Abstract:
Decomposing an image into fixed-sized sub-images has many applications in image storage and analysis, especially in the area of three dimensional imaging. The Connected Component Problem deals with the task of covering a binary image with tiles such that the sum of the number of connected components of each tile is minimized. This problem is relevant for storing and processing three dimensional images, in particular those using voxels. Interest in this study was originally spawned through the study of biological voxel images, and in particular, neural morphology. In this thesis, the Connected Component Problem is studied in both of its basic forms, allowing overlapping and the more restricted version where overlap is not allowed. The names of the two variations of the problem are the Connected Component Covering and the Connected Component Tiling. In the non-overlapping version, the Connected Component Tiling, a variation of this problem is proven to be NP-complete. An approximation factor is proven for both instances. Approximation algorithms are given for both cases along with upper bounds for each. Finally, real world results are given from actual implementations to show the effectiveness of the approximation algorithms.

FIXED-SIZE GEOMETRIC COVERING TO MINIMIZE THE NUMBER OF DISCONNECTED COMPONENTS

by

Andrew Joseph Tomascak

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

June 2003

N378
T591

APPROVAL

of a thesis submitted by

Andrew Joseph Tomascak

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographical style, and consistency and is ready for submission to the College of Graduate Studies.

Binhai Zhu, Ph.D.	[illegible]	June 24, 03
	(Signature)	Date

Approved for the Department of Computer Science

Michael Oudshoorn, Ph. D.	Michael Oudshoorn	June 25, 2003.
	(Signature)	Date

Approved for the College of Graduate Studies

Bruce McLeod, Ph. D.	Bruce L. McLeod	6-30-03
	(Signature)	Date

STATEMENT OF PERMISSION OF USE

Signature [illegible]

Date 25 June 2003

TABLE OF CONTENTS:

LIST OF FIGURES:

LIST OF TABLES:

ABSTRACT

Decomposing an image into fixed-sized sub-images has many applications in image storage and analysis, especially in the area of three dimensional imaging. The Connected Component Problem deals with the task of covering a binary image with tiles such that the sum of the number of connected components of each tile is minimized. This problem is relevant for storing and processing three dimensional images, in particular those using voxels. Interest in this study was originally spawned through the study of biological voxel images, and in particular, neural morphology. In this thesis, the Connected Component Problem is studied in both of its basic forms, allowing overlapping and the more restricted version where overlap is not allowed. The names of the two variations of the problem are the Connected Component Covering and the Connected Component Tiling. In the non-overlapping version, the Connected Component Tiling, a variation of this problem is proven to be NP-complete. An approximation factor is proven for both instances. Approximation algorithms are given for both cases along with upper bounds for each. Finally, real world results are given from actual implementations to show the effectiveness of the approximation algorithms.

INTRODUCTION

Background

In the study of three dimensional images, it is common to store such images in the form of voxels. Voxels, or volumetric pixels, are a convenient way to store data in three dimensions. Voxel images are primarily used in the field of medicine and are applied to X-Rays, CAT (Computed Axial Tomography) Scans, and MRIs (Magnetic Resonance Imaging) so professionals can obtain accurate 3D models of the human body. They are also used for various other applications, from geological surveying to gaming algorithms that make 3D acceleration unnecessary [MMG99, Na00].

Storing three dimensional images in the form of voxels is extremely memory intensive. Voxelization is the process of adding depth to an image using a set of cross-sectional images known as a volumetric dataset [ACY93]. These cross-sectional images (or slices) are made up of pixels. The space between any two pixels in one slice is referred to as interpixel distance, which represents a real-world distance. The distance between any two slices is referred to as interslice distance, which represents a real-world depth. The dataset is processed when slices are stacked in computer memory based on interpixel and interslice distances to accurately reflect the real-world sampled volume.

The motivation behind this study is rooted in computational biology. In the study of neural biology, neurons are photographed in microscopes. This is done by staining a neuron so that it can be seen with the visible eye, and viewing the stained neuron under a confocal microscope. The microscope is then given a precise focus and a picture is taken. The focus is then increased by a precise amount to slightly different depth of focus, and another picture is taken. This process repeats until the neuron has been fully studied. The end result is a "stack" of images covering all depths, allowing the images to be combined into a single three dimensional image, a voxel image.

In this thesis, we will discuss both the Connected Component Covering and the Connected Component Tiling problems and their different restrictions. The goals remain the same for both of these problems, and we shall refer to the general version of these problems as the Connected Component problem. A fast Connected Component algorithm could speed up storage and access times of voxel objects by allowing the compression of the cross-sectional images and optimized data access. This is the primary motivation behind studying the Connected Component problem. Such an image may be too large to work with in its entirety but smaller sub-images may be perfectly acceptable to work with.. Subdividing the cross-sectional images into just relevant image data (storing only the parts of the image containing information) will decrease the storage requirements and therefore speed up access and operations on this data. We can further increase this speed up by making sure that the data is grouped together in an intelligent manner. For instance, it is more desirable to have an image object contained in a single subdivision than split across two or more subdivisions. The more subdivisions that are used to store an object, the more memory must be accessed to do operations on this object.

This study of the Connected Component problem makes all of these optimizations possible. Using a good Connected Component algorithm, one can subdivide a three dimensional voxel image into smaller images that have a minimum number of connected components. This, in turn, allows faster processing and analysis to be done. The Connected Component problem is the task of making these subdivisions (equal in size and shape) such that each object is stored in as few subdivisions as possible. In the case of the neuron imaging, it is much easier to handle images with small numbers of connected components. Generally, a researcher is trying to study a single feature of the neuron, and additional components in the image only complicate the automation processes used for study.

