INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

NOTE TO USERS

The original manuscript received by UMI contains pages with slanted print. Pages were microfilmed as received.

This reproduction is the best copy available

UMI

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

STUDIES IN ALGORITHMIC GRAPH THEORY

by

Maria Belianina

A dissertation submitted to the Graduate Faculty in Mathematics in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1999

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 9917628

Copyright 1999 by
Belianina, Maria

All rights reserved.

UMI Microform 9917628
Copyright 1999, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

COPYRIGHT

1999

MARIA BELIANINA

All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This manuscript has been read and accepted by the Graduate Faculty in Mathematics in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

1/24/99
Date

[signature]
Chair of Examining Committee

1/25/99
Date

Józef Dodziuk (AZ)
Executive Officer

Michael Anshel

Joseph Malkevich

Burton Randol

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

ALGORITHMS FOR CUBIC GRAPHS

by

Maria Belianina

Adviser: Professor Michael Anshel

Recently, L.H.Kauffman has introduced a *planar cubic graph* representation for the so called *association equation*. In addition, he has proven the equivalence between the *Four Color Theorem* (*FCT*) applied to a planar cubic graph and the existence of the solution for the corresponding *association equation* in a *Vector Cross Product Algebra*.

This thesis introduces an algorithm (AET) for transforming a *planar cubic graph* with described properties into a graph form corresponding to a single *association equation*. The theorems presented show the existence of such forms for planar cubic graphs whose dual graphs possess a *Hamiltonian Cycle*.

The method presented transforms a given *Hamiltonian Cycle* in the *dual* of the given graph into a closed curve in the original graph. This curve passes through each *face* of the original graph exactly once and does not touch any of its vertices. Next, it is demonstrated how to use the obtained closed curve to construct several "association equation - friendly" graph forms of the original

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

graph. All of these forms are shown to correspond to a unique association equation which may take several equivalent forms of its own.

The thesis starts with discussion of an important algorithm (HCT) which transforms a given Hamiltonian Cycle into one or more Hamiltonian Cycles of the same graph. This algorithm generalizes the basic transformation technique of Papadimitriou's Hamiltonian Cycle transformation algorithm for cubic graphs. The paper presents a detailed proof of the algorithm's convergence. Applied to Kauffman's $K - maps$, each newly found Hamiltonian cycle (using the previously described AET method) produces another view o f the same graph corresponding to a possibly new *association equation*. This algorithm is a *non-deterministic* one and can be *randomized* as shown.

Next, an auxiliary notion of an *almost cubic graph* is introduced and an algorithm (RA) is presented for reducing any graph to an *almost cubic* one. It is also shown that the HCT method applied to an *almost cubic graph* becomes more efficient and possibly even *deterministic*.

This work has also opened an interesting and challenging problem for future research: an algebraic analysis of the suggested HCT algorithm. It appears that using the operations on finite symmetry groups described in the thesis may be the direction to pursue.

Finally, new unconventional approaches to the Hamiltonian cycle problems are described in the section on DNA Computing, a new field on the edge of mathematics, computer science and molecular biology. The section presents a computer simulation of the biolab computing test on finding a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Hamiltonian cycle in a graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

I would like to thank, first and foremost, Professor Michael Anshel for being the greatest teacher and advisor I have ever met. I would like to thank Professor Burton Randol for his so much needed encouragement and inspiration through all my years of study at the Graduate School. I would like to thank Professor Joseph Malkevitch for taking time to revise this paper very carefully and providing me with deep new insights into Graph Theory.

I would also like to thank very much all of the other professors of the Graduate School Department of Mathematics who introduced me into the breathtaking world of real and abstract mathematics and scientific research.

The support received from my great mom, Marina Belianina, and all of my family and friends I could not possibly acknowledge enough.

This work is dedicated to the memory of my wonderful beloved father, Doctor Iosif Zajdenman. I owe him everything. He taught me how to love mathematics and how to appreciate life.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

1 **Introduction.** **1**

2 **Terminology: graphs, colorings, and algorithms.** **6**

2.1 Graphs. 6

2.2 Walks, paths, cycles. 8

2.3 Colorings. 9

2.4 Algorithms and problems. 10

3 **Analysis of the Hamiltonian cycle transformation algorithm.** **11**

3.1 General facts. 11

3.2 Cubic planar graphs. 12

4 **On a generalization of the transformation algorithm for any graph.** **16**

4.1 Description. 16

4.2 An example and some experimental results. 17

4.3 The advantages of the algorithm for a general graph. . . 24

5 **Reducing a graph to almost cubic.** **25**

5.1 General Reduction Algorithm. 26

5.2 Reduction algorithm preserving a given Hamiltonian cycle. 29

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Applying the transformation algorithm to almost cubic graphs. 31

6 Randomized transformation algorithm for general graphs. 32

6.1 Description. 32

6.2 Algorithm 33

6.3 Program Output Example 1. 34

6.4 Program Output Example 2. 35

7 4-Color Conjecture equivalent form by Whitney. 37

8 Notes on algebra and association equations. 39

8.1 Vector Cross Product Algebra. 39

8.2 Geometric representation of an association. 41

8.3 Geometric representation of an association equation. . 42

9 Different forms of a given K-map. 44

10 Creating a K-map from a planar cubic graph. 47

11 Example. 55

11.1 Reducing a graph to an almost cubic graph. 55

11.2 Transformation of a given Hamiltonian cycle in the reduced graph. 60

11.3 Creating K-maps and solving the corresponding association equations. 63

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

12 Unconventional approach to the Hamiltonian cycle problem: molecular computing. 68

12.1 Introduction. 68

12.2 Possible conventional computer simulation of the bio-lab experiments. 69

A Program Code for the Randomized Transformation Algorithm. 76

B Detailed program output 1. 80

C Detailed program output 2. 85

D Mr. L. Adleman's Algorithm ("Mr. Adleman's Seven Days"). 92

E Example of a simulation program code. 97

References 100

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1 Introduction.

The problem of finding a Hamiltonian cycle in graphs is known to have attracted mathematicians since the mid-19th century, when Sir William Rowan Hamilton first tried to popularize the exercise of finding a closed path in the graph of a dodecahedron, i.e. of finding a way to walk through the dodecahedron along its edges, passing each vertex exactly once, and returning to the starting vertex in the end. There is still no efficient algorithm that will allow solving the problem for a sufficiently large graph.

However, the use of Hamiltonian cycles in different important applications gave rise to a new set of techniques that simplify the search for a Hamiltonian cycle, or at least provide some evidence that a Hamiltonian cycle may or may not exist in certain graphs. For example, it was proved that any graph with all the vertices having the same odd number of edges adjacent to them (i.e., having the same odd degree) have an even number of Hamiltonian cycles through the given edge. The latter, in turn, implied the Smith Computational Problem: given a Hamiltonian cycle in a graph , find another one (see [5]). It is clear that theoretically this problem may be solved provided there exists another Hamiltonian cycle.

There is only one algorithm known so far that actually takes a given cycle passing through the given edge and transforms it into another Hamiltonian cycle through the same edge in a completely defined way. The algorithm was suggested by Christos H. Papadimitriou (see [6] and [5]). However, the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithm was shown to apply only to cubic graphs in which each vertex has degree three. Section 2 describes the algorithm in detail.

This paper addresses two problems. One of them is an extension to Papadimitriou's algorithm. First of all, an extended proof of the Papadimitriou's algorithm is given in Section 3. Section 5 shows that the algorithm still works in a completely defined (or deterministic) way if the graph has all but one of its vertices of degree three.

Secondly, Section 5 discusses reducing a given graph to a so called *almost cubic* graph by deleting certain edges. Almost cubic graphs are defined to have vertices of degree greater than three, and there are no vertices of degree two. However, all the vertices of degree greater than three have only vertices of degree three as neighbors. This means that the almost cubic graph is extremal in the sense that deleting any edge creates a vertex of degree two and the graph is no longer almost cubic.

It is shown in Section 5 that a slightly modified version of the Hamiltonian cycle transformation algorithm applied to an almost cubic graph may be efficient and possibly even deterministic.

Section 11 suggests a random version of the transformation algorithm. If there is more than one choice for the next step, the algorithm makes a random choice. The random algorithm was tested and the detailed output is given in Appendices B and C.

Sections 3 and 4 show some analysis of the transformation algorithm. The topic is planned for future study by the author.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Section 7 through Section 11 discuss an example of a possible application of the transformation algorithm. The application arises from one of the many versions of the Four Color Theorem (see [8] and [9]). One of the latest statements is suggested by Louis H. Kauffman (see [4]). Kauffman shows that the Four Color Theorem is equivalent to a combinatorial problem about the vector cross product algebra in three-dimensional space (see Section 8). Kauffman considers insiders a cubic planar graph drawn on a plane in a special way (called a $K - map$ in this paper), which represents an equation in the vector cross product algebra. Then he proves equivalence between the existence of the graph edge three-coloring and the existence of a solution to the equation.

