Cyclic cutwidth of three dimensional cubes

Ray N. Gregory

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Mathematics Commons

Recommended Citation

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
CYCLIC CUTWIDTH OF THREE DIMENSIONAL CUBES

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Arts
in
Mathematics

by
Ray N. Gregory
March 1998
CYCLIC CUTWIDTH OF THREE DIMENSIONAL CUBES

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Ray N. Gregory
March 1998

Approved by:

Dr. Joseph Chavez, Chair, Mathematics

Dr. Roland Trapp

Dr. Joel Stein

Dr. Terry Hallett, Graduate Coordinator

Dr. Peter Williams, Department Chair
The general cutwidth problem and the problem of embedding a three dimensional cube on a cycle of the form of a regular octagon is described. Next, the solutions of the minimum cyclic cutwidth of a three dimensional cube embedded onto a cycle problem are investigated. The total number of such solutions is found and the solutions are listed in a table and categorized. Switching neighbor vertices between which the cutwidth is three of minimum cyclic cutwidth of three dimensional cubes solutions are described as generators of a permutation group. Lastly, the structure of the permutation groups of the minimum cyclic cutwidth solutions is investigated and the implications of using the permutation groups to find solutions in higher dimensions is discussed.
ACKNOWLEDGMENTS

I wish to recognize the help I received from Bill Rokos. His work coding the computer program in C++ was invaluable. He spent many hours changing the data structures I devised into C++ code. I would also like to thank Dr. Joseph Chavez whose work and guidance gave me the inspiration to complete this project. Lastly, I would like to thank my family for the support they gave me during the long hours of research and typing that went into this project.
# TABLE OF CONTENTS

**ABSTRACT** ................................................................. iii

**ACKNOWLEDGMENTS** ....................................................... iv

**LIST OF TABLES** ............................................................ vii

**LIST OF ILLUSTRATIONS** ................................................ viii

## CHAPTER ONE

1-1 Introduction .............................................................. 1
1-2 The General Cutwidth Problem ........................................ 2
1-3 Cyclic Cutwidth of Three Dimensional Cubes ...................... 5
1-4 Arrangements of a Three Dimensional Cube Embedded on a Regular Octagon ......................................................... 8
1-5 Minimum Cyclic Cutwidth of a Three Dimensional Cube ........ 9
1-6 The Four Types of Minimum Cyclic Cutwidth Solutions .......... 11

## CHAPTER TWO

2-1 Switching Neighbor Vertices of Minimum Cyclic Cutwidth Solutions ................................................................. 13
2-2 Permutation Groups of Solutions Obtained by Switching Neighbor Vertices of Minimum Cyclic Cutwidth Solutions ............. 15
2-3 Graphs of the Permutation Groups Produced by Switching Neighbor Vertices of Minimum Cyclic Cutwidth Solutions ........ 17
LIST OF TABLES

Table 1. List of the Minimum Cyclic Cutwidth Solutions . 10

Table 2. Solution Type Generated by a Switch of Neighbor Vertices Between Which there is a Cutwidth of Three ................. 14

Table 3. Generators of the Permutation Groups ........ 16
LIST OF ILLUSTRATIONS

Figure 1. Domain Graph G and Range Graph H ........... 2
Figure 2. Square with Vertices Labeled with Numerals .. 4
Figure 3. Path Range Graph with Cutwidth = 2 ........... 4
Figure 4. Path Range Graph with Cutwidth = 4 ........... 4
Figure 5. Three Dimensional Unit Cube, Ordered Triple Vertex Labeling ............... 6
Figure 6. Three Dimensional Unit Cube, Numerical Vertex Labeling ............... 6
Figure 7. Vertices of a Three Dimensional Cube Embedded on an Octagon .......... 6
Figure 8. Cutwidth Diagram .......... 6
Figure 9. Solution #1 [] .................. 12
Figure 10. Solution #2 {} .................. 12
Figure 11. Solution #10 <> .................. 12
Figure 12. Solution #4 () .................. 12
Figure 13. Neighbor Switch Permutation Group #1 .......... 18
Figure 14. Neighbor Switch Permutation Group #2 .......... 18
Figure 15. Neighbor Switch Permutation Group #3 .......... 19
Figure 16. Neighbor Switch Permutation Group #4 .......... 19
Figure 17. Neighbor Switch Permutation Group #5 .......... 20
Figure 18. Neighbor Switch Permutation Group #6 .......... 20
The following set of definitions will be used throughout the paper. A graph \( G \) is a structure consisting of a set of vertices \( V \) and a set of edges \( E \) that connect the vertices. An \( n \)-cube is a \( n \)-dimensional unit cube with \( 2^n \) vertices and \( n(2^{n-1}) \) edges. We represent vertices of an \( n \)-cube as \( n \)-tuples of 0's and 1's. Edges connect pairs of vertices on the \( n \)-cube that differ in only one coordinate in their \( n \)-tuples. The word edge will only be used when referring to the \( n \)-cube.

The general cutwidth problem is one of several types of graph labeling problems studied in combinatorics. A graph labeling problem is concerned with finding the optimal way(s) to label the vertices of a domain graph \( G \) on a range or host graph \( H \), subject to certain objectives. The function that labels or embeds the vertices of the domain graph \( G \) on the range graph \( H \) is one to one and onto. Thus, each vertex on the domain graph \( G \) has a corresponding vertex on the range graph \( H \) and each edge of the domain graph \( G \) has a corresponding path on the range graph \( H \). A path is a sequence of edges that connect adjacent vertices on the range graph \( H \). Vertices are called adjacent if connected by an edge on the domain graph \( G \) or by a path on the range graph \( H \).