Definition of the Connected Component Problem

To properly define this problem in more formal terms, some definitions must be given first. The input to this problem consists of two components: an image and a tile size. The input ***image*** component is a two dimensional $n \times m$ array, where each entry corresponds to a pixel value. For the purposes of this paper, all images are considered to be binary in nature, meaning each pixel is either "on" or "off" (a pixel is ***on*** if the array location is drawn black and is ***off*** if the array location is drawn white). In actual use on a computer, 1's and 0's are used in lieu of black and white in the array. A ***tile*** is a $j \times k$ subset of the array or image, and all pixels within this tile are said to be "covered". We will denote a single covering tile as $T = \{p_1, \ldots, p_u\}$ where p_i is each *on* pixel within the tile. Let $|T| = u$ where u is the number of *on* pixels in T and $ccn(T)$ be the number of connected components within T. Also, for simplicity, all images and tiles will be considered square $n \times n$ and $k \times k$ arrays respectively, but all results from this paper can be extended to $n \times m$ arrays or images and $j \times k$ sized tiles. A ***covering*** of an image is a set of tiles such that all on pixels are a member of at least one tile in the set, denoted here as $C = \{T_1, T_2, \ldots, T_v\}$. Let $|C| = v$, where v is the number of tiles in covering C. A ***tiling*** of an image is the same as covering, with the further restriction that no two (or more) tiles cover the same pixel. The number of connected components in each individual tile is then calculated and added to the sum of all the other tiles, producing a ***score*** for that particular tiling, given in Equation 1.1.

$$score(C) = \sum_{i=1}^{m} ccn(T_i) \text{ where } ccn(T_i) \text{ is the number of connected components in covering tile } T_i. \tag{1.1}$$

Given these definitions, one possible goal could be to find the covering that uses the minimum number of covering tiles, studied in [FPT81]. We will refer to this problem as the Minimum Tile Covering Problem and its optimal solution value as $opt_{\#}$. This is a packing problem directly related to the Set Cover problem and is discussed more in Section 3. The goal of the Connected Component problem is to minimize both the number of covering tiles and the total number of connected components (or score) within each covering or tiling. We shall call the optimum solution value to the second part

$opt_{\#CC}$. Therefore, an optimal answer to the Connected Component problem is found by first determining $opt_{\#CC}$ and then finding $opt_{\#}$ for that score,

$$opt_{\#CC} = \min\left(score\left(C\right)\right). \tag{1.2}$$

Connectivity, for this paper, is 4-neighbor connectivity, or *on* pixels that are 4-adjacent [GW02]. In Figure 1.1, there are three examples of connectivity that explain this more clearly. In all three examples, the grey pixel is the current pixel we are looking at. Neighboring *on* pixels are shown in black. The example on the left has four neighbors, all of them sharing an edge with the current pixel. These are the four spots that make up 4-adjacency. These pixels are all considered connected by 4-neighbor connectivity, which is simply the collection of all pixels that are 4-adjacent to each other. In the center example, all eight neighboring pixels are black, and these positions make up all the 8-adjacent positions. This is also known as 8-neighbor connectivity, and all pixels that share and edge or a corner are connected by 8-neighbor connectivity. In the right example, a pixel is shown that is 8-neighbor connected, but not 4-neighbor connected. We also consider 8-adjacency (8-neighbor connectivity) briefly, but it does little to change the basic problem.

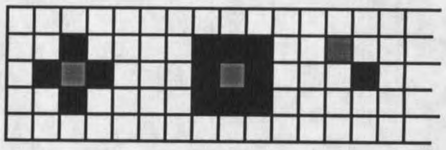

Figure 1.1 Examples of adjacency and connectivity.

The Connected Component problem is defined as the following: given an *n* by *n* array of pixels, with the pixels being on or off, find a tiling C of *k* by *k* tiles such that the total number of connected components of all tiles is minimized and the total number of tiles is also minimized.

In Figure 1.2, there are two instances of the same Connected Component problem, with two different solutions. In these two examples, the covering tiles are all 5 by 5 pixels and the array itself is 15 by 15 pixels. Both of these represent possible correct solutions, as all *on* pixels are indeed covered by at least one tile. The total size of the cover is $|C| = 2$ for the left tiling and $|C| = 3$ for the right tiling and the scores of both of these covers are $score(C) = 3$. There are only three components total in these instances, so this is the lowest score possible and therefore is $opt_{\#CC}$. The left tiling also computes $opt_{\#}$, and therefore is the optimal choice for this Connected Component problem.

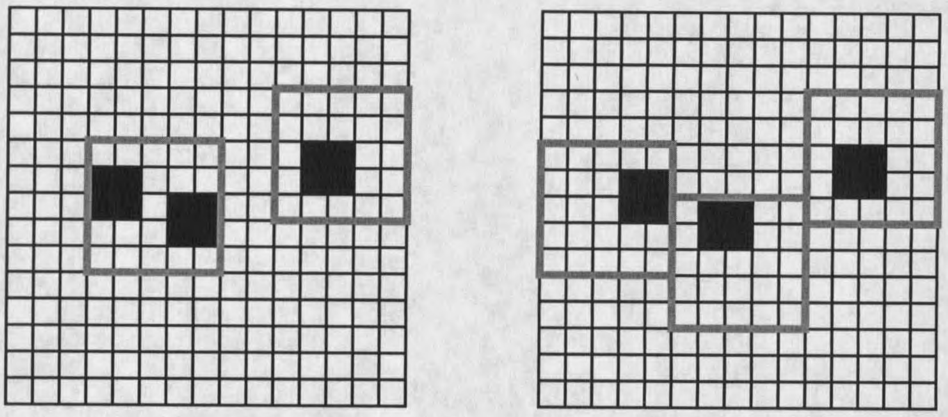

Figure 1.2 Two instances of the general Connected Component Problem.

Given a set of points in two dimensions (2D), covering them using the minimal number of fixed sized boxes (squares) was proven to be NP-complete more than two decades ago [FPT81]. Recently, Khanna, *et al.* studied a slightly different problem; namely, partition an $n \times n$ array of non-negative numbers into the minimum number of tiles (fixed-sized rectangles) such that the maximum weight of any tile is minimized. (Here the weight of a rectangle is the sum of the elements contained in it.) They proved that this problem is also NP-complete and that it can not be approximated closer than a factor of 1.25 in polynomial time. They also proposed a factor-2.5 approximation algorithm [KMP98]. We prove in this thesis that the Connected Component Tiling problem, in which overlapping is not allowed, is also NP-complete, reducing it from the 3SAT problem. Furthermore, we propose a factor-k^2 approximation algorithm.

We consider two variations of this problem. Variation one is the case where overlapping of covering tiles is allowed and will be referred to as the Connected Component Covering problem. The second case is where overlapping is not allowed, and we will refer to it as the Connected Component Tiling problem. This would correspond to real-world situations where redundancy of information is not desired. Examples would include image data that would be loaded twice in the course of processing. Because the second variation is simply a further constrained version of the first, any legal covering of the second type will also be a legal covering of the first given any particular image. A simple upper bound can easily be found for both types simply by finding an upper bound to the second case. The largest number of disconnected components that can be in a single tile, given 4-neighbor connectivity, is $\left\lceil \frac{k^2}{2} \right\rceil$ (in a checkerboard pattern) and given 8-neighbor connectivity is $\left\lceil \frac{k}{2} \right\rceil^2$. We will use $\frac{k^2}{2}$ from now on for 4-neighbor connectivity and $\frac{k^2}{4}$ for 8-neighbor connectivity as it has minimal effect on the results, and simplifies the reading. The largest number of tiles needed to cover an n by n array is $\left(\frac{n}{k}\right)^2$, again ignoring the ceiling function. Therefore the highest score a minimum tiling could achieve would be $\frac{n^2}{k^2} \cdot \frac{k^2}{2} = \frac{n^2}{2}$ for 4-neighbor and $\frac{n^2}{4}$ for 8-neighbor. This makes sense because that would also be the maximum number of disconnected components within an n by n array. This is our initial upper bound to begin with. The obvious initial lower bound for either version of the problem is to simply use the actual number of connected components within the image or array, as this will always be the minimum that a covering can score.

