RICE UNIVERSITY

Understanding k-Assemblies down to its core

Karina Aliaga

New Jersey Institute of Technology

August 8, 2009

Abstract

Ever since psychologists and neuroscientists began studying the physiological inner workings of the brain, they have been puzzled by many questions. How are concepts stored and recalled within our brains? How does learning and memory occur? The answers to these questions are cell assemblies. *A* cell assembly is a group of neurons that are strongly connected in such manner that if a sufficient amount of neurons become activated, the entire cell assembly activates therefore allowing the process to recall a concept. In this paper, Palm's Mathematical definition of a cell assembly is extended to fit a binary integer programming problem.

I. Introduction

One of the most important functions that we, as organisms, have is our ability to store and recall information, in other words our memory. Memory allows us to learn from our mistakes, to create a recollection of our past, build an identity and to grow as individuals. Since memory plays such a crucial role in our lives then it follows that psychologists and neuroscientist have begun studies of the inner workings of the brain in order to explain how memory works.

In the mid-1900s, Donald Hebb, a Canadian psychologist, made important contributions regarding associate memory [1]. Associative memory is referring to a memory organization where the memory is accessed by its content [2]. For instance, it is believed that concepts are stored in "cell assemblies," such that if a new stimulus appears and it is similar to previously stimulus that recalls an already formed cell assembly, then this new stimulus through pattern completion will activate the entire cell assembly thus allowing for the concept to be accessed. Certainly, one of D.O. Hebb's most important postulates refers to his Hebbian theory as explained in his book "The Organization of Behavior".

According to the Hebbian Theory or cell assembly theory, "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" [3/pg62]. In other words, he believed that the strength between the synapses of the neurons could be changed with time. For instance, if the pre-synaptic neuron and the post-synaptic neuron become active approximately around the same time then the strength of the synapse will increase. Thus, from these ideas, it follows that a cell assembly is a group of neurons are strongly connected in such a manner that if a sufficient amount of neurons become activated, the entire cell assembly activates therefore allowing the process to recall a concept.

Even though, D.O. Hebb was the first psychologist to introduce the term of cell assemblies, his work has served as a platform for neuroscientists, psychologists and scientists from different fields to explore his theories of associative memory. Consequently, Günther Palm, a German mathematician, became interested in cell assemblies from a mathematical perspective in his book, "Neural Assemblies: An Alternative Approach to Artificial Intelligence,"[4]. He was able to define cell assemblies using various concepts in linear algebra and discrete mathematics as it will be described in more detail later on.

Cell Assemblies have proven to be an important concept to be explored in the path for a better understanding of the inner-workings of the brain regarding to associative memory. As a result, this summer's goal has been to study in more detail what is encompass in a cell assembly, methods for finding them and further studies. Even though, there are many scientists in addition to Günther Palm that have explored these concepts of cell assemblies, for the purpose of this summer research program, we have mainly focused on D.O. Hebb and Günther Palm's ideas as the basis for this research project.

II. Cell Assemblies: A New Perspective

As previously mentioned, a cell assembly entails a group of neurons that are strongly interconnected. Consequently, in order to visualize the connections between these neurons and develop a relation to mathematics, we use different concepts found in linear algebra.

A. Constructing a Graph

Even though a "cell assembly" refers to a collection of neurons that work together, it can be also refer as a finite network of interconnected nodes or a graph with fixed connections. However, the mathematical definition of a "cell assembly" is far more complex and requires more restrictions as you will read later on. However, it is essential to start by giving the proper definition of a graph.

Definition: Let graph $G(V,W)$ be composed of a set of edges $\{w_i\colon\ i = 1,2,\dots,m\}$ and a set of vertices$\{v_i\colon i = 1,2,\dots,n\}$.

In addition to defining a graph, the connection between the neurons or nodes needs to be specified using an adjacency matrix. An adjacency matrix is a representation of a graph which shows the connections between vertices.

Definition: Let the adjacency matrix A_d represent the connectivity between the neurons. Therefore, if $A_d(n,m) = 1$ then there exists a connection between the neurons *n* and *m*. Conversely, if $A_d(n,m) = 0$ then there is no connectivity between the neurons *n* and *m*.

For Example:

An example of a graph is given by Fig. 1 since it is composed of 4 vertices whose nodes are all interconnected. It's corresponding adjacency matrix, A_d, is showed below where each location contains a binary element explaining the presence or absence of a connection between the nodes or neurons.

