CELL ASSEMBLY ENUMERATION IN RANDOM GRAPHS*

Jordon Cavazos Khalif Halani Zachary Rubenstein Steven J. Cox

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 3.0^{\dagger}

Abstract

This report summarizes work done as part of the Computational Neuroscience PFUG under Rice University's VIGRE program. VIGRE is a program of Vertically Integrated Grants for Research and Education in the Mathematical Sciences under the direction of the National Science Foundation. A PFUG is a group of Postdocs, Faculty, Undergraduates and Graduate students formed round the study of a common problem. This module reproduces the work G. Palm "Towards a Theory of Cell Assemblies". This work was studied in the Rice University VIGRE/REU program in the Summer of 2010. This module builds an algorithm to find cell assemblies by Palm's definition and discusses some preliminary employment of that algorithm.

1 Introduction

1.1 History

Models of individual neurons vary in complexity, but in general, neurons tend to behave like this:

- A neuron is either excited or not excited.
- When excited, a neuron will stimulate neurons to which it has an outgoing connection. Otherwise, it will not stimulate other neurons.
- A neuron becomes excited when it receives a sufficient amount of stimulation from other neurons.

When modeling the brain, see seek to model the collective behavior of neurons. The fundamental type of collective behavior is, by the Donald Hebb model of the brain, the cell assembly.

First introduced by Hebb, the cell assembly is, "a diffuse structure comprising cells... capable of acting briefly as a closed system, delivering facilitation to other such systems and usually having a specific motor facilitation" [4]. A cell assembly is a particular arrangement of a group of neurons with certain properties. The most salient of these properties is that a certain fractional portion of the assembly will excite the entire assembly.

^{*}Version 1.9: Sep 29, 2010 11:50 am -0500

[†]http://creativecommons.org/licenses/by/3.0/

By Hebb's proposal, a cell assembly represents a single concept in the brain. For instance, Hebb proposes that the corner of an abstract triangle may be represented by a cell assembly [4]

Since Hebb first discussed the concept of a cell assembly, there has been some amount of biological research supporting his ideas. For instance, the work of György Buzsáki suggests that groups of cells that fire during a given time period are correlated [2].

1.2 Definitions

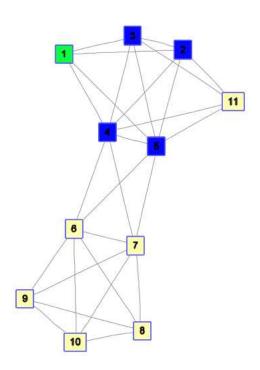


Figure 1: Node 1 (bright green) has the neighborhood {2, 3, 4, 5} (dark blue)

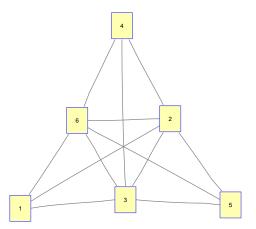


Figure 2: A k-core for which k=3

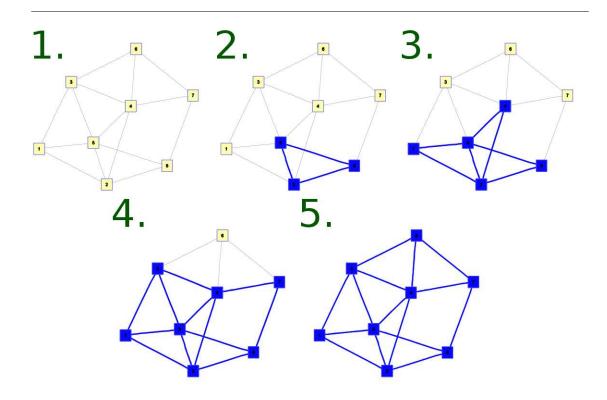


Figure 3: The process of closure for k=2: in step 2, an arbitrary 3 vertices are activated. Through each subsequent step, those vertices having at least 2 neighbors in the active set are activated in turn. After step 5, there are no more vertices to activate, so the vertices highlighted in step 5. are the closure of the vertices highlighted in step 2.

