

    if (length(CIJ) < partsize)
%         disp '--------------------------------------return'
%         CIJ
%         disp '--------------------------------------return'
        return;
    end
    [id,od,deg] = degrees(CIJ);
    for i=1:length(CIJ)
        for j=1:length(CIJ)
            B(i,j) = CIJ(i,j) - id(i)*od(j)/length(CIJ);
        end
    end
    BB = B + B';
    [eigvect,eigval]=eig(BB);
    eigval;
    eigvect;

    maxeig = eigval(end);
    if (eigval(end) > 0.000000001)
        ipart1=0;
        ipart2=0;
        for i=1:length(BB)
            if (eigvect(i,end) >= 0)
                ipart1 = ipart1 + 1;
                part1(ipart1) = i;
                nodelist(2,thisnodes(i)) = nnn;
                thisnodes1(ipart1)=thisnodes(i);
                nodelist(3,thisnodes(i)) = nnn*100+ipart1;
            else
                ipart2 = ipart2 + 1;
                part2(ipart2) = i;
                nodelist(2,thisnodes(i)) = nnn + 1;
                thisnodes2(ipart2)=thisnodes(i);
                nodelist(3,thisnodes(i)) = (nnn + 1)*100 +ipart2;
            end
        end

        for i=1:length(part1)
            for j=1:length(part1)
                part1CIJ (i,j) = CIJ(part1(i), part1(j));

            end
        end
        for i=1:length(part2)
            for j=1:length(part2)
                part2CIJ (i,j) = CIJ(part2(i), part2(j));
            end
        end
        [P1,nnn,nodelist]=communityCIJ(part1CIJ, partsize,nnn,nodelist,thisnodes1);
        [P2,nnn,nodelist]=communityCIJ(part2CIJ, partsize,nnn+1,nodelist,thisnodes2);
        P1;
        P2;
        CIJcomm=zeros(length(BB),length(BB));
        CIJcomm(1:length(P1),1:length(P1))=P1;
        CIJcomm(length(P1)+1:length(P2)+length(P1),length(P1)+1:length(P2)+length(P1))=P2;

    else
        %disp 'no more partitions'
        CIJcomm = CIJ;
        part1=1:length(BB);
```

```
        part2=part1;
    end

    function newCIJ = addCIJ(partCIJ,nodelist,node,CIJ)
        nodelist;
        node;

        newCIJ= partCIJ;
        for iii=1:length(nodelist)
            newCIJ(length(partCIJ)+1,iii) = CIJ(node,nodelist(iii));
            newCIJ(iii,length(partCIJ)+1) = CIJ(nodelist(iii),node);

        end
    end
end



function v3=dispComm(CIJ)
% display community structure

    [a,b,c]=communityCIJ(CIJ,4,1,0,0);

    v3=sortCIJv(CIJ,c);
    figure;
    spy(v3,'s',12);

end


function CIJ = sortCIJv(CIJ,values)
% sort CIJ according to the vector values

[mm,oo]=sort(values(end,:));


for i=1:size(CIJ,1)
    if (i ~= oo(i)) % swap rows and cols
       oldCIJ = CIJ;
       copyXv = CIJ (i,:);
       copyXh = CIJ (:,i);
       copyYv = CIJ (oo(i),:);
       copyYh = CIJ (:,oo(i));
       CIJ(i,:) = copyYv(:);
       CIJ(:,i) = copyYh(:);
       CIJ (oo(i),:) = copyXv(:);
       CIJ(:,oo(i)) = copyXh(:);
       CIJ(i,i) = copyYh(oo(i));
       CIJ(i,oo(i)) = copyYv(i);
       CIJ(oo(i),oo(i)) = copyXv(i);
       CIJ(oo(i),i) = copyXv(oo(i));
       oo(find(oo==i)) = oo(i);
    end
end
```

8.3.5 Network models

8.3.5.1 makeRandInmax

```
function CIJ = makeRandInmax(N,K,maxWeight)
    maxAttempts = 1000;

    i=1;
    attempts=0;
    maxWeight;
    N;

    while (i<=1 && attempts < maxAttempts)
        network = zeros(N);
        for z=1:N
```

```
            knode = 0;
            while (knode < maxWeight)
                node = ceil(rand*N);
                if (node ~= z && network(node,z) == 0)
                    network(node,z) = 1;
                    knode = knode + 1;
                end

            end
        end
        if (sum(diag(network)) > 0)
            network
        end
        %network = makerandCIJ_nondir(N,floor(K/2));
         % discard disconnected networks
        [R,D] = breadthdist(network);
        if (sum(sum(R)) == N*N)

            i=i+1;
            network;
        end
        attempts = attempts + 1;
    end
    if (attempts < maxAttempts)

        CIJ=network;
    else
        CIJ=zeros(N,N);
    end
end
```

8.3.5.2 makesun4CIJ and makecometCIJ

```
function [CIJ] = makesun4CIJ(N,K)