This paper extends the results of Kauffman's proof in the following way. First of all, it is shown in Sections 9 and 10 that, given a cubic graph on n vertices, the corresponding $K - map$ forms depend on (1) the closed curve passing through each face of the graph exactly once (some times in the literature such a curve is called a "face circuit") and leaving exactly $\frac{n}{2}$ vertices on each side , and (2) given the curve described in (1), the $n + 1$ different ways to draw the corresponding $K - map$. This implies that given a planar cubic graph on n vertices, which is already edge-colored in three colors, one immediately obtains solutions to all distinct equations in the vector cross product algebra, each one of which corresponds to one particular $K - map$.

Section 9 actually suggests a method of putting planar cubic graphs into Kauffman's cubic graph form. The closed curve passing through each face

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the graph exactly once and leaving $\frac{n}{2}$ vertices on each side corresponds to the Hamiltonian cycle in the dual of a given graph leaving exactly $\frac{n}{2}$ faces on each side. Once the cycle is found, it is "translated" back into the closed curve in the original graph. Then there are $n+1$ ways to draw a $K - map$.

The dual to a cubic graph is always a triangulation, i.e., a planar graph each face of which has exactly three sides. It is proved in Lemma 10.2 that a Hamiltonian cycle in any triangulation leaves exactly half of the faces on each side. The lemma implies that it is sufficient to find any Hamiltonian cycle in the dual of a given cubic planar graph in order to construct the corresponding $K - map$ (actually, $n+1$ of them.)

Further, if the Hamiltonian cycle transformation algorithm is applied to the cycle in a given graph, it may produce another Hamiltonian cycle that gives rise to another $n+1$ $K - maps$ of a given planar cubic graph. Thus one may reduce a graph to almost cubic keeping a given Hamiltonian cycle (see Section 5) and then transform this Hamiltonian cycle into another one.

This implies that finding as many Hamiltonian cycles as possible (say, x) in the dual of a given graph on n vertices through transformations of a given Hamiltonian cycle produces $x(n+1)$ $K - maps$ (not necessarily distinct), each one producing an equation in the vector cross product algebra. Given an edge three-coloring of a given graph, one immediately obtains solutions to all these equations.

Section 11 contains an example of a cubic planar graph G on 14 vertices and a given Hamiltonian cycle in the graph. First, the dual of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

graph is reduced to almost cubic with the given Hamiltonian cycle. Then the Hamiltonian cycle is transformed into another one using the transformation algorithm. Then the Hamiltonian cycles are "translated" into the face circuits in G and two different $K - maps$ are constructed with corresponding equations, one for each face circuit. Further, it is shown how the edge three -coloring of G implies solutions to the equations.

Section 12 shows some unconventional approaches to the Hamiltonian cycle problems. DNA Computations, or Molecular Computing, a new field of research that brings together mathematicians, computer scientists and biologists, literally uses the DNA molecules and their interaction in the biolab tubes in order to simulate an exhaustive search for hard problems. One of the goals is to create efficient "DNA algorithms" for solving difficult combinatorial problems such as finding Hamiltonian cycles and Hamiltonian paths, and many more. The section discusses the benefits that these algorithms can gain from the theorems on the number of Hamiltonian cycles in a graph. The section includes a description of the computer simulation of the DNA algorithm for finding a Hamiltonian cycle. Appendix D contains the program code. Appendix E contains a description of the original laboratory experiment performed by Leonard Adleman (see [1]), who solved a case of the Traveling Salesman problem, i.e. a problem of finding a Hamiltonian path in a directed graph with given starting and ending vertices in a graph on 7 vertices. Besides, Appendix E presents an extension of Adleman's experiment for a case of a Hamiltonian cycle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2 Terminology: graphs, colorings, and algorithms.

2.1 Graphs.

A *graph* $G(V, E)$ consists of a set of *vertices* V, or $V(G)$, a set of *edges* E, or $E(G)$, and a mapping associating to each edge e in E an unordered pair (v_1, v_2) of vertices called the *endpoints* of e. Two vertices are called *adjacent* if they are distinct and joined by an edge. An edge is said to be *incident* with its endpoints. An edge (v, v) is called a *loop*. The *degree*, or *valency* of a vertex v (usually denoted $deg(v)$) is equal to the number of edges in the graph incident with v. In a *d-regular* graph all the vertices have the same degree d.

A graph is called *finite* if it has a finite number of vertices.

The **"first theorem of graph theory"**:

$$\sum_{x \in V(G)} = 2|E(G)|.$$

Corollary: *A finite graph has an even number of vertices with odd degree.*

A *cubic graph* is a graph $G(V, E)$ in which each vertex has degree three, i.e. it is a *3-regular* graph. Note that $|E| = \frac{3|V|}{2}$, hence, a number of vertices $|V|$ must be even.

Two distinct nonloop edges with the same endpoints are called *parallel*. A graph is *simple* if it has no loops and no parallel edges. All the graphs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

considered are assumed to be *simple.*

A graph is called *directed* or a *digraph* if an ordered pair of vertices is associated to each edge.

A graph can be represented by a *drawing* such that each vertex is represented by a point in the plain, and each edge by a line segment or arc joining the corresponding endpoints.

A *planar graph* is a graph that can be drawn in the plane so that no two edges cross. A *maximal planar graph* is a planar graph such that addition of any new edge makes it non-planar.

A *map*, or *a planar map* M consists of a connected planar graph G together with a particular drawing or embedding of G in the plane. G is called the *underlying graph* of $M(G)$ or M. The map M divides the plane into connected components called *regions*, or *faces*, or *countries* of the *map*. Two *regions* are *adjacent* if their boundaries have at least one common edge.

A map is a *triangulation* if all its faces are triangles.

Note: One graph may be embedded in the plane to produce several different maps.

Two *graphs are isomorphic* if there is a one-to-one correspondence between the vertex sets such that if two vertices are joined by an edge in one graph, then the corresponding vertices are joined by an edge in the other graph.

Two *maps are isomorphic* if there is a continuous deformation of the plane onto itself carrying the vertices, edges, and regions of one map onto

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the vertices, edges, and regions of the other.

The *Dual Graph* $D(G)$ of a given graph G:

- Place a point, or vertex, in the middle of each region of a graph G.
- Join two vertices with a line, or edge, whenever the two regions have a common border.

The graph constructed in such a way is called they *dual graph* $D(G)$.

A *subgraph* of a graph G is a graph H such that $V(H) \subseteq E(G)$, $E(H) \subseteq E(G)$, and the ends of an edge $e \in E(H)$ are the same as its ends in G. H is a *spanning subgraph* when $V(H) = V(G)$.

2.2 Walks, paths, cycles.

A *walk* in a simple graph is a sequence of vertices, any two successive elements of which are adjacent. It can be denoted by $(x_0, x_1, x_2, \ldots, x_k)$ where $x_i \in V(G)$ for i from 0 to k. If all the edges used in a walk $(x_0, x_1, x_2, \ldots, x_k)$ are distinct, then the walk is called a *path* from x_0 to x_k. The number of edges passed in a walk equal to the *length* of a walk.

A walk (or path) is called *closed* if $x_0 = x_k$. A *simple path* is a path in which the vertices are distinct except possibly for $x_0 = x_k$. If $x_0 = x_k$ when $k \geq 1$ in a simple path, then it is called a *simple closed path.*

A Hamiltonian path (HP) is a path along the edges of a graph visiting each vertex of a graph exactly once.

A Hamiltonian cycle (HC) is a Hamiltonian path that starts and terminates at the same vertex.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The length of the shortest walk from the vertex x to the vertex y is called the *distance* $d(x,y)$ between these vertices. Such a shortest walk is necessarily a simple path.

A graph is *connected* if a path from x to y exists for every pair of vertices x, y of G. Otherwise G consists of a number of *connected components*, or *maximal connected subgraphs.*

A *polygon* is a finite connected 2-regular graph, i.e. the graph of a simple closed path.

A connected graph that contains no simple closed paths, i.e. that has no polygons as subgraphs, is called a *tree.*

2.3 Colorings.

For an integer k, a *k-coloring* of a graph G (also called a *vertex k-coloring*) is a mapping $f : V(G) \to \{1, \ldots, k\}$ such that $f(u) \neq f(v)$ for every edge of G with endpoints u and v.

For an integer k, an *edge k-coloring* of a graph G is a mapping $f : E(G) \to \{1, \ldots, k\}$ such that all the edges incident with the same vertex receive different colors, i.e. for any pair of edges e_1 and e_1 incident with the same vertex, $f(e_1) \neq f(e_2)$.

A *Tait-coloring* of a cubic graph or a cubic map is a 3-coloring of its edges.

For an integer k,a *k-coloring of a map* (some times called a *proper k-coloring*) is an assignment of k colors to the countries of the map in such a way that no two adjacent countries receive the same color.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Four Color Conjecture (4CC), (Francis Guthrie):

Every planar map can be four-colored.

Note: The *4CC* is considered by most mathematicians to have been proved by Kenneth Appel and Wolfgang Haken in 1986, using computer-assisted mathematics, and later by Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Hence, in case one believes the proofs, the "conjecture" becomes a "theorem", *4CT*.