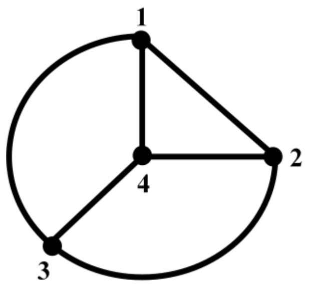

Fig. 1

$$\longrightarrow \qquad A_d = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Fig. 1 is examples of an undirected graph since there are no arrows between the nodes specifying one directional path. In addition, for the purpose of this summer's research project, I will be only be referring to directed graphs with no loops.

B. Threshold

Given graph $G(V, W)$, threshold is denoted as the minimum number of inputs each node receives in order to become excited.

C. Mapping Function

G. Palm uses a graph as a simple model to show the flow of activity in the brain [4].

Definition: Given graph $G(V, W)$ and $c(u, v)$, the weight of the edge from vertex u to v, the resulting flow of subset, *C,* is obtained by using threshold, *k*, based on the mapping function, $e(C, k)$:

$$e: \mathcal{P}(V) \to \mathcal{P}(V)$$

$$(C, k) \to C'$$

$$e(C, k) = \left\{ v \; e V: \sum_{u \; e \mathrm{C}} c(u, v) \geq k \right\}$$

In other words, given graph $G(V, W)$, if a subset C is activated, then other nodes in graph $G(V, W)$, will become activated as long as they overcome threshold k and so forth.

For Example:

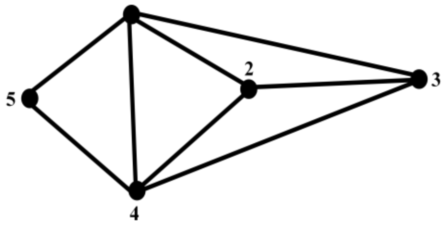

Fig. 2a

Fig. 2a is an example of graph $G(V, W)$ where $V = \{1,2,3,4,5\}$. If subset, $C = \{1,4,5\}$ is activated then the mapping flow can be observed as follows in Fig 2b, Fig 2c and Fig. 2d.

Example of the Mapping Function:

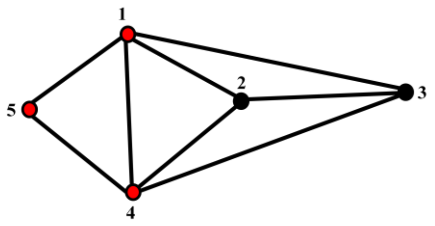

Fig. 2b

Fig. 2b shows the activation of subset, $C = \{1,4,5\}$, for the given graph $G(V, W)$.

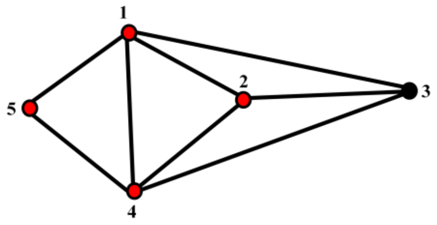

Fig. 2c

Given a threshold of $k = 2$, subset C activates node 2, since it receives 2 inputs from nodes 1 and 4, therefore:

$$e^1(C,k) = \{1,2,4,5\}$$

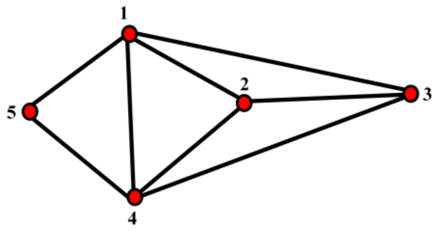

Fig. 2d

If the mapping function, $e(C,k)$, applied again to $e^1(C,k)$ then node 3 becomes activated since it receives at least 2 inputs from nodes 1, 2 and 4. Consequently,

$$e^2(C,k) = \{1,2,3,4,5\}$$

According to G. Palm, in addition to understanding the mapping function, there are a few other definitions that must be understood before defining a cell assembly in a mathematical perspective as it will be described in the next section.

D. *Invariant Sets*

A trivial definition required for describing a cell assembly is the notion of an invariant set.

Definition: A subset C of $G(V,W)$ is called invariant if $e^n(C,k) = e^{n-1}(C,k)$.