CONNECTED COMPONENT COVERING

In the case where overlapping of boxes is allowed, we allow covering tiles to overlap. Two or more tiles are allowed to cover the same pixel or set of pixels, but these pixels still count toward the score of each tile covering them. This would correspond to a real world problem where we are concerned with reducing the total number of components to a minimum. This would be the case if the per object cost of an operation is of more concern than the storage and redundancy of extra data. This instance of the Connected Component problem can be reduced to the Set Cover Problem [CLRS01] and gives us an approximation bound. We shall refer to this instance as the Connected Component Covering, CCC.

Set Cover Problem

In the case where overlapping is allowed, the Connected Component Covering highly resembles the Set Cover problem. The Set Cover problem is defined as follows: Given a set S of n elements, $S = \{x_1, x_2, \ldots, x_n\}$, and a collection C of subsets of S, find the minimum number of subsets such that every element in S is in at least one subset. In Figure 2.1, we can see a visual example of the Set Cover problem. In this example we can see a master set of elements, represented by dots, and the subsets which are smaller collections of the elements. To solve this problem, we would look for a collection C' of subsets, such that all elements are included in at least one set of the collection. It can be seen here that some subsets *must* be included, because some elements are only included in one subset.

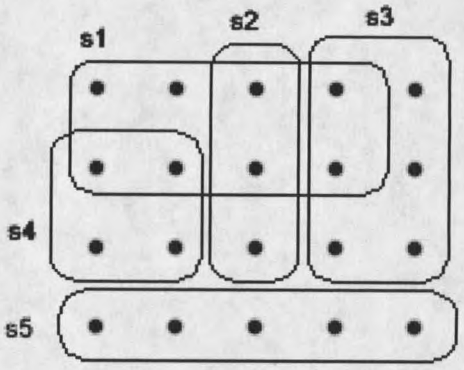

Figure 2.1 An example of the Set Cover problem.

Johnson proved that the Set Cover problem is approximable within a factor of $1 + \ln|S|$, using a simple greedy algorithm, in which the subset with the largest number of non-included elements is picked repeatedly until all elements are included [Jo74]. The idea behind the proof of the GREEDY-SET-COVER algorithm is to give all elements a cost, then to total these costs to give a maximum cost to any one subset. Basically, the proof shows that the optimal answer may not be larger than the harmonic of the largest set. The harmonic of the set is $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} \le \ln n + 1$, where n is size of the set. In Figure 2.1, the greedy algorithm would pick s1, s5, s3, s4, and s2, in that order. At this point the algorithm would end, as all elements are now covered. In this particular example all subsets were used, however this is not always the case.

In the study of the CCC problem, each possible covering tile can be seen as a set of pixels with a score given by the number of connected components within the tile. Reducing the problem as such, we can now use known algorithms for the Set Covering problem to solve the Connected Component Covering problem. Both [Ch79] and [Jo74] have proven an upper bound to the Set Covering problem using a simple greedy heuristic running in $O(n \log n)$ time. The algorithm simply picks a covering box that covers up the most uncovered pixels that will still be connected within a single box. When no more covering boxes can be found that meet these criteria, and there are still uncovered pixels remaining, the connective limit is raised by one. Now the algorithm picks the box covering the most pixels that will still be only be covering two unconnected components. And so forth, until all remaining pixels are covered. First established in [Jo74] and [Lo75], an ever better upper bound was proven (using the same algorithm) later in [Ch79]. When this bound is applied to the Connected Component Covering, this basic greedy algorithm is guaranteed to return an answer within a factor of $1 + 2 \cdot \ln(k)$. In some special cases, an approximation algorithm is not needed because the optimal solution can be found in polynomial time itself. We will look at such a case next.

Special Case

If the height of the bounding cover is the same as or greater than the height of the overall array $(j \geq m)$, then it is basically a one dimensional problem, in the sense that the covers will always be left or right of each other and never above or below. The covering tiles can be seen as sliders on the array that can overlap, much like sliding glass doors on a house. An example can be seen in Figure 2.2.

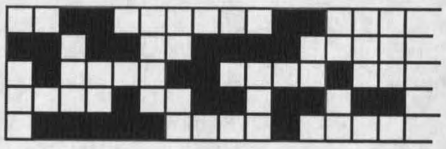

Figure 2.2 An example of the one-dimensional special case of the CCC problem.

This special case can even be solved in $O(n \cdot k)$ time by using dynamic programming, where *n* is the length of the array and *k* is the length of the cover. It is given that for any *n* length array, where $n \leq k$, the cost of $C[n]$ is simply going to be the number of connected components inside the tile, because only one tile is needed to cover the array and using only one tile will automatically result in the optimal solution. Recall that the lower bound for this problem is simply the number of connected components within the array, so covering all connected components with a single tile will result in the lower bound solution. This is the base case for the recurrence relationship, and is used to determine the dynamic programming algorithm. Let $score(x, y)$ be the optimal score of an array starting at *x* and ending at *y* along the horizontal axis. Because we are only concerned about the pixels within this sub array, we consider the space to the left of *x* and the space right of *y* to be all *off* pixels. Let $score_{left}(x, y)$ be the same, with the exception that all pixels to the *left* of this array still must be counted if they are within the covering.

$$\begin{aligned}
&\text{For } n : n \leq k \\
&\quad score(1,n) = \text{Number of connected components between} \\
&\qquad\qquad\qquad \text{start and column } n \\
&\text{For } n : n > k \\
&\quad score(1,n) = \min\left\{score(1,i) + \min\left(score_{left}(i+1,n)\right)\right\}, n-k \leq i \leq n-1
\end{aligned} \tag{2.1}$$

Equation 2.1 shows that this algorithm takes *n* steps to get to length *n*. The minimum cost is recorded in a one-dimensional array at each step, so that it can be used again to generate the remaining steps. Each step takes *k* sub-steps, and each sub-step must find another minimum cost cover. However, this only needs to be done once for all *k* configurations, so this is only the addition of *k* steps. The final time complexity for this special case is then $O(n \cdot k)$ and the space complexity is $O(n+k)$.