2.4 Algorithms and problems.

An algorithm is called *deterministic* if at each step during execution the next step is defined in a unique way.

An algorithm is called *nondeterministic* if at some step during execution there are at least two choices for the next algorithm step.

A *randomized algorithm* is an algorithm that makes random choices during execution. (See Randomized Algorithms by R.Motwani and P. Raghavan, Cambridge University Press 1995.)

An algorithm is called *finite* if it can be proved that it will terminate in a finite number of steps on all possible inputs.

An algorithm is called *infinite*, or *possibly infinite* if there is an input on which it will not terminate.

A decision problem is a problem requiring only a "yes" or "no" answer.

A function problem is a problem, requiring an answer more elaborate than "yes" or "no" (*e.g.*, the Travelling Salesman problem searching for the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

optimal tour.)

3 Analysis of the Hamiltonian cycle transformation algorithm.

3.1 General facts.

Given a graph on $n+1$ vertices, consider an initial Hamiltonian cycle $HC_1 = [s, 1, 2, \ldots, n]$ where s is the starting vertex. Since the algorithm fixes the starting vertex of a cycle, one may consider only permutations of the remaining n vertices. Permutations of HC_1 produced in a sequence of transformations will form a subset of the set of all $|S_n| = n!$ permutations of n elements. Note that there are $|S_{n-1}| = (n-1)!$ permutations of n elements fixing the last element and, therefore, f forming a set containing all answers. Hence, a set of all intermediate permutations is of size at most

$$n! - (n-1)! = (n-1)(n-1)!$$

Each transformation step rewrites backwards the end of a path of certain length k, unique for each step and depending on the vertex adjacency. For example, in case vertex 5 is adjacent to vertices s, 4, and 2,

[s 1 2 3 4 5] can be rewritten as [s $\underline{5\ 4\ 3\ 2\ 1}$] or [$s$ 1 2 $\underline{5\ 4\ 3}$].

It corresponds to permutations: (1 5)(2 4) or (3 5).

Given a path $[s, 1, 2, \ldots, (n-2), (n-1), n]$, transformation permutations

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

rewriting backwards k last vertices can be described by the following table:

k : Permutation

$2 : (n-1, n)$

$3 : (n-2, n)$ **Fixes** $n-1$

$4 : (n-3, n)(n-2, n-1)$

$5 : (n-4, n)(n-3, n-1)$ **Fixes** $n-2$

$6 : (n-5, n)(n-4, n-1)(n-3, n-2)$

$7 : (n-6, n)(n-5, n-1)(n-4, n-2)$ **Fixes** $n-3$

...

$2t : (n-2t+1, n)(n-2t+2, n-1)(n-t, n-t+1)$

$2t+1 : (n-2t, n)(n-2t+1, n-1)(n-t-1, n-t+1)$ **Fixes** $n-t$

...

$n(even) : (1, n)(2, n-1)(\frac{n}{2}, \frac{n}{2}+1)$

$n(odd) : (1, n)(2, n-1)\ldots(\frac{n+1}{2}-1, \frac{n+1}{2}+1)$ **Fixes** $\frac{n+1}{2}$

3.2 Cubic planar graphs.

In a Cubic Planar graph a set of all intermediate permutations is much smaller than $(n-1)(n-1)!$ since, as is shown below, a lot of permutations of the original cycle are forbidden. Consider a cubic planar graph $G(V, E)$ on $n+1$ vertices and a given Hamiltonian cycle $HC_1 = [s, 1, 2, \ldots, n]$ where s is the starting vertex.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

General Smith problem ([5]): A graph with odd degrees has an even number of Hamiltonian cycles through any given edge.

Computational Smith problem ([5]): Given a graph G with odd degrees and a Hamiltonian cycle in it, find another one. Special case: G is a cubic graph.

Note that this is an example of a Total Function Problem (TFNP), i.e., a Function Problem having positive answer for each input string. The solution to these problems relies on the *Parity Argument*:

Any finite graph has an even number of odd-degree nodes.

(Since $|E| = \frac{\text{Sum of } deg(v_i)}{2}$, the sum of $deg(v_i)$ is an even number. Hence, the number of odd degrees is even.)

Theorem 3.1 ([6], p.232): *If a cubic graph G has a Hamiltonian cycle, then it must have a second one as well.*

Proof (extended version):

Given: G - a cubic graph on n vertices ($deg(v) = 3$ for all v).

$[1, n, n-1, \ldots, 2, 1]$ a Hamiltonian cycle in G with n edges.

Find: Another Hamiltonian cycle through the edge $(1, 2)$.

Algorithm

- Delete the edge $(1, 2)$ to obtain a Hamiltonian path $P_1 = [1, n, n-1, \ldots, 2]$ ($n - 1$ edges)
- Consider the paths P_i starting with node 1 and not using the edge $(1, 2)$ (call such paths *candidates*).
- Call two candidate paths *neighbors* if they have $n - 2$ edges in common.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

• Create a sequence of candidate neighbors $P_1, P_2, \ldots, P_i, P_{i+1}, \ldots$

Question: How many neighbors does a candidate path P_i have ?

If the endpoint x is not node 2, then one can obtain two distinct candidate neighbors P_{i-1} and P_{i+1} by "rotating" the path:

• add to the path an edge out of x that is not currently in the path (recall $deg(x) = 3$) and

• break the new cycle through x, deleting another edge out of x.

Note. The described steps can be visualised as a "rotation" of the path's "tail" at the node x. The new "tail" will pass through the same vertices but in the opposite order.

If the endpoint x is 2, then the edge $(1, 2)$ cannot be used. Hence, a path with endpoints 1 and 2 has only one neighbor.

$\Rightarrow$ Two consecutive paths P_i,P_{i+1} in the sequence of "rotations" produce a unique next path, another candidate neighbor of the middle path. I.e., *for any two consecutive paths P_i , P_{i+1} the sequences $P_1, \ldots, P_{i-1}$ and $P_{i+2}, \ldots$ are unique.*

The process will terminate according to the Parity Argument, or because there is a finite number of Hamiltonian paths.

Question: Why can the process not cycle ?

Suppose there is a path A that appears in a sequence of transformations twice before the solution (another HC) is found:

$$P_1, \ldots, A, \ldots, A, \ldots.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Each path has exactly two transformation neighbors. Set B and C be two neighbor paths of A. Then there are two possibilities:

Case 1. $P_1, \ldots, B, A, C, \ldots, C, A, B, \ldots$ or

Case 2. $P_1, \ldots, B, A, C, \ldots, B, A, C, \ldots$.

Case 1. According to (*), if one continues the transformations filling a gap $B, A, C, \ldots, C, A, B$, it will result in one of two palindromes below:

i. $\ldots, B, A, C, \ldots, Y, X, Y, \ldots, C, A, B, \ldots$ - impossible.

ii.$\ldots, B, A, C, \ldots, Y, X, X, Y, \ldots, C, A, B, \ldots$ - impossible.

Case 2. $P_1, \ldots, B, A, C, \ldots, B, A, C, \ldots$.

Case 2a:

$A = P_1$, i.e. there is a sequence $P_1, C, \ldots, B, P_1, C, \ldots$.

However, the original path P_1 has only one neighbor (see...). Hence, both C and B cannot be used. Hence, the original path P_1 cannot repeat in the transformation sequence.

Case 2b:

$P_1, \ldots, B, A, C, \ldots, B, A, C, \ldots$ where $A \neq P_1$.

Set $B = P_{i-1} = P_{i-1+k}$, $A = P_i = P_{i-1+k}$, $C = P_{i+1} = P_{i-1+k}$, and consider a sequence

$$P_1, \ldots, P_{i-1}, P_i, P_{i+1}, \ldots, P_{i-1+k}, P_{i+k}, P_{i+1+k}, \ldots$$

According to (*), the pairs of neighbor paths $\{B, A\} = \{P_{i-1}, P_i\} = \{P_{i-1+k}, P_{i+k}\}$ produce identical sequences of $i-1$ paths towards P_1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Hence, $P_1 = P_{i+k-(i-1)} = P_{k+1}$. Since $k+1 < i+k$, a path P_1 will appear in the sequence before P_{i-1+k}(= the second occurrence of B) resulting in the **Case 2a** that is impossible.

Hence, *the transformation process can neither cycle, nor run into the same original path.*

Q.E.D.

Therefore, the process will terminate in a Hamiltonian path with an end-point 2 since the number of distinct paths is finite and the process cannot cycle.

4 On a generalization of the transformation algorithm for any graph.

4.1 Description.

Consider a complete graph G on $n+1$ vertices with given Hamiltonian cycle $HC_1 = [s, 1, 2, \ldots, n]$. Fix a vertex s and an adjacent edge (s, n). Suppose $d_i = deg(v_i)$. Start transforming a cycle according to the transformation algorithm for a cubic graph. If $d_n > 3$, then at the very first step the algorithm becomes non-deterministic: it does not know which edge out of $d-2$ available edges adjacent to n to add to the path next (note that edges $(n-1, n)$ and (s, n) are not available.)

To overcome that one has the following choices:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. Trace all arising sequences of transformation paths in parallel;
2. Pick the next transformation according to a certain rule;
3. Pick the next transformation randomly.