Figures 2a through 2d illustrate the activity flow of a group of activated neurons using the mapping function. It should be observed that in Fig. 2d all the neurons become activated after applying $e(C,k)$ at the second iteration.

However, if $e^3(C,k)$ is applied, nodes 1 through 5 will become activated once again, therefore $e^3(C,k) = e^2(C,k)$ which is an example of an invariant set. For this particular example, by activating subset $C = \{1,4,5\}$, the entire graph became activated. Nonetheless, this might not be the case for other graphs.

E. Closure

Definition: Given a subset C of $G(V,W)$, the closure of C denoted by $cl(C)$, is the invariant set generated by C.

For instance, the closure of C in Fig. 2a is the entire graph, $cl(C) = \{1,2,3,4,5\}$.

F. Persistent Sets

According to G. Palm, a persistent set can be defined using the mapping function as follows:

Definition: A subset C of $G(V,W)$ is persistent if $e(C) \supseteq C$.

For Example:

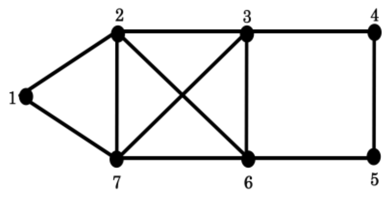

Fig. 3

In Fig. 3 subset $C = \{1,2,7\}$ is a persistent set of $G(V,E)$ since if set C becomes activated for a threshold of $k = 2$ then $e(C) = \{1,2,3,6,7\}$ which satisfies the following:

$$e(C) \supseteq C$$

Definition: A subset C of $G(V,W)$ is minimal persistent if it contains no proper subsets that are persistent.

For Example:

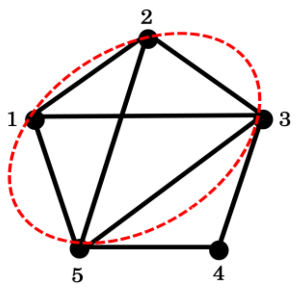

Fig. 4

An example of a minimal persistent set is observed in Fig. 4. The subset $C = \{1,2,3,5\}$ is minimal persistent for threshold $k = 3$ since every node receives 3 inputs and contains no proper subsets that are persistent.

According to G. Palm in his paper, "Towards a Theory of Cell Assemblies," he defines a cell assembly as *the closure of a tight set.* However, the definition of a *tight* set is very complex and encompasses finding a plethora of persistent sets in the process. Nonetheless, G. Palm also states that a ***minimal persistent*** set is also ***tight*** set therefore this paper will primarily focus on *tight* sets produced from minimal persistent sets [4].

III. Research Goal

As mentioned previously, a cell assembly is a group of neurons that are interconnected in such a manner that when a sufficiently large portion of the cell assembly becomes activated, the entire cell assembly activates. G. Palm used the definition of a cell assembly and described it in a mathematical perspective. According to G. Palm, a cell assembly is the closure of a tight set. Since G. Palm provided a mathematical definition of a cell assembly then theoretically it should be possible to find all the cell assemblies for a given network. However, due to the complexity of the mathematical definition of a cell assembly and the time allowed for this summer research program, the search for cell assemblies was narrowed to studying k-assemblies. A k-assembly can be defined using G. Palm's terms as the closure of a tight set where the tight set is produced only from minimal persistent sets. Consequently, the purpose of this summer research program was to study k-assemblies and develop algorithms for finding all the k-assemblies within a given network.

IV. k-Assemblies: Is graph theory the solution?

Social Network Analysis (SNA) studies the interaction between individuals, organizations and other units. Similarly to a cell assembly, a social network is represented using a graph where the nodes symbolize the individual or organization and the edge connecting them symbolizes the relationship among them [5]. Graph theorists interested in Social Network Analysis use different terms found in graph theory such as k-cores to describe this particular network. Fortunately, by studying k-cores, a connection to k-assemblies was established.

A. An introduction to k-cores

According to Seidman, a k-core is a subgraph with minimum degree, $\delta(G)$, at least k [5]. By the definition of a k-core, it is clearly observed that k-cores are persistent sets as defined by G. Palm which are a key element in finding cell assemblies. However, in the process of finding cell assemblies, an algorithm for finding the largest k-core was implement as a starting point.

Definition: Given $G(V,W)$, $C \subseteq V$ is a k-core if $|N(v) \cap C| \geq k$ where $N(v)$ is the set of neighbors of a vertex $v \in V$ [5].