Neighbor vertices are vertices of the range graph \( H \) that are listed or drawn next to each other on the range graph,
without regard to their connection by a path. In Figure 1, vertices 1 and 2 are both adjacent and neighbor vertices. This is because they are both connected by an edge on the domain graph G and are drawn next to each other on the range graph H. Vertices 2 and 3 are not adjacent since they are not connected by an edge on the domain graph G or by a path incident to vertices 2 and 3 on the range graph H. Vertices 2 and 3 are neighbor vertices though, because they are drawn next to each other on the range graph H. Vertices 2 and 4 are adjacent because of the edge incident to them on the domain graph G and the path incident to them on the range graph H. But, they are not neighbor vertices, since they are not drawn next to each other on the range graph H.

The general cutwidth problem is a recently considered problem from graph theory. The survey paper by F. R. K. Chung [2] provides much of the background of what is currently known about the general cutwidth problem. The
general cutwidth problem is concerned with finding the optimal way(s) to embed the vertices and edges of the domain graph G on the range graph H such that the maximum number of paths drawn between any two neighbor vertices on the range graph is minimized.

The linear cutwidth, $L_n(G)$, of a graph G is a special case of the general cutwidth problem with the range graph a path $P$ with n vertices. If any two vertices are connected with an edge on graph G, then they must also be connected with a path on graph P. If a path travels past a vertex on the range graph $P$ on its way to a more distant vertex, then it is considered to pass through each vertex that it passes by. The maximum number of paths passing between any two neighbor vertices on $P$ is calculated and recorded as the cutwidth of that embedding. Each of the $n!$ permutations of the vertices of G embedded on path $P$ has its own cutwidth. The solution of the cutwidth problem $L_n(G)$ is the minimum of the cutwidths recorded for each of the possible embeddings of G on $P$.

For example, let the domain graph be the graph of the square with the vertices labeled numerically. (See Figure 2.) If this square is embedded on a path, then there are $4! = 24$ ways to label the vertices of the path. The value of the solution of the minimum cutwidth must be greater than or equal to two. Since every vertex on the domain graph has two edges incident to it, then two paths must exist that are incident to each vertex on the range graph. Because the range graph is a path, vertices must be placed on the ends of
the path. Hence, any vertex placed on the end has two paths leave it. Thus, the linear cutwidth between an end vertex and its neighbor vertex is two, regardless of the permutation of the vertices. Hence, the value of the linear cutwidth of a square embedded on a path is greater than or equal to two.

Square with Vertices Labeled with Numerals
Path Range Graph with Cutwidth = 2
Path Range Graph with Cutwidth = 4

If we find a permutation of the vertices that has only two paths between each pair of vertices, then we have found a solution. (See Figure 3.) There are a total of eight permutations of the vertices that result in two as the value of the minimum linear cutwidth of a square embedded on a path. These permutations are: 1234, 2341, 3412, 4123, 4321, 3214, 2143, and 1432. Note that each solution in the list also has its reverse order included in the list. The other 16 permutations, one of which is shown in Figure 4, lead to a cutwidth value greater than two.

J. D. Chavez and L. H. Harper [1] have solved the cutwidth problem when n-cubes are embedded on range graphs that are linear and have also solved the cutwidth problem when n-cubes are embedded on grids. When the range graph H
is a grid, the cutwidth problem is concerned with minimizing the number of paths that pass any point on the grid.

The cyclic cutwidth of an n-cube, Q_n, has as its range graph a cycle in the form of a regular 2^n-gon, where 2^n is the number of vertices on the graph of Q_n. The range graph is not required to be regular, but it helps to visualize and draw the problem if it is a regular polygon.

1-3 CYCLIC CUTWIDTH OF THREE DIMENSIONAL CUBES

When calculating the cyclic cutwidth of a three dimensional cube, the domain graph is a three dimensional unit cube Q_3. The vertices of Q_3 are ordered triples (x, y, z) with only 0's and 1's as coordinates. Edges connect vertices that differ in only one coordinate in their ordered triples. Thus (0, 0, 0) connects to (1, 0, 0), (0, 1, 0), and (0, 0, 1). (See Figure 5.)

For ease of use in this paper, we will label the vertices of the three dimensional unit cube in a numerical order. Vertex 1 will correspond to the vertex labeled as an ordered triple (0, 0, 0). Vertex 2 will correspond to (1, 0, 0). Vertex 3 will correspond to (1, 0, 1). Vertex 4 will correspond to (0, 0, 1). Vertex 5 will correspond to (0, 1, 1). The other three vertices will be numbered so as to complete a circuit. (See Figure 6.)

The eight vertices and twelve edges of the three dimensional cube are embedded on a range graph that is a cycle in the form of a regular octagon H_8. (See Figure 7.)
Three Dimensional Unit Cube
Ordered Triple Vertex Labeling

Three Dimensional Unit Cube
Numerical Vertex Labeling

Vertices of a Three Dimensional Cube Embedded on an Octagon

Since the function that embeds the domain graph on the range graph is one to one and onto, the vertices that are connected by edges on the cube are still connected by paths on the octagon. Paths actually travel around the outside of the octagon passing through each vertex they pass by. Paths
can travel in two ways around the octagon. A path is actually directionless, but the two ways correspond to traveling clockwise or counter clockwise around the octagon from a starting point to an ending point.