Before picking any particular strategy, let's analyze the action of transformations on a given path.

4.2 An example and some experimental results.

Consider S_5, the symmetric group of all permutations of 5 elements $\{1,2,3,4,5\}$. Starting with a sequence (or path) $[s, h_1h_2h_3h_4h_5] = [s,1,2,3,4,5]$, consider all its $5! = 120$ permutations fixing the first element s (see the first 6 columns of the table below) and enumerate them (see the column 'Path #'). Set

$$T_2(h_1h_2h_3\underline{h_4h_5}) = h_1h_2h_3\underline{h_5h_4}$$

$$T_3(h_1h_2\underline{h_3h_4h_5}) = h_1h_2\underline{h_5h_4h_3}$$

$$T_4(h_1\underline{h_2h_3h_4h_5}) = h_1\underline{h_5h_4h_3h_2}$$

$$T_5(\underline{h_1h_2h_3h_4h_5}) = \underline{h_5h_4h_3h_2h_1}$$

Then the following table represents the right multiplication table for S_5 and a set of transformations T_2, T_3, T_4, T_5:

Notes on Fig.4.1 below:

- There are $(5-1)! = 24$ paths having 5 on the last position (i.e., $h_5 = 5$).
- Each path with 5 not on the last position (i.e., $h_5 \neq 5$) can be transferred

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to such a path by one and only one of the transformations T_i.

• The last (5-1)!=4!=24 paths (#97-#120) have $h_0 = s$ and $h_1 = 5$. Such paths would represent forbidden transformation paths in a graph where edge $(s, 5)$ is fixed and cannot be used. In the remaining part of the table (#1-#96) such forbidden paths appear only in the last column T_5 as $T_5(h_1 h_2 h_3 h_4 5)$ where $[h_1 h_2 h_3 h_4 5]$ is a path with $h_5 = 5$.

• Each one of the remaining $(5-2)(5-1)! = 3 \times 4! = 3 \times 24 = 72$ paths can be either transformed into:

- a path with $h_5 = 5$

or

- one of other 5-2=3 transformable paths.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.4.1

h_0	h_1	h_2	h_3	h_4	h_5	*Path#*	T_2	T_3	T_4	T_5
s	1	2	3	4	5	1	2	6	24	120
s	1	2	3	5	4	2	1	4	18	96
s	1	2	4	3	5	3	4	5	22	114
s	1	2	4	5	3	4	3	2	12	72
s	1	2	5	3	4	5	6	3	16	90
s	1	2	5	4	3	6	5	1	10	66
s	1	3	2	4	5	7	8	12	23	118
s	1	3	2	5	4	8	7	10	17	94
s	1	3	4	2	5	9	10	11	20	108
s	1	3	4	5	2	10	9	8	6	48
s	1	3	5	2	4	11	12	9	14	84
s	1	3	5	4	2	12	11	7	4	42
s	1	4	2	3	5	13	14	18	21	112
s	1	4	2	5	3	14	13	16	11	70
s	1	4	3	2	5	15	16	17	19	106
s	1	4	3	5	2	16	15	14	5	46
s	1	4	5	2	3	17	18	15	8	60
s	1	4	5	3	2	18	17	13	2	36
s	1	5	2	3	4	19	20	24	15	88
s	1	5	2	4	3	20	19	22	9	64
s	1	5	3	2	4	21	22	23	13	82
s	1	5	3	4	2	22	21	20	3	40
s	1	5	4	2	3	23	24	21	7	58
s	1	5	4	3	2	24	23	19	1	34
s	2	1	3	4	5	25	26	30	48	119
s	2	1	3	5	4	26	25	28	42	95
s	2	1	4	3	5	27	28	29	46	113
s	2	1	4	5	3	28	27	26	36	71
s	2	1	5	3	4	29	30	27	40	89
s	2	1	5	4	3	30	29	25	34	65
s	2	3	1	4	5	31	32	36	47	116
s	2	3	1	5	4	32	31	34	41	92
s	2	3	4	1	5	33	34	35	44	102
s	2	3	4	5	1	34	33	32	30	24
s	2	3	5	1	4	35	36	33	38	78
s	2	3	5	4	1	36	35	31	28	18
s	2	4	1	3	5	37	38	42	45	110
s	2	4	1	5	3	38	37	40	35	68
s	2	4	3	1	5	39	40	41	43	100
s	2	4	3	5	1	40	39	38	29	22

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.4.1 cont.

h_0	h_1	h_2	h_3	h_4	h_5	$Path\#$	T_2	T_3	T_4	T_5
s	2	4	5	1	3	41	42	39	32	54
s	2	4	5	3	1	42	41	37	26	12
s	2	5	1	3	4	43	44	48	39	86
s	2	5	1	4	3	44	43	46	33	62
s	2	5	3	1	4	45	46	47	37	76
s	2	5	3	4	1	46	45	44	27	16
s	2	5	4	1	3	47	48	45	31	52
s	2	5	4	3	1	48	47	43	25	10
s	3	1	2	4	5	49	50	54	72	117
s	3	1	2	5	4	50	49	52	66	93
s	3	1	4	2	5	51	52	53	70	107
s	3	1	4	5	2	52	51	50	60	47
s	3	1	5	2	4	53	54	51	64	83
s	3	1	5	4	2	54	53	49	58	41
s	3	2	1	4	5	55	56	60	71	115
s	3	2	1	5	4	56	55	58	65	91
s	3	2	4	1	5	57	58	59	68	101
s	3	2	4	5	1	58	57	56	54	23
s	3	2	5	1	4	59	60	57	62	77
s	3	2	5	4	1	60	59	55	52	17
s	3	4	1	2	5	61	62	66	69	104
s	3	4	1	5	2	62	61	64	59	44
s	3	4	2	1	5	63	64	65	67	98
s	3	4	2	5	1	64	63	62	53	20
s	3	4	5	1	2	65	66	63	56	30
s	3	4	5	2	1	66	65	61	50	6
s	3	5	1	2	4	67	68	72	63	80
s	3	5	1	4	2	68	67	70	57	38
s	3	5	2	1	4	69	70	71	61	74
s	3	5	2	4	1	70	69	68	51	14
s	3	5	4	1	2	71	72	69	55	28
s	3	5	4	2	1	72	71	67	49	4
s	4	1	2	3	5	73	74	78	96	111
s	4	1	2	5	3	74	73	76	90	69
s	4	1	3	2	5	75	76	77	94	105
s	4	1	3	5	2	76	75	74	84	45
s	4	1	5	2	3	77	78	75	88	59
s	4	1	5	3	2	78	77	73	82	35
s	4	2	1	3	5	79	80	84	95	109
s	4	2	1	5	3	80	79	82	89	67

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.4.1 cont.

h_0	h_1	h_2	h_3	h_4	h_5	*Path#*	T_2	T_3	T_4	T_5
s	4	2	3	1	5	81	82	83	92	99
s	4	2	3	5	1	82	81	80	78	21
s	4	2	5	1	3	83	84	81	86	53
s	4	2	5	3	1	84	83	79	76	11
s	4	3	1	2	5	85	86	90	93	103
s	4	3	1	5	2	86	85	88	83	43
s	4	3	2	1	5	87	88	89	91	97
s	4	3	2	5	1	88	87	86	77	19
s	4	3	5	1	2	89	90	87	80	29
s	4	3	5	2	1	90	89	85	74	5
s	4	5	1	2	3	91	92	96	87	56
s	4	5	1	3	2	92	91	94	81	32
s	4	5	2	1	3	93	94	95	85	50
s	4	5	2	3	1	94	93	92	75	8
s	4	5	3	1	2	95	96	93	79	26
s	4	5	3	2	1	96	95	91	73	2
s	5	1	2	3	4	97	98	102	120	87
s	5	1	2	4	3	98	97	100	114	63
s	5	1	3	2	4	99	100	101	118	81
s	5	1	3	4	2	100	99	98	108	39
s	5	1	4	2	3	101	102	99	112	57
s	5	1	4	3	2	102	101	97	106	33
s	5	2	1	3	4	103	104	108	119	85
s	5	2	1	4	3	104	103	106	113	61
s	5	2	3	1	4	105	106	107	116	75
s	5	2	3	4	1	106	105	104	102	15
s	5	2	4	1	3	107	108	105	110	51
s	5	2	4	3	1	108	107	103	100	9
s	5	3	1	2	4	109	110	114	117	79
s	5	3	1	4	2	110	109	112	107	37
s	5	3	2	1	4	111	112	113	115	73
s	5	3	2	4	1	112	111	110	101	13
s	5	3	4	1	2	113	114	111	104	27
s	5	3	4	2	1	114	113	109	98	3
s	5	4	1	2	3	115	116	120	111	55
s	5	4	1	3	2	116	115	118	105	31
s	5	4	2	1	3	117	118	119	109	49
s	5	4	2	3	1	118	117	116	99	7
s	5	4	3	1	2	119	120	117	103	25
s	5	4	3	2	1	120	119	115	97	1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Hence, the sequence of transformations of the original cycle $(s, 1, \ldots, n)$ (here $n = 5$) resulting in a solution $(h_n = n)$ would correspond to a composition of T_i's.