CONNECTED COMPONENT TILING

Given the case where the overlap of cover boxes is not allowed, the problem becomes more interesting. This is a more common problem of picture and graph partitioning, and data analysis. Again we can tighten the upper approximation bounds to something more accurate. First we show what the hardness result is, and then we show an $O\left(n^2\right)$ approximation algorithm that returns a result within a factor of k^2 of the optimal answer. We refer to this instance of the problem as the Connected Component Tiling problem, or CCT.

Hardness of Approximation

Hardness of approximation is proven by reducing the 3SAT problem to the Connected Component Tiling problem. The 3SAT problem was shown in [Co71] to be NP-complete, and is a common starting point for many NP-complete reductions.

Theorem 2.1 When the input to the problem is strictly points, CCT is NP-complete.

This is proven in two steps. The first is to prove that CCT is in NP, using the decision version of the problem. To do this we think of the problem with an additional variable, *limit*, where the decision is 1 if the array can be tiled with a score less than or equal to *limit* and 0 if it cannot. If we have an answer certificate, it is a simple task to check what the answer's score is and if it is lower than *limit*. All this can be done in polynomial time; therefore the problem is indeed in NP.

The second step to this proof is to show that the CCT problem can be polynomially reduced from a known NP-complete problem, and therefore can be reduced in polynomial time from *all* NP-complete problems. The NP-complete problem that we have chosen to reduce from is the 3SAT problem, which was proven to be NP-complete in [Co71].

3SAT is defined using the following terms. A ***literal*** in a Boolean formula is an occurrence of a variable or its negation. A Boolean formula is in ***conjunctive normal form***, or ***CNF***, if it is expressed as an AND of ***clauses***, each of which is the OR of one or more literals. A Boolean formula is in ***3-conjunctive normal form***, or ***3-CNF***, if each clause has exactly three distinct literals. In the 3SAT problem, we try to see if a Boolean formula in 3-CNF can be satisfied.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_1 \vee \neg x_3) \tag{3.1}$$

Equation 3.1 shows a Boolean formula in 3-CNF format. In this example there are three clauses and each clause has three literals. There are also only three variables in this example. This particular formula can be solved in several ways, including any configuration where x_1 is true.

In some way, the above problem can also be viewed as a graph problem, demonstrating a relationship between the two problems. Each clause is seen as three vertices which represent the literals, none of which connect. Each vertex is connected to every vertex in every other clause, as long as they are not in the same clause, or are contradicting literals. For instance, an x vertex cannot connect to a $\neg x$ vertex in another clause. If the original formula had n clauses, then there should now be $3n$ vertices. The Boolean formula can then be satisfied if and only if a clique of size n exists. A *clique* is a set of vertices in which there exists an edge between every single vertex in the clique, in other words, form a complete sub graph. Because no vertices within the same clause are connected, the only way to get a clique of size n is to have vertex from each clause included. If such a clique exists, then there can be no contradicting literals because the sub graph is complete, and contradicting literals do not have connecting vertices. In the Figure 3.1 example, the darker lines represent a possible 3-*clique* solution where x_1 is true, x_3 is false, and x_2 could be either true or false.

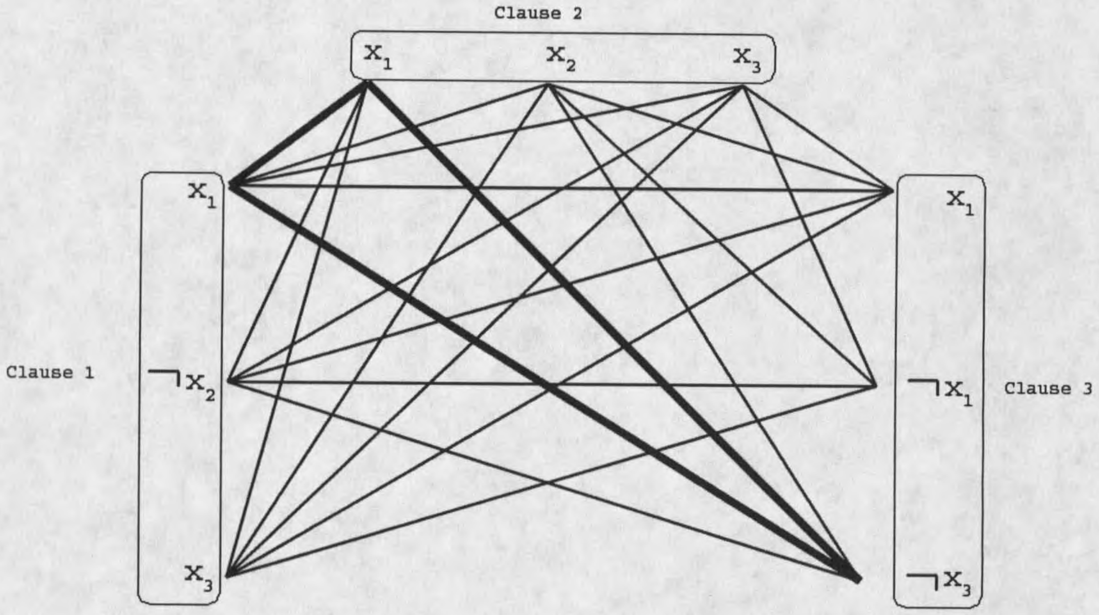

Figure 3.1 An example of a Boolean 3-CNF formula viewed as a graph problem.

Both the BOX-COVER and BOX-PACK problems were proven to be NP-complete in [FPT81]. These problems are the same as the CCC problem and CCT problem respectively, except for the added restriction of the connected components score. If we simply use a special type of image where it is obvious that we can achieve $opt_{\#CC}$, then we only have to worry about the $opt_{\#}$ part of the solution. To create such a special image is rather straight forward: we must simply assure that all connected components within an image can be covered in such a way that no component is split. Using connected components that are no larger than a single point satisfies this requirement. We are now guaranteed that $opt_{\#CC}$ will be achieved.