For visual clarity, we will draw paths to connect the vertices with straight lines through the interior of the octagon. This method of drawing the paths through the interior of the octagon does not change the value of the cutwidth. There is some confusion though when a path travels directly through the center of the octagon. These paths must be drawn slightly curved in order to miss the center of the octagon.

To calculate the cyclic cutwidth, \( C(Q_3) \), count the number of paths that pass between each pair of neighbor vertices of an embedding of the vertices and edges of \( Q_3 \) on \( H_8 \). Given the visual representation of paths passing through the octagon one would locate the center of the octagon \( H_8 \) and draw eight rays that originate at the center and pass between each pair of neighbor vertices of \( H_8 \). (See Figure 8). The number of paths that each ray crosses is then found and the maximum number of paths crossed between any two neighbor vertices of \( H_8 \) is noted and recorded as the cutwidth of that drawing of the range graph \( H_8 \). This value is the cyclic cutwidth of \( Q_3 \) for that particular permutation of the vertices and arrangements of the paths of \( Q_3 \) embedded onto \( H_8 \).

The process is then repeated for a new permutation of the vertices or new arrangement of the paths. When each
possible permutation and all of the arrangements of the paths for each possible permutation have been checked, then the value of \( C(Q_3) \) is the minimum of the recorded maximum values.

1-4 ARRANGEMENTS OF A THREE DIMENSIONAL CUBE EMBEDDED ON A REGULAR OCTAGON

One can think of a permutation of the vertices of \( Q_3 \) as a distinct way to squash the vertices and the stretchable and contractible edges of a three dimensional cube onto a two dimensional regular octagon. Which side of the cube is placed facing upwards along with how the cube is twisted when squashed determines which of the permutations is generated.

The eight vertices of the graph of \( Q_3 \) can be embedded on the octagon \( H_8 \) in \((8 - 1)! = 5040\) ways. After the vertices are embedded on \( H_8 \), the paths representing the edges must still be drawn. The two vertices incident to an edge on the domain graph \( Q_3 \) partition the remaining vertices of the octagon \( H_8 \) into two subsets. The path on \( H_8 \) representing this edge will travel past all of the vertices in one subset or the other. Thus there are two ways to draw each edge as a path. This results in \( 2^{12} = 4096 \) ways to draw the paths for each permutation of the vertices of \( Q_3 \). Thus the total number of different embeddings of the vertices of \( Q_3 \) onto \( H_8 \) with each distinct path pattern counted is \((7!)(2^{12}) = (5040)(4096) = 20,643,840\) cases.
The cutwidth of certain cases of a three dimensional cube $Q_3$ embedded onto an octagon $H_8$ could have a value as high as twelve. This occurs when the twelve paths are drawn such that each path passes between a pair of neighbor vertices.

The computer program in Appendix A found that there are 96 cases out of the 20,643,840 possible cases that lead to a maximum cutwidth of three. This is the smallest cutwidth value found by the program which calculated the cutwidth of each of the 20,643,840 cases. Thus, there are 96 solutions to the minimum cyclic cutwidth of a three dimensional cube embedded on a cycle in the form of an octagon.

The minimum cyclic cutwidth solutions have all of their paths drawn the shortest distance around the octagon and thus pass through the smallest subset of vertices possible. In addition, a minimum cyclic cutwidth permutation will have no path incident to two vertices that are directly across the octagon from one another. This will keep paths from passing through the center of the octagon.