Consider the corresponding multiplication graph $G(V, E)$ with $V = S_n = \{\pi_k\}$ and $E = \{ T_i \mid i = 2, \ldots, n \}$, such that $(\pi_i, \pi_j) \in E$ if and only if for some T_k

$$\pi_i T_k = \pi_j.$$

A path in such a graph written in edge names starting from a vertex with identical permutation $\pi_1 = 1$ would correspond to a sequence of transformations T_i of any given path on $n + 1$ vertices $[s, 1, \ldots, n]$. All such paths terminating in a vertex with a permutation fixing n would correspond to a solution. All cycles passing only through vertices *not* fixing n would correspond to infinite cycles in the algorithm. Suppose the longest such cycle of length L without a solution is found for a given n with the given restrictions on permutations. Then the algorithm does not cycle in L steps.

A search showed that the following sequences of transformations produce cycles:

- If applied to paths such that $h_2 \neq n$ and $h_3 \neq n$ then

$$(T_2 T_3)^3 = 1$$

corresponds to an elementary region in a graph on 6 sides.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- If applied to paths such that $h_3 \neq n$ then

$$(T_4T_2T_4T_3)^3 = 1$$

corresponds to an elementary region in a graph on 12 sides.

Studies showed also that the above mentioned cycles are the only no-solution cycles for a graph based on S_4. Each permutation in each cycle has three neighbors: two from a cycle and one a solution permutation. It means that in any graph on 5 vertices the transformation will terminate in at most 12 steps (12 for K_5).

The situation changes with S_5 where a lot of different cycles have been found so far (however, all except the ones mentioned above are longer than 12 edges).

Remark:

To find all possible solutions:

Start with the first path HP_1.

Step 1: Extend all transformation sequences by 1.

Repeat **Step 1** till all $(n-1)!$ solution paths are found.

Note. All upper boundaries come from considering a complete graph on $(n+1)$ vertices in which all the transformations are allowed at each step.

Among $n! - (n-1)!$ no-solution paths there are $(n-1)!$ forbidden paths

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

using the fixed edge. Hence, there are at most

$$n! - (n-1)! - (n-1)! = n! - 2(n-1)! = (n-2)(n-1)! \text{ no} - \text{solution paths.}$$

Remark:

Each solution can be produced by at most $(n-2)$ different transformations from at most $(n-2)$ distinct paths.

Conversely, all these paths can produce only one solution, each by a different transformation.

There are at most $(n-1)!$ solutions. Hence, there are at most $(n-1)!$ such sets.

Hence, to generate all solutions the algorithm has to be run until at least one representative of each such at most $(n-2)$-set is produced.

Hence, the longest no-solution transformation sequence is at most of length $(n-1)!$ that means that almost all solutions are guaranteed to appear in some acyclic sequence (if any) after $(n-1)!$ steps.

4.3 The advantages of the algorithm for a general graph.

Given a graph G on $n+1$ vertices and a Hamiltonian cycle HC_1 in it, pick a vertex of the highest degree $D = max(deg(v_i))$ to be a starting vertex s in the transformations. Fix an edge (s, n) and start transforming the resulting path $[s, 1, 2, \ldots, n]$. Such a strategy will increase the probability of getting extra Hamiltonian cycles not passing through the originally fixed edge at all.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since each step produces new Hamiltonian paths with the same starting vertex and new final vertices, the probability of each new path being a Hamiltonian cycle is $\frac{D}{n}$.

Besides, if an algorithm keeps track of all formed paths and eliminates transformation sequences that cycle without producing an answer, then all produced paths with a final vertex adjacent to the fixed starting vertex will form a set of distinct Hamiltonian cycles.

Changing the strategy a little bit and using the same starting vertex s of the highest degree D with vertices $(a_1, \ldots, a_D)$ adjacent to s and an algorithm transformation principle, one may start with inspecting the original cycle HC_1 in the following way:

Step 1: Pick a vertex a_i adjacent to s such that $a_i \neq 1$ and $a_i \neq n$.

Step 2: If a vertex $(a_i - 1)$ is adjacent to n then transform $[s, 1, 2, \ldots, n]$ into $[s, 1, \ldots, a_i - 1, n, n-1, \ldots, a_i]$ which is a Hamiltonian cycle.

Repeat steps 1 and 2 for all vertices adjacent to s.

The algorithm applied in a deterministic way is shown on Fig. 4.2. The algorithm involving some randomization is studied in Section 6 below.

5 Reducing a graph to almost cubic.

In an attempt to make a transformation algorithm for a general graph as close to deterministic as possible one may consider deleting some edges from the original graph in the way described below (see *Reduction Algorithm* 2)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and transforming a graph into *cubic* or almost cubic (see definition below). Clearly, it is possible to reduce a graph to cubic by deleting edges if and only if a graph $G(V, E)$ has a cubic subgraph on all $|V|$ vertices.

5.1 General Reduction Algorithm.

Definition: A graph is called *almost cubic* if

- all its vertex degrees are greater than 2 and
- deleting any edge will create a vertex of degree 2.

Consider a graph $G(V, E)$ on $|V|$ vertices $V = \{v_1, \ldots, v_{|V|}\}$ where each vertex v_i has degree $deg(v_i)$. Assign a *weight* $w(v_i, v_j)$ to each edge (v_i, v_j) equal to the minimal number among $deg(v_i)$ and $deg(v_j)$:

$$w(v_i, v_j) = min(deg(v_i), deg(v_j)).$$

Hence, all edge weights are always greater or equal to the smallest vertex degree in a graph.

Reduction Algorithm 1:

Step 1: Pick any edge of the highest weight greater than 3 and delete it.

Step 2: Adjust the corresponding vertex degrees and edge weights in the following way:

- decrease each $deg(v_i)$ and $deg(v_j)$ by one;
- for each edge (v_i, x) adjacent to v_i if $w(v_i, x) > deg(v_i)$ then reduce $w(v_i, x)$ by one;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

• for each edge (v_j, x) adjacent to v_j, if $w(v_j, x) > deg(v_j)$ then reduce $w(v_j, x)$ by one;

Step 3: Repeat steps 1 and 2 until all the edge weights are equal to 3.

Note 1. Step 1 is defined in order to make the algorithm more deterministic. It is an open question whether picking the smallest or any other edge weight greater than 3 will produce better results.

Note 2. The algorithm may produce a graph with more than one connected component. In case it does, all further arguments in this paragraph apply to each connected component separately.

Note 3. **Step 1** of the algorithm can be changed to **Step 1*** below so that the graph connectivity is preserved:

Step 1*: Pick any edge of the highest weight greater than 3 that does not disconnect the graph and delete it.

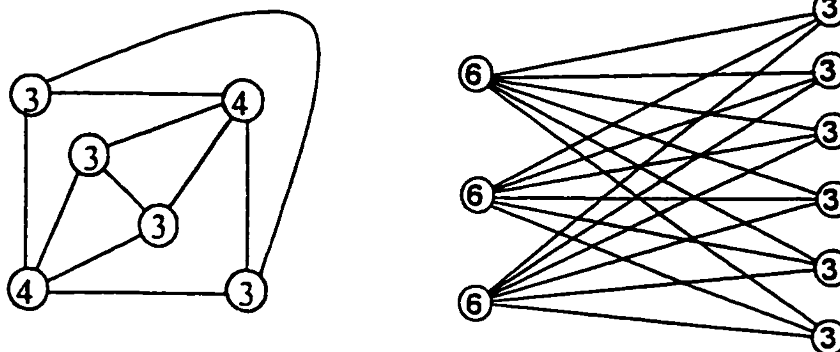
In this case the algorithm may produce a minimal connected graph closest to almost cubic. **Step 3** also has to be transformed to **Step 3*** below:

Step 3*: Repeat **Step 1*** and **Step 2** until all the edges that do not disconnect a graph have weights equal to 3.

In a formed graph each vertex x of degree $deg(x) > 3$ is surrounded only by vertices of degree 3. A vertex y of degree 3 can have both trivalent and other vertices adjacent to it. Hence, the number of trivalent vertices is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

greater or equal to the highest degree in a graph. See examples on Fig.5.1.



Note that since a triangulation is a maximal planar graph, the addition of one edge turns it into a non-planar graph. Hence, no planar non-reduced graph can be reduced by the algorithm to triangulation.

Consider a **triangulation** G_T embedded on the surface of a sphere. Pick a vertex x of any degree. Consider edges (x, a) and (x, b) so that a path $a - x - b$ is on the boundary of some elementary region. Since all elementary regions are triangles. vertices $\{a. x, b\}$ do form a triangle and there is an edge (a, b). Since it is true for all such consecutive pairs a, b of vertices adjacent to x. all vertices adjacent to x with edges connecting them form a closed simple path. Hence. the degree e of any vertex adjacent to x is at least 3.