At this point, we can now use the same proof method found in [FTP81]. We create a closed loop of connected components (or points in this case) which we will call a wire. Figure 3.2 gives a slightly abstracted demonstration of the construction and its use. This figure shows a construction with *m* variables and *n* clauses. Each wire loop will represent a variable in the 3SAT problem, and the Clause Points are where three of the variables come within close proximity to each other. The wire itself is made from single pixels spaced far enough apart that only two of them can be covered by a single covering tile. As can be seen from Figure 3.3, each covering tile can only cover two pixels, and therefore, the loop of wire can only be covered in two different ways. This corresponds to a variable being either true or false.

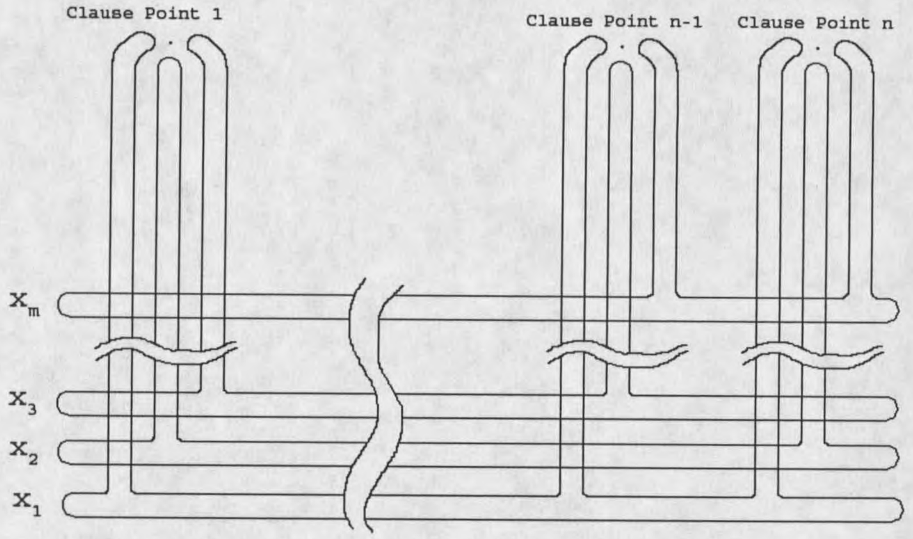

Figure 3.2 A conceptual overview of the reduction construction.

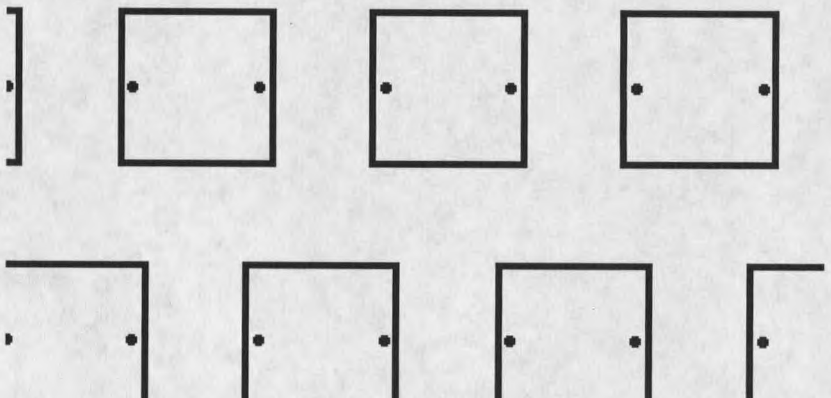

Figure 3.3 The two different ways that a horizontal line of points can be covered.

If we let a given variable i of 3SAT have $2 \cdot K_i$ pixels in its wire, then $opt_{\#}$ for the wire will have to be equal to K_i. Figure 3.4 shows how intersections of wire are handled. In this image, the pixels are offset in the given pattern to demonstrate how the relative "polarity" of the wire is maintained. To negate a variable, a pixel is removed from the part of the wire leading into a clause and added to the part of the wire leaving the clause. This reverses the "polarity" of the wire at the clause, and all other properties are maintained. Finally, Figure 3.5 demonstrates how a Clause Point is handled.

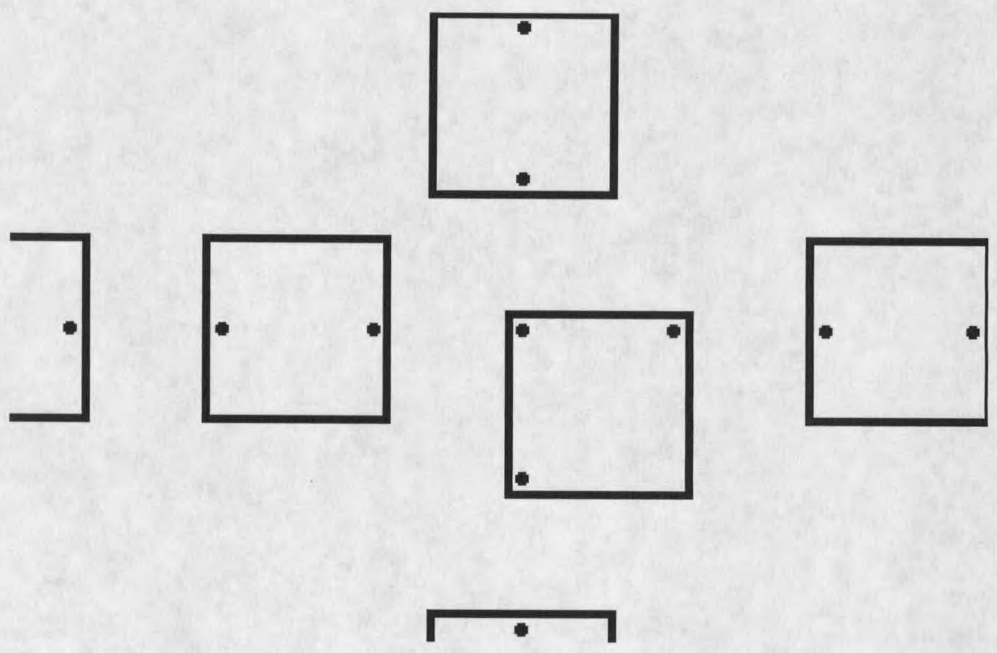

Figure 3.4 Example of an intersection.