Due to the symmetry of the underlying range graph, rotation of the vertices by $\pi/4$ or any of its multiples does not change the value of the cutwidth. Hence, there is no lack of generality if the permutations are listed in Table 1 with vertex number 1 first. If you reverse the order of the vertices of any of the solutions, then this results in a new solution that is also included in the 96 solutions. The number of this reverse order of the vertices solution is also
<table>
<thead>
<tr>
<th>Solution Number of Reverse Order</th>
<th>Permutation</th>
<th>Solution Number of Reverse Order</th>
<th>Permutation</th>
<th>Solution Number of Reverse Order</th>
<th>Permutation</th>
<th>Solution Number of Reverse Order</th>
<th>Permutation</th>
<th>Solution Number of Reverse Order</th>
<th>Permutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>178765432</td>
<td>4</td>
<td>15876432</td>
<td>6</td>
<td>12365478</td>
<td>8</td>
<td>13456782</td>
<td>2</td>
<td>14365872</td>
</tr>
<tr>
<td>2</td>
<td>17856432</td>
<td>3</td>
<td>15864327</td>
<td>5</td>
<td>12354678</td>
<td>7</td>
<td>13246783</td>
<td>4</td>
<td>14756832</td>
</tr>
<tr>
<td>3</td>
<td>17856342</td>
<td>2</td>
<td>15863427</td>
<td>4</td>
<td>12354768</td>
<td>6</td>
<td>13246587</td>
<td>5</td>
<td>14756382</td>
</tr>
<tr>
<td>4</td>
<td>17875643</td>
<td>1</td>
<td>15875643</td>
<td>3</td>
<td>12365784</td>
<td>5</td>
<td>13456782</td>
<td>2</td>
<td>14856372</td>
</tr>
<tr>
<td>5</td>
<td>17685342</td>
<td>6</td>
<td>15687342</td>
<td>4</td>
<td>12375468</td>
<td>7</td>
<td>13246783</td>
<td>5</td>
<td>14756382</td>
</tr>
<tr>
<td>6</td>
<td>17986534</td>
<td>5</td>
<td>15986345</td>
<td>6</td>
<td>12387456</td>
<td>8</td>
<td>13456782</td>
<td>2</td>
<td>14856372</td>
</tr>
<tr>
<td>7</td>
<td>17985342</td>
<td>4</td>
<td>15987345</td>
<td>5</td>
<td>12387465</td>
<td>7</td>
<td>13456872</td>
<td>4</td>
<td>14765382</td>
</tr>
<tr>
<td>8</td>
<td>17985342</td>
<td>3</td>
<td>15987345</td>
<td>4</td>
<td>12387465</td>
<td>7</td>
<td>13456872</td>
<td>4</td>
<td>14765382</td>
</tr>
<tr>
<td>9</td>
<td>17985342</td>
<td>2</td>
<td>15987345</td>
<td>4</td>
<td>12387465</td>
<td>7</td>
<td>13456872</td>
<td>4</td>
<td>14765382</td>
</tr>
<tr>
<td>10</td>
<td>17985342</td>
<td>1</td>
<td>15987345</td>
<td>4</td>
<td>12387465</td>
<td>7</td>
<td>13456872</td>
<td>4</td>
<td>14765382</td>
</tr>
</tbody>
</table>

List of the Minimum Cyclic Cutwidth Solutions

TABLE 1

Each of the solutions is drawn in Appendix B.
The 96 solutions of the minimum cyclic cutwidth problem have 4 distinct arrangements of their paths. The first pattern has all of the neighbor vertices of the octagon connected by a path and a square pattern in the center of the octagon. (See Figure 9.) Since the neighbor vertices are connected by a path, they are considered adjacent. This square pattern arrangement, denoted by a [], appears 12 times in the 96 solutions. It is interesting to note that this pattern occurs when the permutation of the vertices requires you to travel a circuit though the vertices and edges of the cube ending at the vertex you started from. The vertices are labeled on the octagon in the order they are traveled through while completing the circuit. The second pattern looks similar to the first pattern, but a pair of neighbor vertices which had three paths between them have switched places. This leaves two pairs of neighbor vertices not adjacent, with one adjacent pair between them. (See Figure 10.) This pattern, which is denoted by a {}, appears the most of the four arrangements. It appears a total of 48 times in the 96 solutions. The third pattern has four pairs of neighbor vertices not adjacent alternately around the octagon. (See Figure 11.) This pattern, denoted by <>, occurs 12 times in the 96 solutions. The fourth pattern also has two pairs of neighbor vertices not adjacent, but now these pairs of nonadjacent neighbor vertices are directly across the octagon from one another. (See Figure 12.) There are 24 instances
of this pattern in the 96 solutions, each denoted by a (). The first time that each pattern occurs in the 96 solutions is shown in the figures below.

Solution #1 []  
Figure 9

Solution #2 {}
Figure 10

Solution #10 <>  
Figure 11

Solution #4 ()  
Figure 12
Once the solutions of the minimum cyclic cutwidth problem are found, it is natural to ask if it is possible to generate additional solutions if only one solution is known. The answer to this question is yes. If neighbor vertices between which there is a cutwidth of three are switched, then a different solution from the list 96 solutions is the result. When Figures 9 through 12 are studied you will notice that each type of solution has four sets of neighbor vertices between which the cutwidth is three and four sets of neighbor vertices between which the cutwidth is two. On every type of solution, the neighbor vertex sets between which the cutwidth is two and three alternate as you travel in either direction around the octagon.

If you begin with a solution similar to solution #1 in Figure 9, then switching any of the four sets of neighbor vertices between which there is a cutwidth of three will yield a solution similar to solution #2 in Figure 10. Thus a solution of type [ ] becomes a { } after undergoing a switch of neighbor vertices between which there is a cutwidth of three. When you begin with a solution of type <> similar to solution #10 in Figure 11 and switch any of the four sets of neighbor vertices between which there is a cutwidth of three, then a solution of type { } is also the result. Likewise, if you
start with a solution of type () similar to solution #4 in Figure 12, then a solution of type {} is generated by a neighbor vertex switch between which the cutwidth is three.

If you begin with a solution of type {}, then any of the other three types could be the result. If the neighbor vertices that were switched to generate the {} solution from the [] solution are switched again, then the original [] solution is the result. When the neighbor vertices of cutwidth three directly across the octagon from the neighbor vertices that were switched in the first step are switched, then a solution of type {} becomes a solution of type <>. If one of the other two neighbor vertices between which there is a cutwidth of three are switched, then a solution of type () is created. Hence, a {} type solution will become a [] under one potential neighbor vertex switch. It will become a <> under a second potential neighbor vertex switch and a () under the other two potential neighbor vertex switches. The above discussion is summarized in Table 2.