Gunther Palm defined Hebb's assembly in the concrete language of graph theory [6]. In brief, Palm's discussion constructs, or depends upon, the following definitions:

- graph: A graph G has a vertex set, V(G), and a set of edges, $E(G) \subseteq V(G) \times V(G)$. If $u, v \in V(G)$ and $uv \in E(G)$, we say that the graph G has an edge directed from the vertex u toward the vertex v. Vertices are labeled with integers by convention.
- neighborhood: We define the neighborhood of some vertex $v \in V(G)$ with respect to G, call it N(v, G), as $\{u : \exists uv \in E(G)\}$ (Figure 1).
- degree: The degree of a vertex $v \in V(G)$ with respect to G, call it D(v,G), is |N(v,G)|.
- subgraph: A graph g is a subgraph of G iff $V(g) \subseteq V(G)$ and $E(g) \subseteq E(G)$. Further, g is an induced subgraph of G iff g is a subgraph of G and $\forall e \in E(G) \cap (V(g) \times V(g))$, $e \in E(g)$.
- k-core: A subgraph g of G is a k-core iff $\forall v \in V(g), |X| \geq k$, where $X = N(v) \cap V(g)$ (Figure 2).
- minimal k-core A subgraph q of G is a minimal k-core iff it has no induced subgraphs which are k-cores.
- maximum k-core A subgraph g of G is a maximum k-core iff G contains no k-cores h for which the |V(h)| > |V(g)|.
- activation: We say that a vertex $v \in V(G)$ can be either active or inactive. We generally say that, initially, an arbitrary subset of vertices $M \subseteq V(G)$ are active and the rest inactive. We further define a

map $f_k(M,G)$: (sets of vertices, graphs) \rightarrow (sets of vertices) which performs the following operation:

- · Take a graph G, and a set of vertices, M, where $M \subseteq V(G)$.
- $\cdot \quad \forall v \in V(G)$:
 - * let $Y = M \cap N(v, G)$
 - * iff $|Y| \ge k$, then $v \in R$
- · Return R.

For convenience, we add a superscript $f_k^n(M,G)$, where $f_k^2(M,G) = f_k(f_k(M,G),G)$, $f_k^3(M,G) = f_k(f_k(M,G),G)$, etc.

- closure: We say that the closure of a set of active nodes M with respect to the graph G, call it $cl_k(M,G)$, is equal to $f_k^{\infty}(M,G)$. If $f_k^n(M,G)$ does not converge for some sufficiently large n, then the closure of M is undefined (Figure 3).
- k-tight: A k-core T, which is an induced subgraph of G, is k-tight iff it satisfies the following condition:
 - · $\forall K$ where K is an induced subgraph of T and a k-core:
 - * $cl(V(K), G) \supseteq V(T)$, or,
 - $* cl(V(T) \setminus V(K), G) = \emptyset$
- k-assembly: An induced subgraph A of G is a k-assembly iff V(A) = cl(V(T), G) where T is a k-tight induced subgraph of G.

2 Goal

We seek to understand Palm's definition of cell assembly in context. In particular, we seek to determine whether a brain-sized random graph can contain a realistic number of assemblies by Palm's definition.

In this report, we describe the process of k-assembly enumeration and explain some preliminary experimentation using that algorithm.

The report exists in two forms. The abridged version includes the material described above, while the full version goes on to include more trend examples, an extensive collection of visual examples of k-Assemblies, and implementation.

This is the abridged version. The full version is available here¹.

3 Finding k-Assemblies

Finding k-assemblies takes place in two steps: k-core enumeration and k-assembly confirmation:

3.1 k-Core Enumeration

The process of k-Core enumeration follows the general form of a branch and bound algorithm. That is, it follows the process:

- 1. If possible, solve the problem, otherwise:
- 2. Break the problem into several smaller problems, for each of these smaller problems, go to step 1.

Applied to our problem, the general algorithm looks like this, given some input graph G (Figure 4):

¹See the file at http://cnx.org/content/m34843/latest/fullreport.pdf

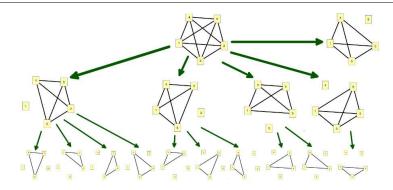


Figure 4: An exhaustive 2-core enumeration: Ultimately, every induced subgraph is enumerated except those that are not 2-cores.

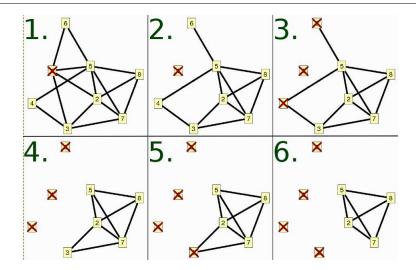


Figure 5: The maximum k-core algorithm for a 3-core: 1,2) A vertex is eliminated from the graph, leaving an induced subgraph which is not a minimal 3-core. 3,4) All vertices which have insufficient degree with respect to the new subgraph graph (less than 3 in this case) are eliminated. 5,6) Again, a vertex without sufficient degree is eliminated. The subgraph pictured in step 6 has no vertices of less than k degree, so the process is completed.