% inputs:
%           N        number of vertices
%           K        number of edges
% outputs:
%           CIJ      connection matrix
%
% makes a graph with the largest fully connected components while other
% nodes are connected to one node of the cluster
% Annamaria Cucinotta
% ================================================================================

% initialize
%disp 'makesun4CIJ'
CIJ = zeros(N);
delta = 9-4*(2*N-K);
if (delta <= 0)
    centreSize = 0;
else
    centreSize = floor((3 + sqrt(delta))/2)
    if (centreSize > N)
       centreSize = N;
    end
    CIJ(1:centreSize,1:centreSize) = 1;
    for (i=1:centreSize)
        CIJ(i,i) = 0;
    end
end

if (centreSize < N)
    KK = centreSize*(centreSize-1); % number of vertices added to the matrix
    node = centreSize;
    centralN = 1;

    while (KK+1<K);
        node = node + 1;
        if (node > N)
            centralN = centralN + 1;
```

```
            node = centreSize + 1;
        end
        if (node ~= centralN)
           CIJ(node,centralN) = 1;
           CIJ(centralN,node) = 1;
           KK= KK+2;
        end
        %sum(sum(CIJ));
    end;
    if (KK < K)
        node = node + 1;
        if (node > N)
            centralN = centralN + 1;
            node = centreSize + 1;
        end
        CIJ(node,centralN) = 1;

        KK= KK+1;
        sum(sum(CIJ));
    end
end



function [CIJ] = makecometCIJ(N,K)

% inputs:
%           N        number of vertices
%           K        number of edges
% outputs:
%           CIJ      connection matrix
%
% makes CIJ matrix, with size = N,K. The final graph as a 'comet' shape:
% there is a fully connected subgraphs of the maximum possible size, while
% other nodes form a tail
% Annamaria Cucinotta
% ================================================================================

% initialize
%disp 'makecometCIJ'
CIJ = zeros(N);
delta = 9-4*(2*N-K);
if (delta < 0)
    delta = 0;
end

centreSize = floor((3 + sqrt(delta))/2);

CIJ(1:centreSize,1:centreSize) = 1;
for (i=1:centreSize) % no self connections diagonal
    CIJ(i,i) = 0;
end
KK = centreSize*(centreSize-1); % number of vertices added to the matrix

% connect remaining nodes to the central component as a tail
node = centreSize + 1;

while (KK<K & node <= N );
    CIJ(node,node-1) = 1;
    CIJ(node-1,node) = 1;
    KK= KK+2;
    sum(sum(CIJ));
    node = node + 1;
end;


% connect remaining nodes to random nodes in the tail

node = centreSize + 1;
while (KK<K);
    node;
    if (node > N)
        node = centreSize + 1;
    end;
    centralN = ceil(rand*(N-centreSize)) + centreSize;
```

```
    if ( CIJ(node,centralN) == 0)
        CIJ(node,centralN) = 1;
        CIJ(centralN,node) = 1;
        KK= KK+2;
    end
    sum(sum(CIJ));
    node = node + 1;
end;
```

8.3.5.3 makeBip

```
function [CIJ] = makeBip(N,K)

% inputs:
%           N        number of vertices
%           K        number of edges
% outputs:
%           CIJ      connection matrix
%
% makes a bipartite graph
% Annamaria Cucinotta
% ================================================================================

%disp 'makesun6CIJ'
CIJ = zeros(N);
delta = (1-N)*(1-N) - 4*(K-N)
if (delta <= 0)
    left = N-1;
else
    left = floor((N-1 - sqrt(delta))/2);
    if (left > N)
       left = N-1;
    end
end

    CIJ(N-left+1:N,1:N-left) = 1;
    for (i=1:left)
        CIJ(i,i+N-left) = 1;
    end
    for (i=left+1:N-left)
        CIJ(i,N) = 1;
    end

    for (i=1:N)
         CIJ(i,i) = 0;
    end
    edges = K-sum(sum(CIJ));

    while edges > 0
        node1 = ceil(rand*N);
        node2 = ceil(rand*N);
        if (node1 ~= node2 && CIJ(node1,node2) == 0)
            CIJ(node1,node2) = 1;
            edges = edges - 1;
        end
    end
end
```

8.3.6 Other programs

8.3.6.1 measureCIJ

```
function [labels,meas]=measureCIJ(CIJ)

% display statistical measures
```

```
    [R,D] = breadthdist(CIJ);
    [lambda,ecc,radius,diameter] = charpath(D);
    [RCIJ,rhoG] = reciprocal(CIJ);
    [gamma,gammaG] = clustcoef(CIJ);

    meas(2)=lambda;
    labels(:,2) = 'path        ' ;
    meas(1)=diameter;
    labels(:,1) = 'diam        ';
    CIJr= reshape (CIJ,[size(CIJ,1)*size(CIJ,1) 1]);
    if rhoG == Inf
        meas(3) = 1;
    else
        meas(3)=rhoG;
    end
    labels(:,3) = 'reciprocal  ';
    if (gammaG == 0)
        gamma
    end
    meas(4)=gammaG;
    labels(:,4) = 'clustering  ';