TABLE 2

<table>
<thead>
<tr>
<th>Solution Type</th>
<th>Solution Type Generated by a Switch of Neighbor Vertices Between Which there is a Cutwidth of 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>{}</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>{}</td>
</tr>
<tr>
<td>()</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>[] or &lt;&gt; or ()</td>
</tr>
</tbody>
</table>
While it is possible to generate additional solutions from other solutions, it isn't possible to generate all of the 96 solutions from a single solution using only switches of the sets of neighbor vertices between which the cutwidth is three. Given any of the 96 solutions, it is only possible to generate 15 additional solutions. The 16 solutions that can be generated from one another are isomorphic to a permutation group. It is possible to generate any of the other 15 members of a permutation group using one or more switches of the neighbor vertices between which the cutwidth is three, but not possible to generate any solution outside of the permutation group. The elements of the permutation group are switches of the neighbor vertices between which the cutwidth is three along with composition of the switches. The operation on the group is composition of switches of neighbor vertices between which the cutwidth is three. The order of the the set of solutions S is $|S| = 96$ and the order of each of the permutation groups $|P| = 16$. Thus, $|S|/|P| = 96/16 = 6$ which is the number of permutation groups of the solutions. Thus, six permutation groups are isomorphic to six families of order 16 in S.

Each permutation group is isomorphic to each of the other five permutation groups in regards to the types of solutions in each of the permutation groups. Each permutation group contains two solutions of type [], two solutions of type <>, four solutions of type (), and eight
solutions of type \{{\}}.

If we begin with solution #1 in Figure 9 then we see that the neighbor vertices between which the cutwidth is three are: 1 and 2, 3 and 4, 5 and 6, 7 and 8. Written as cycles, these neighbor vertex switches become: (1 2), (3 4), (5 6), (7 8), with (1 2) signing switching vertices 1 and 2. These neighbor vertex switches are the generators of neighbor vertex switch permutation group #6. (See Figure 18).

As shown in Table 3, the generators do not necessarily remain the same for each of the six permutation groups. Pairs of permutation groups will have the same generators because each solution has a reverse order of its vertices that is also one of the 96 solutions. This causes each of the six neighbor switch permutation groups to have a reverse order of the vertices neighbor switch permutation group with the same generators. The reverse order of the vertices neighbor switch permutation group #1 is neighbor switch permutation group #4. The reverse order of the vertices of permutation group #2 is permutation group #5 and the reverse order of the vertices of permutation group #3 is permutation group #6.

<table>
<thead>
<tr>
<th>Permutation Group</th>
<th>Generators of the Permutation Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 and 4</td>
<td>(1 4), (2 3), (5 8), (6 7)</td>
</tr>
<tr>
<td>2 and 5</td>
<td>(1 8), (2 7), (3 6), (4 5)</td>
</tr>
<tr>
<td>3 and 6</td>
<td>(1 2), (3 4), (5 6), (7 8)</td>
</tr>
</tbody>
</table>

TABLE 3
The identity $I$ of a permutation group is defined as the neighbor vertex switch where every vertex remains in its place on the cycle. An identity neighbor vertex switch is also generated by a composition of two of the same switches executed in succession. It is important to note that all 16 members of a permutation group will have a cutwidth of three between the same four pairs of neighbor vertices. Thus permutation groups 3 and 6, which are generated by the generators: $(1\ 2), (3\ 4), (5\ 6), (7\ 8)$, can be written as the following cycles and composition of cycles: $I$, $(1\ 2)$, $(3\ 4)$, $(5\ 6)$, $(7\ 8)$, $(1\ 2)(3\ 4)$, $(1\ 2)(5\ 6)$, $(1\ 2)(7\ 8)$, $(3\ 4)(5\ 6)$, $(3\ 4)(7\ 8)$, $(5\ 6)(7\ 8)$, $(1\ 2)(3\ 4)(5\ 6)$, $(1\ 2)(3\ 4)(7\ 8)$, $(1\ 2)(5\ 6)(7\ 8)$, $(3\ 4)(5\ 6)(7\ 8)$, $(1\ 2)(3\ 4)(5\ 6)(7\ 8)$.

Permutation groups 2 and 5 can be written as the cycles and composition of cycles: $I$, $(1\ 8)$, $(2\ 7)$, $(3\ 6)$, $(4\ 5)$, $(1\ 8)(2\ 7)$, $(1\ 8)(3\ 6)$, $(1\ 8)(4\ 5)$, $(2\ 7)(3\ 6)$, $(2\ 7)(4\ 5)$, $(3\ 6)(4\ 5)$, $(1\ 8)(2\ 7)(3\ 6)$, $(1\ 8)(2\ 7)(4\ 5)$, $(1\ 8)(3\ 6)(4\ 5)$, $(2\ 7)(3\ 6)(4\ 5)$, $(1\ 8)(2\ 7)(3\ 6)(4\ 5)$.

Permutation groups 1 and 5 can be written as the cycles and composition of cycles: $I$, $(1\ 4)$, $(2\ 3)$, $(5\ 8)$, $(6\ 7)$, $(1\ 4)(2\ 3)$, $(1\ 4)(5\ 8)$, $(1\ 4)(6\ 7)$, $(2\ 3)(5\ 8)$, $(2\ 3)(6\ 7)$, $(5\ 8)(6\ 7)$, $(1\ 4)(2\ 3)(5\ 8)$, $(1\ 4)(2\ 3)(6\ 7)$, $(1\ 4)(5\ 8)(6\ 7)$, $(2\ 3)(5\ 8)(6\ 7)$, $(1\ 4)(2\ 3)(5\ 8)(6\ 7)$.