- 1. If G is a minimal k-core, stop, otherwise:
- 2. Dissect G as follows:
 - a. Find the maximum k-core that is an induced subgraph of G, call it H (Figure 5).
 - b. For each vertex $v \subset V(H)$, go to step 1, using for the new G and induced subgraph of H such that $V(G) = V(H) \setminus v$.

This process finds all k-cores in the original G. To sketch a proof:

- An algorithm that finds all induced subgraphs in G and then filters k-cores from the rest will trivially enumerate all induced k-cores in G.
- The k-core enumeration algorithm described is equivalent to such an algorithm. By in turn eliminating every vertex from a graph, the algorithm find all induced subgraphs except for those it skips. The skipped subgraphs will never generate k-cores not already generated:
 - · Skipped subgraphs have at least one vertex of degree less than k, with respect to that subgraph; call this vertex set W_1 . The subgraph may contain vertices that would have degrees less than k if excluding the vertices in W_1 ; call this set W_2 . There may also be vertices dependent on the vertices in W_1 and W_2 to maintain a degree of k; call those W_3 . We can continue forming these sets until $G \setminus (W_1 \cup W_2 \cup ... \cup W_n)$ is a k-core for some positive integer n. Define W as $(W_1 \cup V_2 \cup ... \cup W_n)$
 - · Trivially, take some induced subgraph of G called g. $\forall v \in V(g)$, $D(v,g) \leq D(v,G)$
 - · Consequently, no induced subgraph of G will contain a k-core including any vertices in W, since no vertex in W has gained degree upon finding an induced subgraph, and, consequently, the collective W still cannot meet the degree threshold to be included in a k-core.
 - · Subgraphs are skipped iff they contain vertices in W.

3.1.1 Complexity

The algorithm described above runs in approximately exponential (O(n!)) time, which is to be expected since it solves an NP-hard problem.

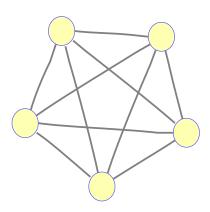


Figure 6: A clique for which k = 4

The enumeration of k-cores reduces to the NP complete clique problem [5] as follows:

- A clique is a k-core for which the number of vertices in the core is equal to k+1 (Figure 6).
- The clique problem is: determine whether an arbitrary graph contains a clique of at least size n.
- If we could accomplish k-core enumeration in polynomial time, say $O(n^k)$, where n is the number of nodes in the graph, then:
 - · We could find cores of all k in, at most, $O(n \times n^k) = O(n^{k+1})$ time, because there is no k core in a given graph for which k exceeds n.

- · Without increasing fundamental run-time, we could flag each of those k-cores which is a clique. In doing so, we find all cliques.
- · By simply finding the largest of this group of cliques, we have solved the clique problem in polynomial time.

Consequently, k-core enumeration belongs to the class of NP-hard problems, meaning that it is not clear whether an algorithm that runs significantly faster than exponential time can be devised.

That is not to say, however, that the algorithm cannot be improved upon at all. For instance, one of our implementations takes advantage of the fact that, if we form an induced subgraph H by removing some vertex v from a graph G, if v is not in a k-core, then the graph I resulting from removing any of v's neighbors that are not in a k-core from G will have the same maximum k-core as H. Also, our algorithm takes care not to revisit previously enumerated branches. Different approaches can certainly speed up an algorithm to solve the problem, but it is not clear that any approach will make the algorithm run in sub-exponential time.

3.2 Assembly Enumeration

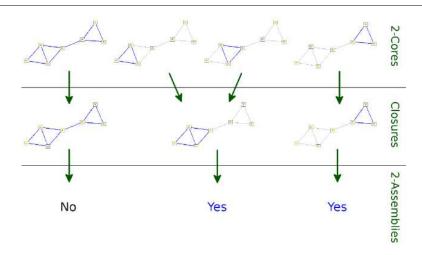


Figure 7: k-Assembly enumeration on a reduced set of k-cores.

Upon enumerating all k-cores, every unique closure of a k-core is analyzed to determine whether any one of the k-cores that closes to that closure is tight (Figure 7). Checking tightness involves nothing more clever than simply verifying the above definition of tight. If at least one of the cores that closes to a given closure is tight, then that closure is a collected into the set of k-assemblies, otherwise, it is ignored.