    [id,od,deg] = degrees(CIJ);

    meas (5) = min(id);
    labels(:,5) = 'min in deg  ';
    labels(:,6) = 'max in deg  ';
    labels(:,7) = 'std in deg  ';
    labels(:,8) = 'min out deg ';
    labels(:,9) = 'max out deg ';
    labels(:,10) = 'std out deg ';

    meas (6) = max(id);
    meas (7) = std(id);
    meas (8) = min(od);
    meas (9) = max(od);
    meas (10) = std(od);
    labels=labels';
    size(labels);
    size(meas);
    ns=num2str(meas(:,:)');
    size(ns);
    labels(:,end+1:end+size(ns,2)) = ns;

end
```

8.3.6.2 measure2

```
function measure2(netw, type, wei, vari)
%-------------------------------------------------
% display graph metrics and complexity values
%
%   netw    input network
%   type    1-oldCOV 2-bothCOV 3-newCOV
%   wei     weigth normalisation
%   vari    variance normalisation


    measureCIJ(netw)
    if type < 3
        complexity_ce=ccomplex(netw,'ce', wei, vari)
        complexity_cne=ccomplex(netw,'cne', wei, vari)
        complexity_he=ccomplex(netw,'he', wei, vari)
        complexity_ie=ccomplex(netw,'ie', wei, vari)
    end
    if type > 1
        complexity_c=ccomplex(netw,'c', wei, vari)
        complexity_cn=ccomplex(netw,'cn', wei, vari)
        complexity_cnx=ccomplex(netw,'cnx', wei, vari)
        complexity_h=ccomplex(netw,'h', wei, vari)
        complexity_i=ccomplex(netw,'i', wei, vari)
    end
```

```
end
```

8.3.6.3 compareCIJ

```
function compareCIJ(N1,N2,K1,K2,complexType,wei,vari)
% compare network complexity varying connections or nodes

    if (N1==N2) % vary connections

        ccomp = zeros(K2,1);
        for i=K1:K2
            %CIJ = makesunCIJ(N1,i);
            %CIJ = getRand(N1,i);
            %CIJ = makerandInmax(N1,i,floor(i/N1));

            CIJ = makeringlatticeCIJ(N1,i);
            ccomp(i,1) = ccomplex(CIJ,complexType,wei,vari);
        end
        figure
        plot(1:K2,ccomp)

    else % vary nodes
        ccomp = zeros(N2,1);
        for i=N1:N2
            %CIJ = makesunCIJ(i,K1);
            %CIJ = getRand(i,K1);
            CIJ = makeringlatticeCIJ(i,K1);
            %CIJ = makerandInmax(i,K1,floor(K1/i));
            ccomp(i,1) = ccomplex(CIJ,complexType,wei,vari);

        end
        figure
        plot(1:N2,ccomp)


    end
     title (complexType)

    function network=getRand(No,Ko)
        ir = 0;
        while ir < 100
            ir = ir + 1;
            network = makerandCIJ(No,Ko);
            if(sum(sum(network)) ~= Ko)
                disp 'missing edges'
            end
            [R,D] = breadthdist(network);
            if (sum(sum(R)) == No*No)
                return;
            end
        end
        disp 'cannot find connected network'
    end

end
```

8.3.6.4 compareCIJr

```
function compareCIJr(N1,N2,K1,K2,complexType,wei,vari)

% compare newtworks varying nodes or connections scaling complexity
% to random network

    if (N1==N2)  % vary connections

        ccomp = zeros(K2,1);
        for i=K1:K2
            %CIJ = makesunCIJ(N1,i);
```

```
            CIJ = getRand(N1,i);
            %CIJ = makerandInmax(N1,i,floor(i/N1));
            ccomp1 = ccomplex(CIJ,complexType,wei,vari);
            CIJ = makeringlatticeCIJ(N1,i);
            ccomp2 = ccomplex(CIJ,complexType,wei,vari);
            ccomp(i,1) = ccomp2/ccomp1;

        end

        figure
        plot(1:K2,ccomp)

    else  % vary nodes
        ccomp = zeros(N2,1);
        for i=N1:N2
            %CIJ = makesunCIJ(i,K1);
            CIJ = getRand(i,K1);

            %CIJ = makerandInmax(i,K1,floor(K1/i));
            ccomp1 = ccomplex(CIJ,complexType,wei,vari);
            CIJ = makeringlatticeCIJ(i,K1);
            ccomp2 = ccomplex(CIJ,complexType,wei,vari);
            ccomp(i,1) = ccomp2/ccomp1;
        end

        figure
        plot(1:N2,ccomp)


    end
     title (complexType)

    function network=getRand(No,Ko)
        ir = 0;
        while ir < 100
            ir = ir + 1;
            network = makerandCIJ(No,Ko);
            if(sum(sum(network)) ~= Ko)
                disp 'missing edges'
            end
            [R,D] = breadthdist(network);
            if (sum(sum(R)) == No*No)
                return;
            end
        end
        disp 'cannot find connected network'
    end

end
```