Definition: $C \subseteq V$ is the largest k-core in $G(V,W)$ if no nodes outside subgraph, C, are at least k.

Algorithm for Finding the Largest k-cores

A greedy algorithm can be used to find the largest k-core in a given graph in polynomial time as described below:

1. First, given a graph, $G(V,W)$, pick a vertex v and determine its degree, $\delta(v)$, which is the number of inputs vertex v receives.
2. If $\delta(v) \geq k$ for every $v \in V$, then the largest k-core was found.
3. If $\delta(v) < k$, then node v is deleted.

   Repeat the process for $G \leftarrow G - v$ until every node is checked.

MATLAB Pseudocode

By using the principles behind the greedy algorithm, an algorithm for finding the largest k-core was developed using the software called MATLAB as described below:

1. First, the adjacency matrix, A, of graph, $G(V,W)$, must be constructed.

2. A vector, `bval`, is created to store the indices of vertices with values less than k.

```
bval=find(sum(A)<k & sum(A)>0);
```

3. While loop is created to erase vertices v found in `bval` whose degree is less than k.

```
while sum(bval)>0

        i=1;
        A(:,bval(i))=0;
        A(bval(i),:)=0;

        bval=find(sum(A)<k & sum(A)>0);

        i=i+1;

end
```

4. Function called largestKcore3 outputs the adjacency matrix, B, containing the largest k-core.

Example of Largest k-core

Fig. 5 shows an example of a graph G, containing 8 nodes respectively. If the threshold is $k = 3$, then the largest 3-core found using the MATLAB algorithm is observed in Fig. 6.

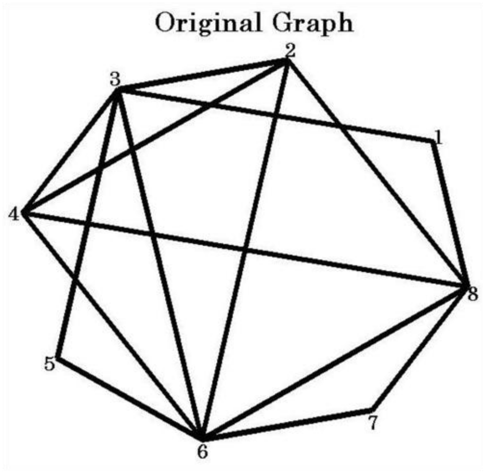

Fig. 5

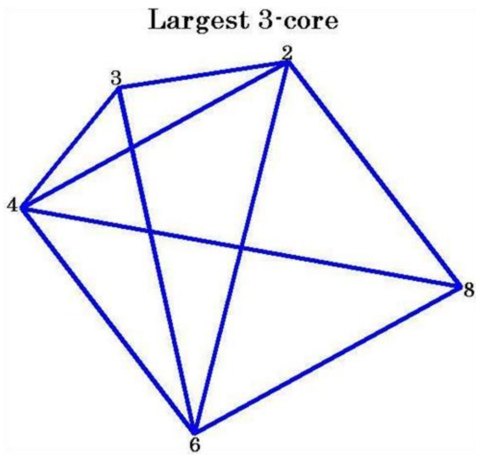

Fig. 6

As observed in Fig. 6, nodes 1, 5, and 7 do not become part of the largest 3-core since they only receive 2 inputs when the required threshold is $k = 3$.

In addition, k-cores are restricted to undirected graphs as shown in the previous examples. By finding the largest k-core, the search for a k-assembly is narrowed since it throws away “un-necessary” vertices that are not part of the assembly. However, further steps must be taken in order to find k-assemblies which the reason why minimal k-cores will be introduced next.

B. Are Minimal k-cores k-assemblies?

As mentioned before, a k-assembly is defined as the closure of a tight set where the tight set is produced **only** from minimal persistent sets. Since k-cores are persistent sets then a minimal k-core is a minimal persistent set. Consequently, a k-assembly can be defined in terms of k-cores as described below:

Definition: Subset C of $G(V,W)$ is a minimal k-core if no proper subsets of C are k-cores.

Definition: The closure of a minimal k-core is a k-assembly.

Examples of Minimal k-cores

In fig. 7, lets denote a graph, G, be composed of a set of edges V such as $V = \{1,2,3,4,5,6,7,8\}$, then the subgraph C is a minimal 3-core composed of $V' = \{2,4,6,8\}$.