Suppose $deg(x) = 3$. Then the vertices $\{a, b, c\}$ adjacent to x form a triangle. Suppose there is another vertex d adjacent to a such that $d - a - b$ is part of some elementary boundary. Then. by the same argument as for vertex x above. vertices $\{d, a, b\}$ form a triangle. That means that the degrees of both a and b are at least four. Hence, the weight $w(a, b) \geq 4$ and an edge $(a. b)$ will be deleted by Step 1 of the algorithm creating an elementary region

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with at least four sides.

Suppose $deg(x) = d > 3$. Then the vertices $\{a_1, \ldots, a_d\}$ adjacent to x form a polygon with d sides. Such a construction itself is not a triangulation due to the elementary region $[a_1, \ldots, a_d]$ that is not a triangle. However, if it is a part of the connected graph which is a result of reduction by the algorithm , for each i, $deg(a_i) = 3$, and no other vertices can be added.

Hence, *the algorithm cannot produce a triangulation*, or *the algorithm reduces any triangulation to a graph that is not a triangulation.*

Remark: In other words, the arguments above imply that in any triangulation with more than four vertices there is at least one pair of adjacent vertices both of degree greater than three.

5.2 Reduction algorithm preserving a given Hamiltonian cycle.

The algorithm of transforming a given Hamiltonian cycle into another one passing through the same edge is deterministic in the case of cubic graphs and most probably non-deterministic in the case of general graphs, since most certainly a path produced at a certain step admits more than one further transformation. Slightly changing the algorithm above and applying it to a general graph with a given Hamiltonian cycle so that no cycle edge is deleted can make the transformation algorithm work more efficiently.

Given: A graph $G(V, E)$ on $|V|$ vertices and a Hamiltonian cycle HC.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Goal: Find a subgraph of G on V that has the same Hamiltonian cycle.

Reduction Algorithm 2:

Step 1: Pick an edge that:

- has the highest weight greater than 3;
- does not belong to HC.

Delete it.

Step 2: Adjust the corresponding vertex degrees and edge weights in the following way:

- decrease each $deg(v_i)$ and $deg(v_j)$ by one;
- for each edge (v_i, x) adjacent to v_i, if $w(v_i, x) > deg(v_i)$ then reduce $w(v_i, x)$ by one;
- for each edge (v_j, x) adjacent to v_j, if $w(v_j, x) > deg(v_j)$ then reduce $w(v_j, x)$ by one;

Step 3: Repeat **Step 1** and **Step 2** till all the edges that do not belong to HC have weights equal to 3.

End of Algorithm 2.

Note 1: Since a Hamiltonian graph is always connected, **Reduction Algorithm 2** will always produce a connected graph.

For an example of the reduction algorithm see Section 9 below.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Applying the transformation algorithm to almost cubic graphs.

For an *almost cubic* graph with only one vertex of degree greater than three the transformation algorithm works exactly as in a cubic graph. Given a Hamiltonian cycle, set a vertex with degree greater than 3 as a starting vertex (Start) for further transformations. Fix any of two edges in the Hamiltonian cycle adjacent to the Start vertex. Transform a path. Since all the vertices that can become final in paths have degree 3, at each step the next transformation is unique. Besides, all the arguments used in the proof of the algorithm convergence in the case of cubic graphs apply. For an example of reducing a general graph to the almost cubic case see Example 1.

For all other almost cubic graphs the algorithm depends on the number of vertices of degrees greater than 3, and on the degrees themselves. If at a certain transformation step a Hamiltonian path final vertex has degree d, then the next transformation can be done in $d - 2$ ways (one edge is currently used, another edge was used at the previous step). The next step in the analysis of the algorithm for this case would be to study different kinds of *almost cubic* graphs arising in the reduction algorithm.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6 Randomized transformation algorithm for general graphs.

6.1 Description.

Since the transformation algorithm is most certainly non-deterministic in a general non-reduced graph, and the probability of producing a cycle distinct from the original cycle is relatively high, it seems natural to make the algorithm random.

Given:

• A planar graph $G(V, E)$ on $|V| = n + 1$ vertices with all vertex degrees $\geq$ 3;

• Vertex s in V of the highest degree $deg(s) = d$;

• A Hamiltonian cycle HC =[s,1,...,n].

Problem: Find another Hamiltonian cycle.

Consider a *random* form of the general transformation algorithm with the following changes:

• Instead of producing all possible transformations for a certain number of steps, the *random transformation algorithm* will randomly choose next transformation from the list of allowed transformations. This way it will work only with one sequence.

• Instead of looking for a Hamiltonian cycle passing through a fixed edge it will look for any Hamiltonian cycle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2 Algorithm

Step 1

Randomly pick a vertex $HP_0(t) = adj(HP_0(n)) = adj(n) = t$ from the list of vertices (if any):

- adjacent to $HP_0(n) = n$;
- other than s or $HP_0(n-1) = n-1$.

Step 2: Rewrite a path:

$HP_0(i)$ **where:**	**as** $HP_1(i)$ **where:**
$HP_0(0) = s$ $HP_0(1) = 1$ ... $HP_0(t) = t$	$HP_1(0) = s$ $HP_1(1) = 1$ ... $HP_1(t) = t$
$HP_0(adj(n)+1) = t+1$ $HP_0(adj(n)+2) = t+2$ ... $HP_0(n-1) = n-1$ $HP_0(n) = n$	$HP_1(t+1) = n$ $HP_1(t+2) = n-1$ ... $HP_1(n-1) = t+2$ $HP_1(n) = t+1$

Step 3: While there is no edge $(s, HP_k(n))$ repeat Step 4 and Step 5:

Step 4: Randomly pick a vertex $HP_k(t) = adj(HP_k(n))$ from the list of vertices

- adjacent to $HP_k(n)$ (i.e., the last vertex in a path $HP_k(i)$;
- other than $HP_k(n-1)$

  or

- $adj(HP_{k-1}(n))$ (to avoid the transformation reversing a previous step).

Step 5: Rewrite a path:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$HP_k(i)$ as $HP_{k+1}(i)$ where
$HP_{k+1}(0) = HP_k(0)$ $HP_{k+1}(1) = HP_k(1)$ ... $HP_{k+1}(t) = HP_k(t)$
$HP_{k+1}(t+1) = HP_k(n)$ $HP_{k+1}(t+2) = HP_k(n-1)$ ... $HP_{k+1}(n-1) = HP_k(t+2)$ $HP_{k+1}(n) = HP_k(t+1)$

See **Appendix A** for a code.

6.3 Program Output Example 1.

Given: A graph on 9 vertices with maximum vertex degree equal to 6.

Graph adjacency matrix:

Notes: The vertices are numbered 0 through 8. The first column cantains vertex numbers, the next six columns contain the numbers of the vertices adjacent to the vertex in the first column. The last column contains the vertex degree.

0	1	3	4	5	7	8	6
1	2	3	6	8	0	-	5
2	1	3	4	5	6	-	5
3	1	2	4	0	-	-	4
4	2	3	5	0	-	-	4
5	2	4	6	0	-	-	5
6	1	2	5	8	-	-	5
7	5	6	8	0	-	-	4
8	1	6	7	0	-	-	4

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(**Note for the program:** In Input file all empty cells should be filled with zeros.)

Sorted list of cycles produced after 49 program runs from the original cycle

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 8 7 (10 times)

0 1 8 7 6 2 3 4 5 (7 times)

0 1 8 7 6 5 2 3 4 (7 times)

0 1 8 7 6 5 4 2 3 (25 times)

(49 copies)

For detailed output showing all transformations in each run see **Appendix B.**

6.4 Program Output Example 2.

Given: A graph on 25 vertices with maximum vertex degree equal to 10. It is a dual graph to the Tutte cubic graph on 46 vertices

Notes: The vertices are numbered 0 through 24. The first column contains vertex numbers, the next ten columns contain the numbers for the vertices adjacent to the vertex in the first column. The last column contains the vertex degree.

Graph adjacency matrix:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

0	21	22	24	1	2	3	4	8	16	17	10
1	0	2	6	7	17	-	-	-	-	-	5
2	0	1	3	6	-	-	-	-	-	-	4
3	0	2	4	5	6	-	-	-	-	-	5
4	0	3	5	8	-	-	-	-	-	-	4
5	0	3	4	6	7	8	-	-	-	-	6
6	1	2	3	5	7	-	-	-	-	-	5
7	1	5	6	8	17	-	-	-	-	-	5
8	0	4	5	7	9	10	11	12	16	17	10
9	8	10	14	15	17	-	-	-	-	-	5
10	8	9	11	14	-	-	-	-	-	-	4
11	8	10	12	13	14	-	-	-	-	-	5
12	8	11	13	16	-	-	-	-	-	-	4
13	11	12	14	15	16	-	-	-	-	-	5
14	9	10	11	13	15	-	-	-	-	-	5
15	9	13	14	16	17	-	-	-	-	-	5
16	18	19	20	21	0	8	12	13	15	17	10
17	18	24	0	1	7	8	9	15	16	-	9
18	19	23	24	16	17	-	-	-	-	-	5
19	18	20	23	16	-	-	-	-	-	-	4
20	19	21	22	23	16	-	-	-	-	-	5
21	20	22	0	16	-	-	-	-	-	-	4
22	20	21	23	24	0	-	-	-	-	-	5
23	18	19	20	22	24	-	-	-	-	-	5
24	18	22	23	0	17	-	-	-	-	-	5

Sorted list of Hamiltonian cycles produced after 49 program runs from the original cycle:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 23 24 22 21 (2 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24 (6 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22 (11 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22 (16 copies)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17 (4 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 23 18 24 17 (3 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 23 24 18 17 (5 copies)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 24 23 18 17 (2 copies)

49 copies

For the detailed output showing all transformations in each run see **Appendix C.**

Remark: The output and transformation analysis suggest that a probability of getting new cycles with the same starting path of some length is higher than producing totally different paths. Hence, to produce other cycles one may try to choose another fixed edge adjacent to the starting vertex. Making such changes one should keep in mind that the same cycle can be written in two ways starting with the same starting vertex.