2-3 GRAPHS OF THE PERMUTATION GROUPS PRODUCED BY SWITCHING NEIGHBOR VERTICES OF MINIMUM CYCLIC CUTWIDTH SOLUTIONS

The lines connecting each solution indicate that the solutions are one neighbor vertex switch from each other.
Figure 13

Figure 14
NEIGHBOR SWITCH PERMUTATION GROUP #3

Figure 15

NEIGHBOR SWITCH PERMUTATION GROUP #4

Figure 16
NEIGHBOR SWITCH PERMUTATION GROUP #5

Figure 17

NEIGHBOR SWITCH PERMUTATION GROUP #6

Figure 18
CHAPTER THREE

3-1 TIME COMPLEXITY

Due to the multiple calculations involved with finding a new permutation from the 5040 permutations and then finding and checking the cutwidth of each of the 4096 arrangements of the edges around the octagon for that permutation, a computer must execute many instructions as it evaluates the cyclic cutwidth of each case. The computer used to evaluate the minimum cyclic cutwidth of a three dimensional cube has a 25 MHz processor that took approximately 3 minutes to solve the problem and print the data on the screen. Considering recent advances in computer technology, it is not beyond the ability of current computers with faster processing to calculate the cyclic cutwidth of a three dimensional cube in less than one minute. For simplicity I would like to propose that all 20,643,840 cases of the cyclic cutwidth of a three dimensional cube could be checked and the minimum found today or in the near future on an advanced computer in 1 second. This is \( \frac{1}{180} \) of the time actually used by the 25 MHz processor.

The four dimensional cube embedded onto a two dimensional 16-gon would have 16 vertices to permute and 32 edges to draw. Thus the four dimensional cube would have

\[
(15!)(2^{32}) = (1,307,674,368,000)(4,294,967,296) = 5.62 \times 10^{21}
\]
cases to check. This is $2.72 \times 10^{14}$ times the number of cases as the three dimensional cube. If it would take an advanced computer one second to check each case and find the minimum cyclic cutwidth of a three dimensional cube, then it would take $2.72 \times 10^{14}$ seconds or approximately 8.6 million years to find the value of the minimum cyclic cutwidth of a four dimensional cube using a fast computer. Hence, finding the minimum cyclic cutwidth of an $n$-dimensional cube, $n \geq 4$, is impossible using my computer program without astronomical advances in the processing speed of computers.

3-2 IMPLICATIONS OF SWITCHING NEIGHBOR VERTICES OF MINIMUM CYCLIC CUTWIDTH SOLUTIONS IN HIGHER DIMENSIONS

If permutation groups of minimum cyclic cutwidth of $n$-dimensional cubes solutions exist in dimensions higher than three, then it could be possible to use the knowledge of how the permutation groups are generated in three dimensions to help find all of the solutions in higher dimensions. The $n$-cube would be embedded onto a two dimensional $2^n$-gon, with $2^n$ the number of vertices of the $n$-cube. If one solution of the type, or any other type, could be found from each permutation group, then all of the solutions could be generated by hand in a reasonable amount of time using neighbor vertex switches between which the cutwidth is three. It is possible to believe that one could find all of the type minimum cyclic cutwidth solutions to the four or even higher dimensional cube embedded onto a two dimensional
2^n-gon by inspection, since this type of solution is generated by completing a circuit of the vertices. The solver would need to find all of the possible circuits of the vertices and check them manually by drawing a picture. Then using neighbor vertex switching and the symmetric distribution of the types of solutions in the permutation groups, it would be possible to find all of the other solutions within all of the permutation groups. While this would definitely require a great attention to detail and many hours, it is conceivable that all of the solutions and hence the number of solutions to a n-dimensional cube embedded onto a two dimensional 2^n-gon could be found without using a computer which are currently too slow to solve the problem using my program in a reasonable amount of time.
APPENDIX A

COMPUTER PROGRAM

This program written in C++ calculates the number of permutations that are generated when the eight vertices of the three dimensional cube are arranged at the vertices of a regular octagon. It also finds the total number of cases that occur when the edges of the cube are drawn on the octagon, either clockwise or counterclockwise, as paths for each permutation. The main purpose of the program is to find the minimum cyclic cutwidth of a three dimensional cube embedded on a regular octagon and list all of the solutions. The solution's permutation and a binary code that describes the direction the path is drawn is printed. 0 indicates the edge is drawn clockwise around the octagon and 1 indicates the edge is drawn counterclockwise around the octagon.

```c++
program octagon;
uses crt, strings;

const

OUTPUT_FILE_NAME='c out.dat';
FORWARD = 0;
BACKWARD = 1;
EXPECTED_PERMS = 5040;
EXPECTED_CASES = 20643840;
NUMBER_OF_VERTICES = 8;
NUMBER_OF_EDGES = 12;
MAX_CUT = 4;
DIGITS_OF_BINARY = 12;
NUMBER_OF_CONFIGS = 4096;
```
type
temp_counter_type = record
    point_1: integer;
    point_2: integer;
end;
edge_record_type = record
    point_1: integer;
    point_2: integer;
    direction: integer;
end;
ftype = text;