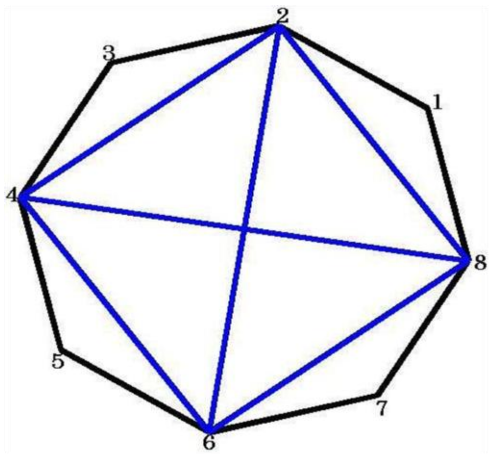

Fig. 7

In addition, subgraph C is a k-assembly for $k = 3$. For this particular example, there is only one minimal 3-core, however if the threshold is changed to $k = 2$ then there are 8 minimal 2-cores. For instance, $M = \{1,2,8\}$ is just one example of a minimal 2-core.

As the number of nodes and connectivity of a graph increases, the number of minimal 2-cores can become exponential thus it becomes a complex problem for finding all of the minimal 2-cores. Before explaining in more detail the complexity of this problem, an algorithm will be described for finding minimal k-cores.

C. *Minimal k-cores: A Binary Integer Programming Problem*

The purpose of this summer research program is to find k-assemblies, however, the first step is to find minimal k-cores. MATLAB has a built-in program called bintprog, a solver for binary integer programming problems, which has proven to be helpful in finding minimal k-cores.

bintprog Algorithm

The bintprog algorithm solves a series of linear programming relaxation problems in search for finding an optimal solution using the branch and bound method. Since in the branch and bound method, a search tree is created, bintprog finds a feasible solution to the LP problems and updates the best binary integer point found so far as the search tree grows [6].

bintprog Equation

bintprog solves the object function in the following form:

$$\begin{gathered}\min \boldsymbol{f^T} \\ \boldsymbol{Ax \leq b} \\ \boldsymbol{e_i \in \{0,1\}^n}\end{gathered}$$

bintprog Inputs

bintprog minimizes the objective function $x_1 + x_2 + \ldots x_n$ according to its input arguments f, A, b

$\boldsymbol{f}$: Represents the coefficients of the variables of the objective function

For simplicity of finding k-assemblies, f will be composed of a vector of weight 'one.' In addition, the threshold inequality must be defined in order to find the minimal k-cores and determine the inputs A, and b.

The threshold inequality is defined as follows:

$$A_d x \geq kx \ \ \rightarrow \ \ 0 \geq (kI - A_d)x$$

This inequality specifies the connections between all the neurons as well as the required threshold for a neuron to become ignited or excited.

Sine *bintprog* minimizes the objective function, $f^T x$, constrained to $Ax \leq b$, an additional constrain must be added. Clearly, if the threshold inequality would be solved as stated above, the solution obtained would have resulted in the zero vector. Consequently, the following constrain was added:

$$x_1 + x_2 + \ \dots \ x_n \geq 1$$

By simply, adjusting the inequalities to satisfy those established by *bintprog,* the input arguments A and b can be defined as follows:

$$A = \left(\frac{kI - A_d}{-1 -1 -1 \dots -1}\right) \qquad\qquad b = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}$$

***bintprog* Example:**

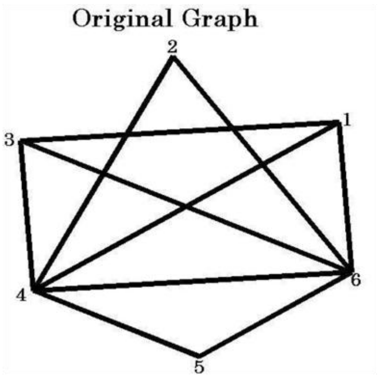

Fig. 8a

Given a threshold of $k = 3$, the graph G on Fig. 8a has a minimal 3-core. In order to find this minimal 3-core, *bintprog* must be used, thus a MATLAB function called "*minimal_k_core*" was created to facilitate the process of inputting *bintprog* arguments.