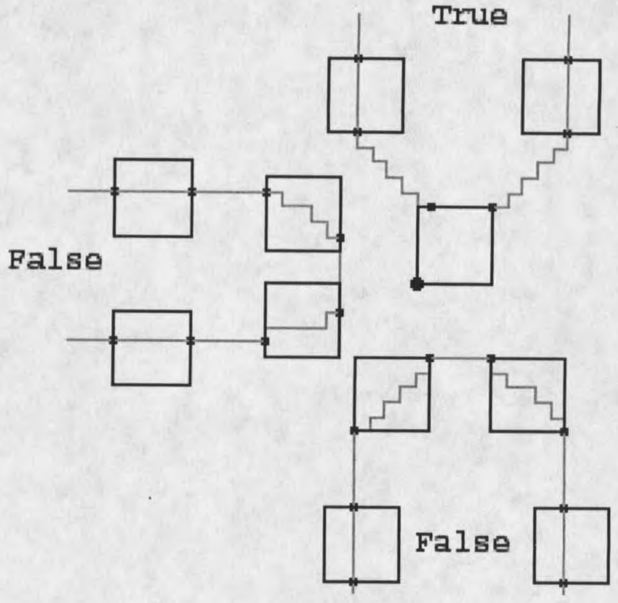

Figure 3.5 A conceptual overview of the reduction construction.

An additional point is added to a spot where three wires come within close proximity to each other. In this figure, the clause point is the largest point, and the grey lines represent the layout of the wires. Basically, the loops are arranged so that this additional spot can be covered without the addition of another tile as long as one or more of the wires is in the "TRUE" configuration. If none of the literals is true, then an additional tile will be needed to cover this clause point. In the above figure, it can be seen how the two "false" loops cannot cover the additional point, but the third "true" loop can cover it simply by shifting an existing tile.

The minimum number of covering tiles, should no additional tiles be needed for clause points, can be found by Equation 3.2. It is simply the sum of all the tiles needed for each loop of wire minus the number of intersections, since we use one less tile for each intersection.

$$\sum_{i=1}^{M} K_i - N_C \quad | \, M = \text{number of variables}, N_C = \text{number of intersections} \qquad (3.2)$$

The construction is then coverable by this minimum sum if and only if the 3SAT formula is satisfiable. Thus the reduction is complete.

□

Strip-Approximation

We can now prove the existence of an approximation within a factor of 2 of the optimal answer for the Minimum Tile Covering problem. We will then use this result to prove the existence of a k^2 approximation of the CCT problem. An approximation algorithm is an algorithm that returns an answer with much less work than finding the actual solution. However, the answer returned is not guaranteed to be the optimal answer and thus is only considered an approximation. If an approximation algorithm is said to be with a factor of *n* of the optimal answer, then the answer is guaranteed to be no worse than *n* times more than the optimal answer (in the case of a minimization problem such as CCT). This is also known as the approximation performance ratio.

As we discussed earlier, the goal of the Minimum Tile Covering Problem is simply to find the covering that minimizes $|C|$, and it has been proven by [FPT81] to be NP-complete. We refer to the optimal solution to this problem as $opt_{\#}$. We will call the optimal solution to the CCT score problem $opt_{\#CC}$.

Lemma 3.2 There exists a factor-2 approximation of the Minimum Tile Covering Problem.

First, the array is sliced with horizontal lines, so that each sub array is exactly the height of the bounding box. (This could also be done with vertical strips, but we will stick with the horizontal for this paper.) Each of these strips can now be treated as a special case one-dimensional version of the Minimum Tile Covering Problem. Each can also be solved optimally using dynamic programming similar to the special case algorithm described for the CCC problem, with the major difference being that overlap is now forbidden. Each strip will have a set of covers T_i where $|T_i| \leq \left\lceil \frac{n}{h} \right\rceil$. ($|T_i|$ is the number of covering tiles in T_i, and also the minimum number of tiles needed to cover all pixels in T_i regardless of score.) The total number of boxes is then $|MinStripCover| = \sum_i |C_i|$.

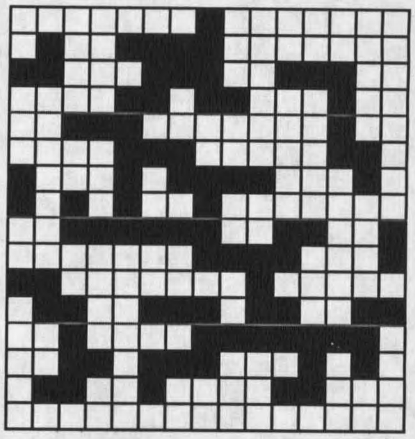

Figure 3.6 An example of a "sliced" array.

Using the striping method above is unlikely to find $opt_{\#}$ because one or more of the optimal tiles may be across two strips. However, it will result in no more than twice the optimal number of bounding boxes. This can be seen by looking at an optimally placed tile that does not fall exactly into the boundaries of a strip. This tile will be bisected by at most one strip boundary. Therefore, the pixels in this optimal covering tile can be covered with only two tiles from within each of the strips that the original tile intersected with. As the optimal covers cross into each strip, it takes only one additional cover (per optimal cover) to cover the same area. See Figure 3.6 for an example of this.

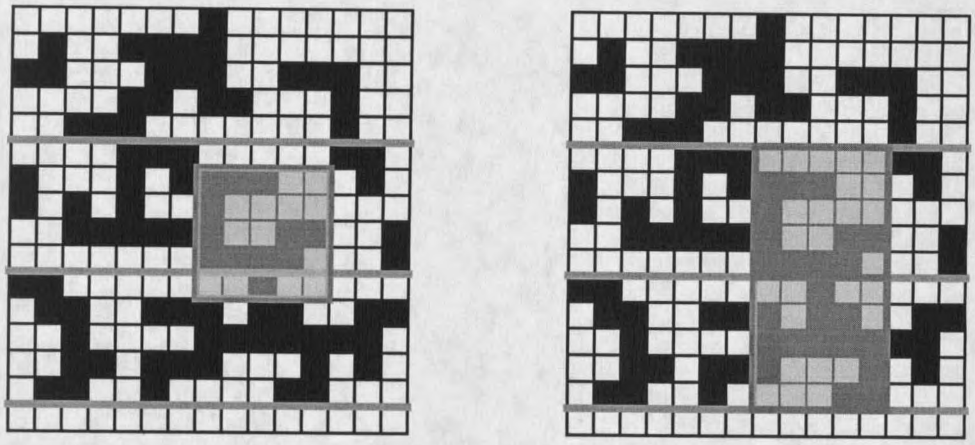

Figure 3.7 An example of how an optimal tile translates into two different tiles in the strip approximation.

Overlapping tiles are simply pushed out toward the sides. Since we now have at most two covering tiles for every one that the optimal answer has, we are guaranteed that the approximate answer is within a factor of two of $opt_{\#}$. We now know that $\sum_i |N_i| \leq 2opt_{\#}$.

□

Theorem 3.3 There exists a factor-k^2 approximation for the CCT score problem.