var
    output_file: ftype;
    perm: array[1..NUMBER_OF_VERTICES] of integer;
    octagon_array: array[1..(NUMBER_OF_VERTICES * 2)] of integer;
    edges_array: array[1..NUMBER_OF_EDGES] of edge_record_type;
    counter_array: array[1..NUMBER_OF_VERTICES, 1..NUMBER_OF_VERTICES] of integer;
    temp_counter: temp_counter_type;
    LARGEST CUT: integer;
    count: integer;
    ACTUAL_PERMS: integer;
    ACTUAL_CASES: longint;

function open_data_file (var output_file: ftype): boolean;
var
    output_open_error: boolean;
begin
    assign (output_file, OUTPUT_FILE_NAME);
    rewrite (output_file);
    if ioreturn = 0 then
        output_open_error := FALSE
    else
        output_open_error := TRUE;
    if (output_open_error = FALSE) then
        begin
            open_data_file := TRUE;
            exit;
        end
    else
        open_data_file := FALSE;
end;

procedure zero_counters;
var
    index1, index2: integer;
begin
    for index1 := 1 to NUMBER_OF_VERTICES do
for index2:= 1 to NUMBER_OF_VERTICES do
  counter_array[index1][index2]:= 0;
end;

function get_largest_cut: integer;
var
  index1, index2, largest_cut: integer
begin
  largest_cut:= 0;
  for index1:= 1 to NUMBER_OF_VERTICES do
    for index2:= NUMBER_OF_VERTICES downto (index1 + 1) do
      if (counter_array[index1][index2] > largest_cut) then
        largest_cut:= counter_array[index1][index2];
      end;
  get_largest_cut:= largest_cut;
end;

procedure set_up_first_perm;
var
  index: integer;
begin
  for index:= 1 to NUMBER_OF_VERTICES do
    perm[index]:= index;
end;

procedure zero_configuration;
var
  index: integer;
begin
  for index:= 1 to DIGITS_OF_BINARY do
    edges_array[index].direction:= FORWARD;
end;

procedure set_octagon_array
var
  index: integer;
begin
  for index:= 1 to NUMBER_OF_VERTICES do
    begin
      octagon_array[index]:= perm[index];
      octagon_array[index+NUMBER_OF_VERTICES]:= perm[index];
    end;
end;

procedure set_edges_array;
var
  index: integer;
begin
  edges_array[1].point_1:= 1;
  edges_array[1].point_2:= 2;
edges_array[2].point_1:= 2;
edges_array[2].point_2:= 3;

edges_array[3].point_1:= 3;
edges_array[3].point_2:= 4;

edges_array[4].point_1:= 4;
edges_array[4].point_2:= 5;

edges_array[5].point_1:= 5;
edges_array[5].point_2:= 6;

edges_array[6].point_1:= 6;
edges_array[6].point_2:= 7;

edges_array[7].point_1:= 7;
edges_array[7].point_2:= 8;

edges_array[8].point_1:= 8;
edges_array[8].point_2:= 1;

edges_array[9].point_1:= 1;
edges_array[9].point_2:= 4;

edges_array[10].point_1:= 2;
edges_array[10].point_2:= 7;

edges_array[11].point_1:= 3;
edges_array[11].point_2:= 6;

edges_array[12].point_1:= 5;
edges_array[12].point_2:= 8;
end;

procedure initialize_vars;
begin
zero_counters;
set_edges_array;
set_up_first_perm;
ACTUAL_PERMS:= 0;
ACTUAL_CASES:= 0;
end;

procedure print_results;
var
index, largest_cut: integer;
begin
largest_cut:= get_largest_cut;
write(output_file, 'Perm: ');
for index:= 1 to NUMBER_OF_VERTICES do
begin

write(output_file, perm[index]);
end;
write(output_file, ' '); write(output_file, 'Octagon: ');
begin
write(output_file, octagon_array[index]);
end;
write(output_file, ' '); write(output_file, 'Binary: ');
begin
write(output_file, edges_array[index].direction);
end;
write(output_file, ' '); write(output_file, 'Largest Cut: ', largest_cut);
end;

function find_index(target: integer): integer;
var
index: integer;
found: boolean;
begin
index := 1;
found := FALSE;
while (index <= NUMBER_OF_VERTICES) and (not found) do
begin
if octagon_array[index] = target then
found := TRUE;
inc(index);
end;
find_index := index - 1;
end;

function increment_counter(digit_1: integer; digit_2: integer): boolean;
var
increment_successful: boolean;
begin
increment_successful := TRUE;
inc(counter_array[digit_1][digit_2]);
if counter_array[digit_1][digit_2] >= MAX_CUT then
increment_successful := FALSE;
increment_counter := increment_successful;
end;

function do_counters_between(octagon_index: integer; stop_vertex: integer): boolean;
var
temp: integer;
go: boolean;
begin
  go := TRUE;
  while (octagon_array[octagon_index] <> stop_vertex) and (go) do
    begin
      temp_counter.point_1 := octagon_array[octagon_index];
      inc(octagon_index);
      temp_counter.point_2 := octagon_array[octagon_index];
      if temp_counter.point_2 < temp_counter.point_1 then
        begin
          temp := temp_counter.point_1;
          temp_counter.point_1 := temp_counter.point_2;
          temp_counter.point_2 := temp;
        end;
        go := increment_counter(temp_counter.point_1,
                                  temp_counter.point_2);
      end;
      do_counters_between := go;
  end;
end;