7 4-Color Conjecture equivalent form by Whitney.

Hassler Whitney proposed one of the first equivalent statements of the **4CT** in 1930. In a paper *A Theorem On Graphs* Whitney proved (Whitney Theorem 1) that given a planar graph composed of elementary triangles, in which there are no circuits of 1, 2, or 3 edges other than these elementary triangles, there exists a circuit which passes through every vertex of the graph exactly once, i.e. a Hamiltonian cycle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Hence, given any graph **G** on n vertices as described in the theorem above, one can construct a graph isomorphic with it (and called the *Whitney Normal [Polygonal] Form*) as follows (see Fig.7.1):

• Draw a regular polygon of n sides formed by the Hamiltonian cycle (that exists by Theorem 1);

• Draw diagonals, no two of which cross, dividing the inside of the polygon into triangles;

• Draw circular arcs, no two of which cross, dividing the outside of the polygon into circular triangles.

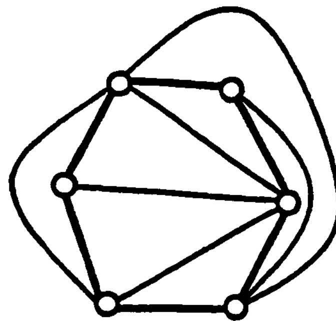


Fig.7.1. Whitney Polygonal Form.

Suppose one can color the n vertices of such a *polygonal form* by four colors so that no adjacent vertices have the same color. Consider the dual graph **D(G)** and a corresponding map **M(D(G))**. Color each region of the map by the same color as the corresponding vertex of **G**. Any two regions with a common boundary correspond to two vertices of **G** joined by an edge. and are therefore of different colors. Hence. the existence of a vertex 4-coloring of the *polygonal form* implies a 4-coloring of the corresponding dual

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

map.

Conversely, since every *polygonal form* is a dual of a map. 4-coloring of the map implies a vertex 4-coloring of a dual *polygonal form.*

It is easy to see that a 4-coloring of any map dual to a *Whitney polygonal form* implies a 4-coloring of any map on the surface of a sphere.

Hence, one obtains the following statement equivalent to **4CC**:

If the vertices of any Whitney polygonal form can be colored in four colors, then every map on the surface of a sphere can be colored in four colors, and conversely.

Note. A graph **G** on n vertices described in Whitney's Theorem 1 is a maximal planar graph. (Suppose **G** has n vertices, m edges and r regions. Then $n - m + r = 2$ (by Euler's Theorem) and $3r = 2m$ (by construction). This implies an equality $m = 3n - 6$ that corresponds to a maximal planar graph.) It is also true that any maximal planar graph is a triangulation.

8 Notes on algebra and association equations.

8.1 Vector Cross Product Algebra.

Definition 8.1: If i, j, k denote a standard unit orthogonal basis for Euclidian 3-dimensional space as a vector space over the real numbers, then the *Vector Cross Product Algebra* is defined in the following way:

1. $00 = 0$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. $0i = 0j = 0k = 0 = i0 = j0 = k0 = 0$

3. $ii = jj = kk = 0$

4. $ij = k, ji = -k, jk = i, kj = -i, ki = j, ik = -j$.

5. Distributive Law:

For any vectors u,v, and w

$u(v + w) = (uv) + (uw)$ and $(u + v)w = uw + vw$

6. Scalar Linearity:

For any vectors u,v, and w and any scalar k

$k(uv) = (ku)v = u(kv)$

Note: As a matter of convenience, vw is used here instead of the customary notation $v \times w$. Written with the customary notation, the scalar linearity property has to be written as $k(u \times v) = (ku) \times v = u \times (kv)$.

The *Vector Cross Product Algebra* is non-associative. E.g., consider an equation $x(yz) = (xy)z$. For example, $x = y = i$ and $z = j$:

$$(ii)j = 0j = 0$$

and

$$i(ij) = ik = -j.$$

However, the associated products may be equal for certain values of the variables. Set $x = i$, $y = k$. $z = i$:

$$(ik)i = -ji = k$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and

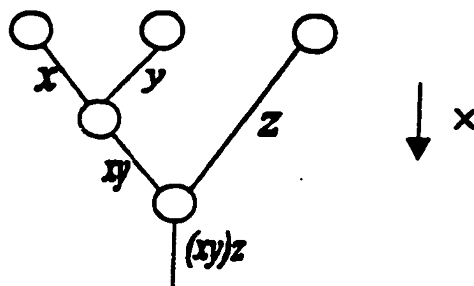
$$i(ki) = ij = k.$$

Definition 8.2 ([4]): Given any collection of variables $x_1 x_2 x_3 \ldots x_n$, let **L** (for left) and **R** (for right) denote two specific associations of the product $x_1 x_2 x_3 \ldots x_n$. A solution to the equation $\mathbf{L} = \mathbf{R}$ in the Cross Product Algebra is said to be sharp if both sides are non-zero, and the values for the variables are chosen from the elements i, j, and k.

8.2 Geometric representation of an association.

Any association can be represented by a tree similar to an expression parsing tree in a modern compiler. Set the variables x, y, z as tree leaf edges:

Fig.8.1.



T (L); $\mathbf{L} = (xy)z$

I.e., all $(n-2)$ parenthesis structures of an n-variable cross product correspond to $(n-2)$ tree representations $\mathbf{T}(\ldots)$.

It follows from the construction above that a root edge of such a tree is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

assigned the value of the whole association.

For a given **L**, one may find a solution of an equation

$$\mathbf{L} = a \text{ where } a = i, j, \text{ or } k$$

using a tree **T(L)**:

• 3-color the edges of **T(L)** in the following way: Starting from the root edge and moving up according to **Fig.8.1**, for each edge colored in one of 3 colors, representing a product vertex, color the two edges representing product factors by two different colors.

• Replace the three colors by the variables i, j, k. This produces a solution to one of the equations $\mathbf{L} = a$ or $\mathbf{L} = -a$.

• The sign of the solution is the same for both equations (see [4] for detailed proofs.)

8.3 Geometric representation of an association equation.

To represent an equation $\mathbf{L} = \mathbf{R}$ on n variables by a graph (Fig.8.2):

• Draw two trees **T(L)** and **T(R*)** where **R*** is an association **R** with the letters in reverse order;

• Collapse each pair of leaves adjacent to the edges representing the same variables into one leaf and then delete the leaves completely, leaving an edge for each variable;

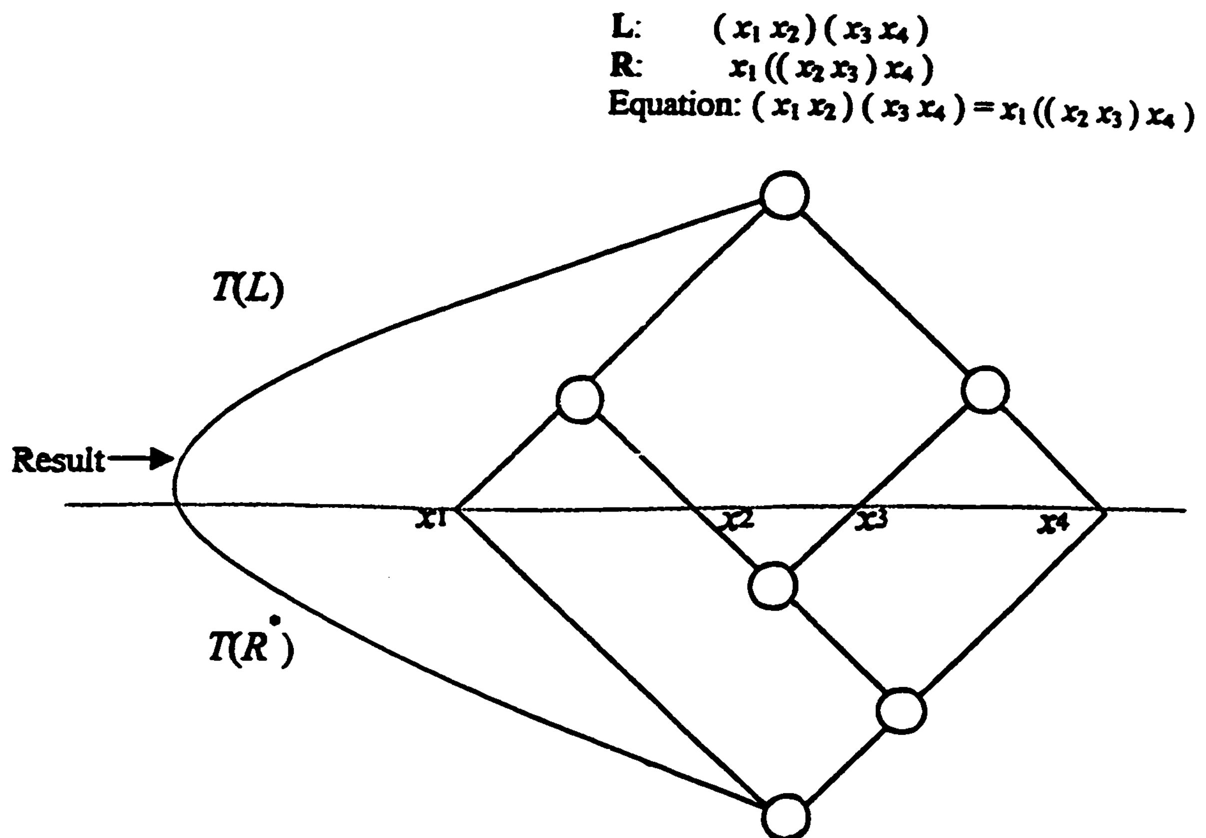
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