By using the MATLAB function "*minimal_k_core",* the adjacency matrix of Fig. 8a and threshold k is only required. Consequently, the function outputs a binary vector indicating the nodes that belong to the minimal k-core as follows:

$$A_d = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \rightarrow bintprog \rightarrow \; x = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \rightarrow$$

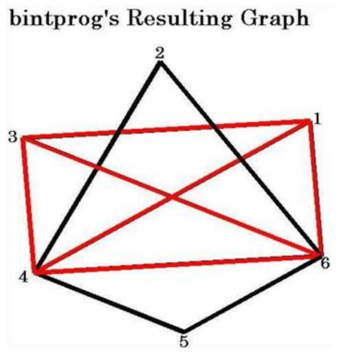

Fig. 8b

Fig. 8b shows the minimal 3-core found by *bintprog.*

V. $\boldsymbol{k}$-Assemblies

In this paper, a k-assembly is defined as the closure of a minimal k-core. Consequently, the steps to find a k-assembly are as follows:

1. First, given a graph G, construct it respective adjacency matrix, A_d.
2. Use the largest k-core algorithm if desired to narrow the search.
3. Input the adjacency matrix, A_d, and threshold, k, into the MATLAB function "*minimal_k_cores,"* which uses *bintprog* to find a minimal k-core.
4. Find the closure of the minimal k-core, thus a k-assembly.

Example:

Fig. 9 shows a graph, G, with its respective adjacency matrix, A_d, and resulting vector x indicating which nodes form the minimal 2-core.

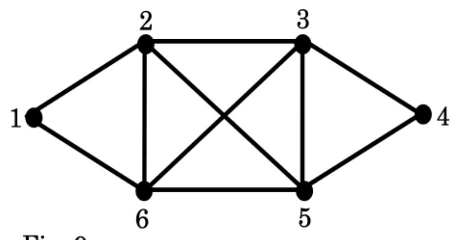

Fig. 9

$$A_d = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \qquad x = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

According to Fig. 9, the minimal 2-core found contains nodes 1, 2 and 6. However, in order to find the k-assembly or 2-assembly, the closure must be applied using the mapping function $e(x,k)$ as follows:

1. Find $e^1(x,k)$:

$$\overset{A_d}{\begin{pmatrix} 0&1&0&0&0&1\\ 1&0&1&0&1&1\\ 0&1&0&1&1&1\\ 0&0&1&0&1&0\\ 0&1&1&1&0&1\\ 1&1&1&0&1&0 \end{pmatrix}} \overset{x}{\begin{pmatrix}1\\1\\0\\0\\0\\1\end{pmatrix}} = \begin{pmatrix}2\\2\\2\\0\\2\\2\end{pmatrix} \qquad e^1(x,k) = \begin{pmatrix}1\\1\\1\\0\\1\\1\end{pmatrix}$$

2. Since $e^1(x,k) \neq x$, the mapping function must be applied again, therefore $e^2(x,k)$ is as follows:

$$\overset{A_d}{\begin{pmatrix} 0&1&0&0&0&1\\ 1&0&1&0&1&1\\ 0&1&0&1&1&1\\ 0&0&1&0&1&0\\ 0&1&1&1&0&1\\ 1&1&1&0&1&0 \end{pmatrix}} \overset{x}{\begin{pmatrix}1\\1\\1\\0\\1\\1\end{pmatrix}} = \begin{pmatrix}2\\4\\3\\2\\3\\4\end{pmatrix} \qquad e^2(x,k) = \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix}$$

By generating $e^3(x,k)$, an invariant set will be generated since the entire graph is excited, thus the entire graph is a k-assembly for threshold of $k = 2$. For understanding purposes, the example above is relatively small with a threshold that is not realistic for a neuron, since it is believe that a neuron has a threshold of $k = 10$ approximately. Nonetheless, we can find k-assemblies for networks of higher cardinality and threshold as observed in Fig. 10.