Now we can prove the k^2 approximation algorithm for the CCT problem. It is apparent that $opt_{\#} \leq opt_{\#CC}$ because $opt_{\#}$ is the smallest number of tiles that cover all of the *on* pixels regardless of score. Doubling both sides of this inequality and combining it with Lemma 3.2, we get Equation 3.3:

$$\sum_i |N_i| \le 2opt_{\#} \le 2opt_{\#CC} \tag{3.3}$$

Since every box has at most $\frac{k^2}{2}$ connected components, the total number of connected components is at most $\frac{k^2}{2}\sum_i |N_i| \le k^2 opt_{\#} \le k^2 opt_{\#CC}$. This gives us a final bound of our approximation algorithm of being no worse than a factor of k^2.

□

Theorem 3.4 There also exists a factor-k(k-1) approximation for the CCT score problem.*

The previous bound can be further improved by moving the strips so that we try all k different striping configurations along a given axis. This can be accomplished by starting a striping one pixel down from the last try and repeating until we return to our original striping configuration. Due to the Pigeon Hole Principal, at least one striping configuration has $\left\lceil \frac{opt_{\#}}{k} \right\rceil$ tiles that fit optimally. If we then double the remaining tiles (because the non-optimal tiles must be covered with two tiles) we get a slightly tighter approximation bounds as seen in Equation 3.4:

$$\begin{aligned}
&\sum_i |N_i| \le 2 \cdot opt_{\#} - \left\lceil \frac{2 \cdot opt_{\#}}{k} \right\rceil \le 2 \cdot opt_{\#CC} - \left\lceil \frac{2 \cdot opt_{\#CC}}{k} \right\rceil \\
&\frac{k^2}{2} \sum_i |N_i| \le k^2 \cdot opt_{\#} - \left\lceil \frac{k^2 \cdot opt_{\#}}{k} \right\rceil \le k^2 \cdot opt_{\#CC} - \left\lceil \frac{k^2 \cdot opt_{\#CC}}{k} \right\rceil \\
&\frac{k^2}{2} \sum_i |N_i| \le k \cdot opt_{\#} (k-1) \le k \cdot opt_{\#CC} (k-1)
\end{aligned} \tag{3.4}$$

□

In the case of the neuron mapping application, we are dealing with a single connected object in the input image, so we can actually improve upon this approximation. A neuron exhibits a branching structure resembling a tree. In the worst case of a tree structure, a covering tile has no actual branch joints within it but instead has branches coming in from every side. Each side can support $\frac{k}{2}$ independent branches, so the total of four sides would be $2k$. Therefore, in the case where the pixels within the image form a tree, our final approximation is guaranteed to be no worse than $2k$, giving us the following corollary:

Corollary 3.5 If the pixels within the image form a tree, then there is a 4k approximation for CCT.

IMPLEMENTATION

These strip-and-solve algorithms have final running times of $O\left(n^2\right)$ and $O\left(n^2 \cdot k\right)$, and are proven to be within a factor of k^2 of the optimal solution. However, real world usage of these algorithms shows a return of much better results. A program was written to produce random images with a random pixel density. The optimal tiling solution was found for each image, and two different approximation algorithms were also run on each image. The first approximation algorithm was run with a single set of strips where each strip is solved optimally, and the second algorithm was run where all possible strips are tested. The strip-and-solve algorithm used is as follows:

1) Segment the image into *k* high strips, resulting in $\left\lceil \frac{n}{k} \right\rceil$ strips.

2) Use the $O(n \cdot k)$ dynamic programming algorithm to solve each strip.

The second part of the testing algorithm simply tests all *k* possible different segmentation configurations. An example image follows to help clarify the test and its results.

First, a random density is produced between 0 and 1. This number is used to determine how many possible *on* pixels will in the resulting image. The image is then created with by randomly positioning these pixels. Figure 4.1 shows one of these random images that were created.

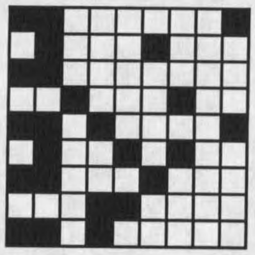

Figure 4.1 An example of a random image created by the test implementation.

First, the test algorithm finds the optimal answer to the CCC problem. In the case of Figure 4.1, the optimal answer is 13 components in 7 tiles. This can be seen in Figure 4.2, which shows an optimal tiling.

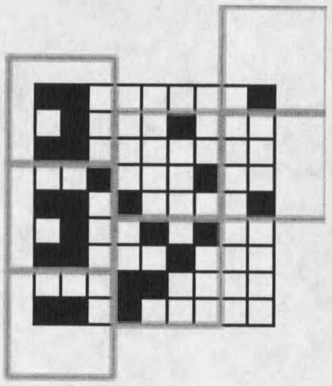

Figure 4.2 An optimal tiling.

Next, the test algorithm uses the strip-and-solve algorithm to find an approximate solution. The tiling resulting from this part of the test program can be seen in Figure 4.3.

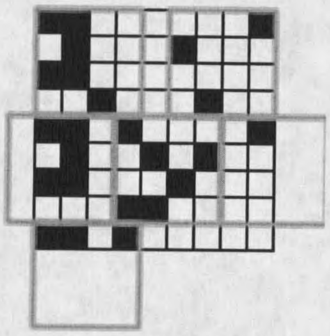

Figure 4.3 The tiling resulting from running the approximation algorithm with a single offset.

The first strip starts with the very first row of pixels. As the reader can see, this actually results in one *less* tile being needed. However, the total number of connected components has increased by one due to the component in the lower-middle of the image being split by two tiles. The score for this tiling is 14 components and 6 tiles.

The last part of the test conducts this same basic algorithm, but in all *k* possible strip configurations. This change increases the running time of the algorithm by a factor of *k* because of this, but improves our upper bound as was proven by Theorem 3.4. The results of running this version of the algorithm can be seen in Figure 4.4.

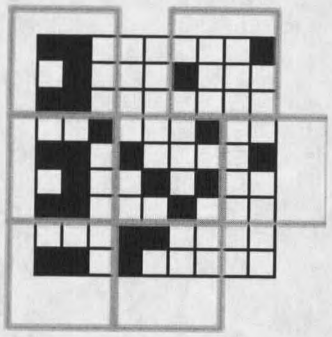

Figure 4.4 The tiling resulting from running the approximation algorithm on all offsets.

The score for the tiling of Figure 4.4 is 13 components and 7 tiles, which was also the optimal tiling score. Therefore this is also an optimal tiling.