4 Experimentation

After devising an algorithm to find every k-assembly, we attempted to discover attributes about k-assemblies through, first, generating random graphs, and then, enumerating the assemblies contained in those graphs.

4.1 Random Graph Generation

We employed two primary types of random graphs.

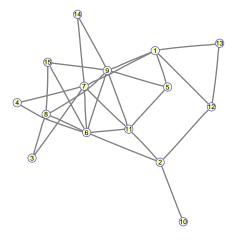


Figure 8: A Bernoulli random graph

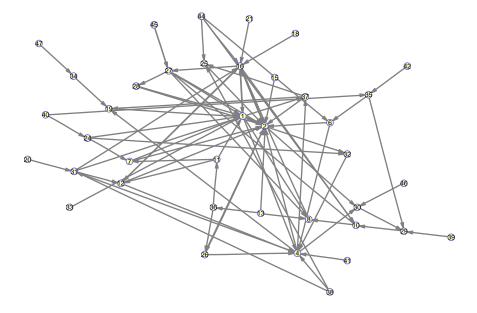


Figure 9: A scale-free Cooper-Frieze random graph

- 1. The classical Bernoulli random graph (Figure 8):
 - Pick some probability p, and a number of vertices, n. Create a graph G, for which V(G) = 1, 2, ... n.
 - For all pairs of vertices, (i, j), where $i \neq j$ and $0 < i, j \le n, \exists$ an edge $ij \in E(G)$ with probability p.
- 2. The scale-free Cooper-Frieze random graph (Figure 9) [1]. As we implement it:
 - pick some positive integer T; $\alpha, \beta, \gamma, \delta \in [0, 1]$; P, Q, which are 1-indexed lists. $P = (p_1, p_2, ..., p_n)$, where $\sum_{i=1}^n p = 1$, and $p_i \in [0, 1] \, \forall i \in \{1, 2, ..., n\}. Q = (q_1, q_2, ..., q_m)$, where $\sum_{i=1}^m q = 1$, and $p_i \in [0, 1] \, \forall i \in \{1, 2, ..., m\}$.
 - Begin with a graph G, where $V(G) = \{1\}$, and $E(G) = \{11\}$. (That is, G is a graph with a single vertex which has a single edge connected to itself).
 - $\forall t \in \{0, 1, 2, ..., T\}$:
 - Do the "Old" procedure with probability α , otherwise, do the procedure "New."
 - · Do the procedure "Add Edges."
 - *Old*:
 - · with probability δ , choose the vertex *start* from among the set of vertices in V(G) randomly, giving each vertex an even chance. Otherwise, choose *start* with the probability for each vertex proportional to the degree of that vertex with respect to G.
 - · with probability γ set the boolean variable terminateUniformly to true. Otherwise, set the variable to false.

· choose an index of Q so that the index i has a probability q_i of being chosen. Set the integer variable number Of Edges to this chosen index.

• *New*:

- · Add a new vertex v to V(G). Call that vertex start.
- · With probability β set the boolean variable terminate Uniformly to true. Otherwise, set the variable to false.
- · choose an index of P so that the index i has a probability p_i of being chosen. Set the integer variable number Of Edges to this chosen index.

Add Edges:

- · Create the set *END* by choosing *numberOfEdges* vertices from *G*. The vertices are chosen randomly, either with uniform probability if *terminateUniformly* is *true*, or in proportion to degree otherwise.
- · \forall vertices $e \in END$, add to G an edge directed from start to e.

It should be noted that throughout the book, only scale-free graphs are allowed either:

- Edges with weights greater than 1 (That is, there may exist more than one edge $uv \in E(G)$ where u and v are vertices in V(G).)
- "Loops" which connect nodes to themselves (That is, edges of the form uu.)

We allow these exceptions so that our graphs will be in full compliance with the Cooper-Frieze model.

4.2 Data

After generating a number of random graphs, we investigated a number of relationships between cell assemblies vs. other features in random graphs. We speculate on a few of these relationships here.

When we vary the probability that any given pair of vertices has an edge between them over a number of undirected, Bernoulli random graphs, we can see a number of distinct phases in the number of cell assemblies that these graphs will have, on average (Figure 10).

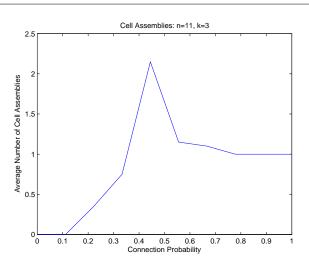


Figure 10: Bernoulli random graphs appear to undergo phase changes when varying edge probability

First, edge probability is insufficient to form even one assembly, then the number of assemblies increases to a peak as the graph becomes more dense. Past that peak, the closure of all tight sets converges to a single cell assembly. It would be interesting to find a method for predicting the probability at which this peak occurs, and then comparing that value to biology.