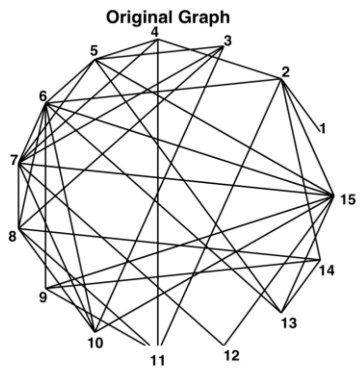

Fig. 10a

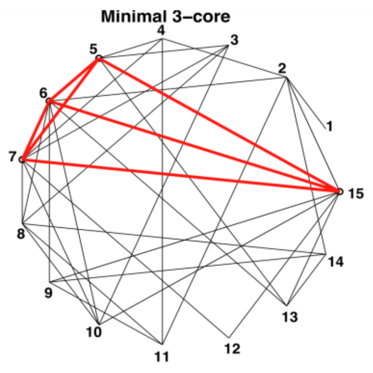

Fig. 10b

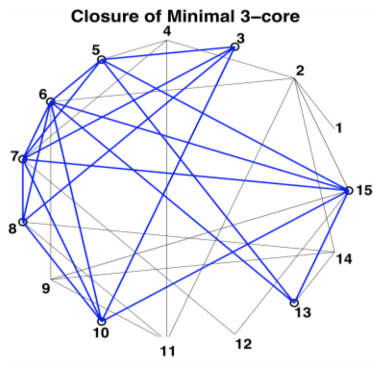

Fig. 10c

Fig. 10 shows an example of a graph composed of 15 nodes where the minimal 3-core has 4 nodes but the k-assembly has 8 nodes. In addition, Fig 10a only shows an example of one minimal 3-core, however, there exits other minimal 3-cores.

VI. k-Assemblies and Inhibition

Gunter Palm models of cell assemblies disregard the possibility of having inhibition since they are formed by the excitatory long range of cortico-cortical connections [4]. However, in order to improve the model of a cell assembly, inhibition was added to k-assemblies.

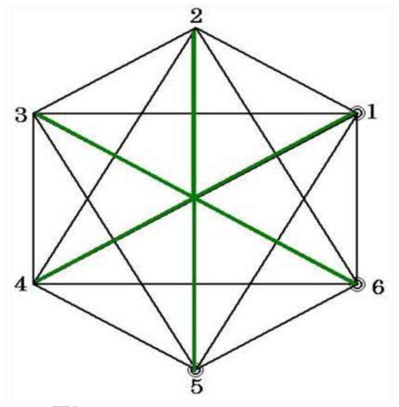

Fig. 11

Fig. 11 is an example of a network that has inhibition showed by the colored lines. This inhibitory network was modeled according to these guidelines:

~Nodes adjacent to each other have a weight of 1.

For example: $c(1,2) = 1$; $c(2,3) = 1$; $c(3,4) = 1$;

$c(4,5) = 1$; $c(5,6) = 1$; $c(6,1) = 1$.

~Any other connection, with the exception of the inhibition lines (colored in green), has a weight of ½. Consequently, its adjacency matrix is as follows:

$$A_d = \begin{pmatrix} 0 & 1 & 0.5 & -0.5 & 0.5 & 1 \\ 1 & 0 & 1 & 0.5 & -0.5 & 0.5 \\ 0.5 & 1 & 0 & 1 & 0.5 & -0.5 \\ -0.5 & 0.5 & 1 & 0 & 1 & 0.5 \\ 0.5 & -0.5 & 0.5 & 1 & 0 & 1 \\ 1 & 0.5 & -0.5 & 0.5 & 1 & 0 \end{pmatrix}$$

By using the algorithms described previously, a k-assembly can be found with the given adjacency matrix. As observed in Fig. 12, one k-assembly found by *bintprog* is composed of nodes 1,5 and 6 for threshold $k = 1.5$.

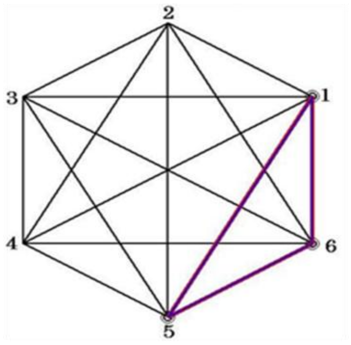

Fig. 12

VII. Discussion and Future Work

This summer research project was concentrated on the study and understanding of cell assemblies. In addition, we have established the definition of a k-assembly by creating a bridge between G. Palm's work and graph theory. As mentioned previously, a k-assembly is a group of neurons that are interconnected in such manner that by exciting a sufficiently large group of them, the entire network becomes excited, thus a concept can be recalled. In mathematical terms, a k-assembly is the closure of a minimal k-core. By using *bintprog*, a binary integer programming solver built in MATLAB, minimal k-cores can be found. Similarly, an algorithm was constructed to find the closure of a minimal k-core, a k-assembly. Since by increasing the cardinality of the network and connectivity it creates an exponential number of minimal k-cores, this algorithm finds at least one k-assembly. Finally, inhibition was added to the k-assembly model therefore a step closer to a more realistic and biological model of a cell assembly.