The previous example represents a fairly average case among the various tests. More often then not, running the algorithm on all the offsets produced a component score that was optimal. This just was not generally done with an optimal number of tiles. On average, running only the single offset approximation algorithm produced an answer that was a factor of 1.13 of optimal for the number of tiles and a factor of 1.14 of optimal component score. Running the algorithm using all offsets produced an average factor of 1.13 of the optimal number of tiles and a factor of 1.04 of the optimal component score. The majority of these tests were done with 9x9 and 10x10 images, so more testing is probably warranted to determine averages among larger images. See Table 4.1 for a listing of tiling and score results for the 9x9 images with 4x4 coverings. The first two columns represent the optimal num of tiles and score respectively. The third and fourth columns represent the approximation result returned from running the approximation algorithm on only one set of strips. The fifth and sixth columns give the approximation performance ratio. The closer this value is to one, the more accurate the result was and result of one means that the optimal value was returned. The seventh and eighth columns represent the approximation result returned when the algorithm was tested on every strip configuration. The ninth and tenth columns are the approximation performance ratios for the ALL OFFSETS results. The averages at the bottom represent the average performance ratio for all the runs, including several runs not shown in the table. These additional runs were two 10x10 images with 4x4 tiles and a single 10x10 image with 3x3 tiles. Results were similar to the runs with 9x9 images with 4x4 tiles.

Table 4.1 Tiling and Score results

OPTIMAL		SIN. ROW APP.		factor of optimal		ALL OFFSETS		factor of optimal	
# Tiles	Score	# Tiles	Score	# Tiles	Score	# Tiles	Score	# Tiles	Score
7	12	8	15	1.142857	1.25	9	12	1.285714	1
5	11	6	14	1.2	1.272727	7	11	1.4	1
6	18	7	18	1.166667	1	7	18	1.166667	1
7	13	6	14	0.857143	1.076923	7	13	1	1
1	2	2	3	2	1.5	1	2	1	1
8	10	8	11	1	1.1	8	11	1	1.1
7	11	8	12	1.142857	1.090909	8	12	1.142857	1.090909
8	13	8	15	1	1.153846	9	14	1.125	1.076923
6	10	7	13	1.166667	1.3	8	10	1.333333	1
4	8	6	10	1.5	1.25	5	8	1.25	1
7	10	8	13	1.142857	1.3	8	11	1.142857	1.1
2	2	2	2	1	1	2	2	1	1
6	11	7	12	1.166667	1.090909	7	12	1.166667	1.090909
3	5	4	6	1.333333	1.2	4	5	1.333333	1
5	12	7	14	1.4	1.166667	7	12	1.4	1
8	12	8	13	1	1.083333	8	12	1	1
4	11	4	11	1	1	4	11	1	1
5	8	5	8	1	1	5	8	1	1
5	14	7	17	1.4	1.214286	7	14	1.4	1
5	9	6	9	1.2	1	6	9	1.2	1
7	11	8	12	1.142857	1.090909	8	12	1.142857	1.090909
3	5	4	5	1.333333	1	3	5	1	1
3	4	3	4	1	1	3	4	1	1
7	9	8	11	1.142857	1.222222	9	10	1.285714	1.111111
2	3	2	3	1	1	2	3	1	1
7	9	7	10	1	1.111111	8	9	1.142857	1
8	15	9	16	1.125	1.066667	9	16	1.125	1.066667
6	10	6	10	1	1	6	10	1	1
7	9	8	11	1.142857	1.222222	8	11	1.142857	1.222222
8	12	8	12	1	1	8	12	1	1
6	9	7	11	1.166667	1.222222	7	11	1.166667	1.222222
8	14	7	17	0.875	1.214286	8	15	1	1.071429
7	9	7	11	1	1.222222	7	9	1	1
7	12	8	14	1.142857	1.166667	9	13	1.285714	1.083333
5	12	6	13	1.2	1.083333	6	13	1.2	1.083333
7	12	7	12	1	1	7	12	1	1
7	12	7	12	1	1	7	12	1	1
9	10	8	13	0.888889	1.3	9	10	1	1
6	9	8	11	1.333333	1.222222	8	9	1.333333	1
8	10	8	12	1	1.2	9	10	1.125	1
Averages				**1.12978**	**1.14349**			**1.129402**	**1.035819**

CONCLUSIONS

In this thesis, background and motivation has been given for the study of the Connected Component problem. A variation of the problem was proven to be NP-complete, giving insight to the difficulty of this problem. We have looked into the two main variations of the problem, CCC and CCT, and have developed approximation algorithms for both. The CCC problem was rather trivial in complexity, and an approximate solution within a factor of $1+2\cdot\ln(k)$ was easily found with a simple greedy algorithm. The non-overlapping version, the CCT problem, was more interesting. An algorithm was given to find an approximate solution within a factor of k^2. Furthermore, if the pixels in the image are known to form a tree, then the solution is guaranteed to within a factor of $2k$. Finally, real world tests were run to compare actual results with theoretical ones, and were shown to be much better.

REFERENCES

[ACY93] A. Kaufman, D. Cohen, R. Yagel. Volume Graphics. *IEEE Computer*, 1993.

[Ch79] V. Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res. 4*, pp. 233-235, 1979.

[Co71] Steven Cook. The complexity of theorem proving procedures. *Procedings of the Third Annual ACM Symposium on Theory of Computing.* 151-158, 1971.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms, 2nd ed.*, Cambridge: MIT Press, 2001.

[FPT81] R. Fowler, M. Paterson and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Proc. Letters*, 12(3):133-137, 1981.

[GW02] Rafael Gonzalez, Richard Woods. *Digital Image Processing, 2nd ed.*, New Jersey: Prentice Hall, 2002.

[Jo74] D. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci. 9*, pp. 256-278, 1974.

[KMP98] S Khanna, S. Muthukrishnan and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th ACM-SIAM Symp on Discrete Algorithms (SODA '98)*, pages 384-393, Jan, 1998.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J Comput.* 11(2):329-343, 1982.

[Lo75] L. Lovasz. On the ratio of optimal integral and fractional covers, *Discrete Math. 13*, pp. 383-390, 1975.

[MGG99] S. Märkle, A. Gothan, H. Gothan. Interactive Simulation of Soft-Tissue for Surgery Training. In: *Proceedings of CARS'99 Computer Assisted Radiology*, pp. 855-859, June 1999.

[Na00] D. Nadeau. Volume Scene Graphs. *Proceedings of the 2000 IEEE symposium on Volume visualization,* pp. 49-56, 2000.