The Cooper-Frieze scale-free graph has no parameter to directly adjust edge density, but varying the related parameter, α , does have similar effects.

A fairly promising, although ostensibly not very accurate, indicator of the number of k-cores is the maximum eigenvalue of a graph's adjacency matrix (Figure 11).

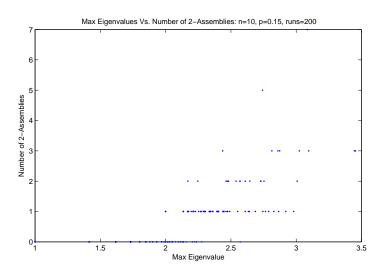


Figure 11: Maximum eigenvalues apparently have a positive correlation with number of assemblies in a Bernoulli random graph. (r = 0.7309)

On undirected Bernoulli random graphs, the two variables appear to have a nontrivial, positive correlation. However, here we see that the method of graph construction is extremely important to any such observations, since on directed scale-free graphs, we see a relationship that is not nearly as clear (Figure 12).

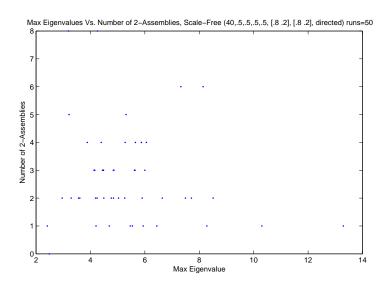


Figure 12: For Cooper-Frieze random graphs, this correlation is much more dubious. (r = -0.1315)

5 Future Work

- On commodity hardware, the k-core enumeration algorithm terminates in a few minutes on graphs on the order of 10 nodes. It can process larger graphs given a low density of edges or a large k. Consequently, it is not clear that strict enumeration will be useful on human-brain-sized graphs with approximately 100 billion neurons and 1000 connections per neuron, but it may be useful for smaller graphs. For instance, a honey bee has fewer than 1 million neurons [3].
- If strict enumeration fails even in smaller cases, it would perhaps be possible to take advantage of certain knowledge about the structure of brains in order to speed up enumeration by restricting the types of graphs that need to be enumerated. For instance, if we could safely use a bipartite model for a brain, the algorithm could likely be optimized for bipartite graphs.
- Also, cell assembly enumeration could proceed further from a statistical standpoint, as hinted at in the "Data" section. Perhaps some function could be devised that maps some more easily discerned information about a graph to the probability that that graph will contain a given number of cell assemblies.
- The definition of cell assemblies, as presented here, is fairly cumbersome. If certain aspects of the definition, such as closure, could be simplified, perhaps an easier approach to cell assemblies may become apparent.
- Cell assemblies should theoretically have far more function than simply existing as static structures in random graphs. Studying the interaction of cell assemblies and the learning processes that create cell assemblies may yield interesting insights into neuroscience in general.

6 Conclusion

Our approach to enumeration of cell assemblies in arbitrary graphs probably runs insufficiently fast to explore the problem as an end in itself. However, it does give us a useful tool to help us understand cell assemblies. We can now find an unlimited number of examples of cell assemblies which we may use as tools to explore general trends and gain insight into the structure of cell assemblies. Perhaps with enough work on the subject, we may find a viable way to understand the workings of the brain through random graph theory.

References

- [1] B[U+FFFD] Bollob[U+FFFD] nd Oliver Riordan. Mathematical results on scale-free random graphs. In Handbook of Graphs and Networks: From the Genome to the Internet, pages 1–32. Wiley, Weinheim, 2003.
- [2] Gyrgy Buzs [U+FFFD]. Rhythms of the Brain. Oxford University Press, 2006.
- [3] Eric H. Chudler. Brain facts and figures. http://faculty.washington.edu/chudler/facts.html. Retrieved 21 Jul 2010.
- [4] D. O. Hebb. The Organization of Behavior. Lawrence Erlbaum Associates, Mahwah, 1949.
- [5] Richard M. Karp, Raymond E. Miller, and James W. Thatcher. Reducibility among combinatorial problems. *The Journal of Symbolic Logic*, 40(4):85–103, 1975.
- [6] Gunther Palm. Towards a theory of cell assemblies. Biological Cybernetics, 39:181–194, 1981.