function go_thru_edges: boolean;
var
  start_index, edge_num: integer;
  proceed: boolean;
begin
  zero counters;
  edge_num := 1;
  proceed := TRUE;
  while (edge_num <= NUMBER_OF_EDGES) and (proceed) do
    begin
      if edges_array[edge_num].direction = FORWARD then
        begin
          start_index := find_index(edges_array[edge_num].point_1);
          proceed := do_counters_between(start_index,
                                          edges_array[edge_num].point_2);
        end;
      else
        begin
          start_index := find_index(edges_array[edge_num].point_2);
          proceed := do_counters_between(start_index,
                                          edges_array[edge_num].point_1);
        end;
        inc(edge_num);
    end;
go_thru_edges := proceed;
end;
procedure increment_binary(index: integer);
begin
  if edges_array[index].direction = FORWARD then
    edges_array[index].direction := BACKWARD
  else
    begin
      edges_array[index].direction := FORWARD;
      increment_binary(index - 1);
    end;
end;

procedure calculate_cut;
var
  count, smallest_cut: integer;
  print: boolean;
  percent_done: real;
begin
  zero_configuration;
  percent_done := (ACTUAL_PERMS / EXPECTED_PERMS) * 100;
  writeln(' % ', round(percent_done));
  count := 1;
  while (count <= (NUMBER_OF_CONFIGS - 1)) do
begin
  print := go_thru_edges;
  if print then
    print results;
  inc(ACTUAL_CASES);
  increment_binary(DIGITS_OF_BINARY);
  inc(count);
end;
  inc(ACTUAL_CASES);
  print := go_thru_edges;
  if print then
    print_results;
end;

procedure next_perm;
var
  temp: array[1..NUMBER_OF_VERTICES] of integer;
  i;
  h;
  index;
  temp_int;
  max_num;
  max_index;
  times: integer;
  done: boolean;
begin
  done := FALSE;
  for i := (NUMBER_OF_VERTICES - 1) downto 2 do
begin
if (perm[i] < perm[i+1] and not done) then
begin
  max_num := NUMBER_OF_VERTICES + 1;
  for h := NUMBER_OF_VERTICES downto (i+1) do
    begin
      if (perm[i] < perm[h]) and (perm[h] < max_num) then
        begin
          max_num := perm[h];
          max_index := h;
        end;
    end;
  temp_int := perm[i];
  perm[i] := perm[max_index];
  perm[max_index] := temp_int;
  for h := i+1 to NUMBER_OF_VERTICES do
    begin
      temp[h] := perm[h];
    end;
  end;
end;
begin
  clrscr;
  if open_data_file(output_file) = TRUE then
  begin
    writeln('Data File Opened - Run is Successful');
    initialize_vars;
    set_octagon_array;
    inc(ACTUAL_PERMS);
    calculate_cut;
    for count := 2 to EXPECTED_PERMS do
      begin
        next_perm;
        set_octagon_array;
        calculate_cut;
      end;
    writeln('Expected Permutations: ', EXPECTED_PERMS);
    writeln(output_file, 'Expected Permutations: ',
      EXPECTED_PERMS);
    writeln('Actual Permutations: ', ACTUAL_PERMS);
    writeln(output_file, 'Actual Permutations: ',
      ACTUAL_PERMS);
    writeln('Expected Cases: ', EXPECTED_CASES);
    writeln(output_file, 'Expected Cases: ',
      EXPECTED_CASES);
    writeln('Actual Cases: ', ACTUAL_CASES);
  end;
end;
writeln(output_file, 'Actual Cases: ', ACTUAL_CASES);
close(output_file);
end;
else
  writeln('Error - Data Files Not Opened');
  writeln('Press the -Enter- key to end the program');
  readln;
end.
APPENDIX B

DRAWINGS OF THE 96 MINIMUM CYCLIC CUTWIDTH SOLUTIONS

The drawings of the 96 minimum cyclic cutwidth solutions when a three dimensional cube is embedded onto a cycle in the form of a regular octagon are listed in order from the smallest numerical value of the permutation of the vertices to the largest numerical value of the permutation of the vertices. Rotating the octagon by $\pi/4$ or any of its multiples does not change the value of the minimum cyclic cutwidth, thus vertex 1 is always drawn at the top of the octagon with no loss of generality. The solution numbers in Table 1 are the same as the drawing numbers found to the top left of each drawing. The characters [], {}, <>, () listed after the drawing number correspond to the types of solution that are described in section 1-6, and show which type of solution is drawn.
Solution #1 []

Solution #2 {}

Solution #3 {}

Solution #4 ()

Solution #5 {}

Solution #6 ()

34
Solution #25 <>

Solution #26 {}

Solution #27 {}

Solution #28 ()

Solution #29 {}

Solution #30 ()
Solution #43 {}

Solution #44 []

Solution #45 {}

Solution #46 ()

Solution #47 []

Solution #48 {}
Solution #79

Solution #80

Solution #81

Solution #82

Solution #83

Solution #84
Solution #91

Solution #92

Solution #93

Solution #94

Solution #95

Solution #96
BIBLIOGRAPHY