Even though there have been a lot of accomplishments this summer, there is still future work ahead. For instance, a given network of neurons most likely will contain many k-assemblies therefore it will be very useful to develop a method that will find all of them. In addition, the algorithm for adding inhibition to the k-assembly model could be improved. Also, G. Palm defines a cell assembly as the closure of a tight set, however we have only studied and worked with k-assemblies composed only a fraction of G. Palm's definition of a cell assembly [4]. In other words, if all of the k-assemblies are found in a given network, this doesn't mean that we have found all the cell assemblies according to G. Palm, therefore an algorithm can be constructed to find these cell assemblies [4].

Acknowledgments

This summer research project was supported by a NSF REU Grant DMS-0755294. I would to thank Dr. Steve Cox and Dr. Hicks for their guidance throughout this project. Finally, I would like to thank Tyler Young, Diane Taylor and Shaunak Day for their support in the project.

References

1. Milite, George A. “Hebb, Donald O. (1904-1985).” Encyclopedia of Psychology. 2001.

2. “Associative Learning.” Encyclopedia Britannica. http://www.britannica.com/EBchecked/topic/39477/associative-learning

3. Hebb, D. O. The Organization of Behavior: A neuropsychological theory. New York: Wiley. 1949.

4. Palm, Gunther. “Towards a Theory of Cell Assemblies.” Biological Cybernetics. 39, 181-194 (1981).

5. Balasundaram, Balabhaskar, Sergiy, Butenko, Hicks, Illya, and Sandeep Sandeep Sachdeva. “Clique Relaxations in Social Network Analysis: The Maximum k-plex Problem.” 2008.

6. “bintprog.” The MathWorks: Accelerating the pace of engineering and science.

MATLAB Code

```matlab
%function: Returns the largest k-core in the given graph

%Inputs:
%       k: The threshold/ k-value
%       n: The number of nodes in the network
%       d: The density or connectivity in the matrix (0-1)

%output: B-the adjacency matrix for the largest k-core

function B=largestKcore3(k,n,d)

A=sprand(n,n,d);
A=ceil((A+transpose(A))/2);
A(1:n+1:end)=0;

%bval vector is the nodes with less than k edges
bval=find(sum(A)<k & sum(A)>0);

%While there are values in bval keep erasing columns and rows in A
while sum(bval)>0

        i=1;
        A(:,bval(i))=0;
        A(bval(i),:)=0;

%Creates a new bval vector
        bval=find(sum(A)<k & sum(A)>0);

        i=i+1;

end
    B=A;
```

MATLAB Code

```
%function: Returns the minimal k-core in a graph

%Inputs:
%       k: The threshold/ k-value
%       n: The number of nodes in the network
%       B: The adjacency matrix for the largest k-core

%output: d: vector containing the minimal k-core

function P=using_bintprog(k,n,B)

%Inputs for bintprog:

f(:,length(B))=1;

Bp=B;
b=zeros(size(B,1),1);
b(length(b)+1,1)=1;

%Bintprog adjacency matrix for our purposes requires the diagonal
%to contain -k values.

for i=1:size(Bp,1)
    Bp(i,i)=-k;
end

Bp(size(Bp,1)+1,:)=1;

%bintprog built-in program is applied
d=bintprog(f,-Bp,-b)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plots the minimal k-core

A=B;

n1=size(A);
t1 = 2*pi/n1(1):2*pi/n1(1):2*pi;
x1 = cos(t1);
y1 = sin(t1);

for m=1:floor(n/2)
    text(x1(m)-.03, y1(m)+.06,num2str(m))
end

for z=floor(n/2)+1:n
    text(x1(z), y1(z)-.06,num2str(z))
end

%Outputs Figure 1 containing the Original Graph
```

```
figure(1)

title('Original Graph')
hold on

for i=1:size(A)

    for j=1:size(A)

        if A(i,j) > 0
            plot([x1(i) x1(j)],[y1(i) y1(j)],'k-','LineWidth',5)
        end
    end
end

axis off
```