• Collapse the root-edges of the two trees into one edge.

One obtains a cubic planar graph denoted by $\mathbf{K} = \mathbf{T}(\mathbf{L})\#\mathbf{T}(\mathbf{R}^{\bullet})$.

It will follow that the existence of a sharp solution for an equation $\mathbf{L} = \mathbf{R}$ produces a 3-coloring of K's edges that, in its turn, will produce a 4-coloring of its regions. Conversely, a 4-coloring of the regions corresponds to a unique 3-coloring of the edges that, in turn, produces a solution for the equation above (as was shown).

Fig.8.2



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

9 Different forms of a given K-map.

On the one hand, a graphical method can be used to enumerate all possible sharp solutions of an association equation. since these are in one-to-one correspondence with the edge-colorings of the map **K(L. R)**.

On the other hand. given an equation $\mathbf{L} = \mathbf{R}$ in n variables and a corresponding map $\mathbf{K}(\mathbf{L}.\mathbf{R}) = \mathbf{T}(\mathbf{L})\#\mathbf{T}(\mathbf{R}^{*})$, one may use an edge 3-coloring of **K(L, R)** to produce solutions to certain other association equations in n variables. This will be done by considering different graphs isomorphic to **K(L. R)**.

n isomorphic transformations of a K-tree $K(L, R)$

producing $n+1$ equivalent equations.

Consider a map **K(L. R)**.

- Enumerate **K**'s $(n+1)$ regions by $r_1, \ldots, r_{n+1}$.
- Enumerate $\mathbf{K}(\mathbf{L},\mathbf{R}) \cap \mathbf{T}(\mathbf{L})$ by $k_1, \ldots, k_{n-1}$.
- Enumerate $\mathbf{K}(\mathbf{L},\mathbf{R}) \cap \mathbf{T}(\mathbf{R}^{*})$'s vertices by $k_n, \ldots, k_{2n-2}$.
- Enumerate edges tieing two trees $\mathbf{T}(\mathbf{L})$ and $\mathbf{T}(\mathbf{R}^{*})$ through $x_1, \ldots, x_n$ for variable edges and x_{n+1} for the root edge (representing the result of the equations **L** and **R**) so that a boundary of each region r_i includes a pair of edges x_i, x_{i+1} (for $i < n+1$) and x_{n+1} , x_1 (for $i = n+1$).

For each x_i there is a pair of adjacent vertices. one in $\mathbf{T}(\mathbf{L})$ and one in $\mathbf{T}(\mathbf{R}^{*})$. There are $n+1$ such pairs.

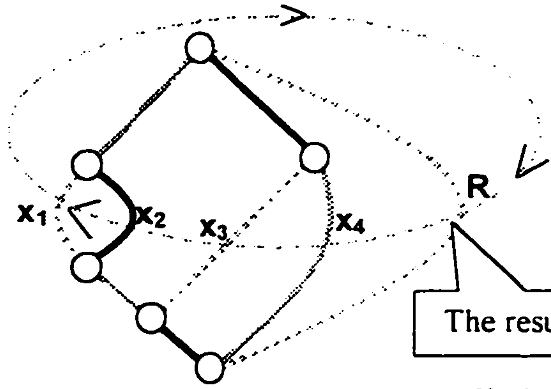
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

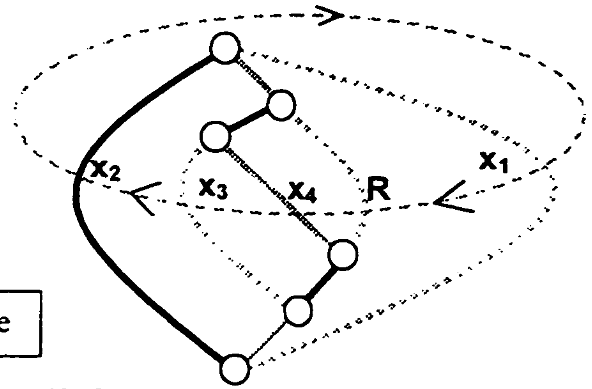
Consider an embedding of a map $\mathbf{K}(\mathbf{L},\mathbf{R})$ on a sphere. Transforming the map on a sphere so that each time another variable edge x_i becomes a root edge (or a result edge) and, correspondingly, x_i's adjacent vertices become root vertices of two trees, one will obtain n more equations in the same n variables. Each such transformation makes regions having x_i in a boundary into outside regions for $\mathbf{T}(\mathbf{L})$ and $\mathbf{T}(\mathbf{R}^*)$.

See the example below (Fig.9.1) for a case with $n = 4$ variables.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.9.1.



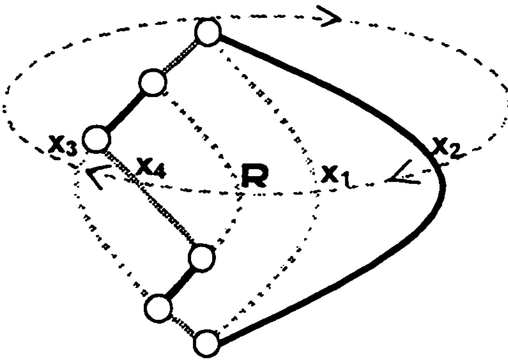


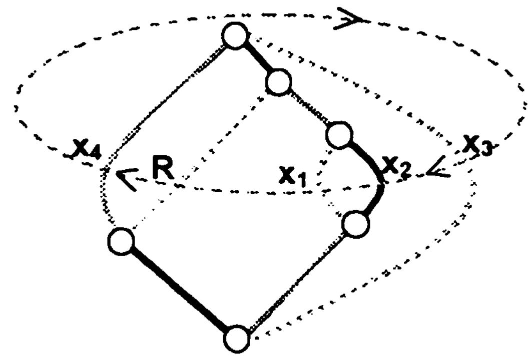
 # 1

$$(x_1 x_2)(x_3 x_4) = ((x_1 x_2) x_3) x_4 = \mathbf{R}$$

 # 2

$$x_2((x_3 x_4)\mathbf{R}) = x_2(x_3(x_4 \mathbf{R})) = x_1$$



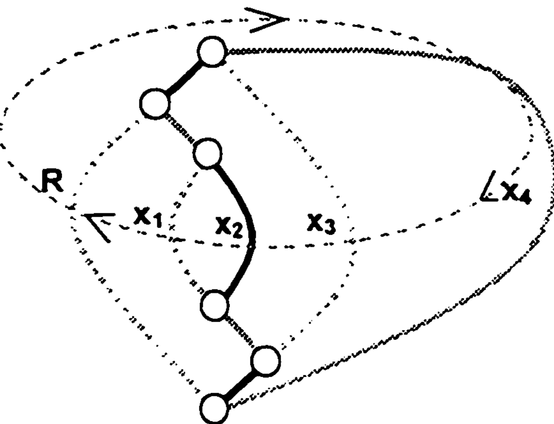


 # 3

$$((x_3 x_4)\mathbf{R}) x_1 = (x_3(x_4 \mathbf{R})) x_1 = x_2$$

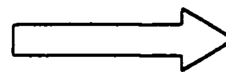
 # 4

$$x_4(\mathbf{R}(x_1 x_2)) = (x_4 \mathbf{R})(x_1 x_2) = x_3$$



 # 5

$$(\mathbf{R}(x_1 x_2)) x_3 = \mathbf{R}((x_1 x_2) x_3) = x_4$$



Back to # 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

10 Creating a K-map from a planar cubic graph.

Given a planar cubic graph. one may attempt to put it into $K - map$ form. It may be used, for example, when a graph is already 3-colored. According to the previous sections, if a graph is 3-colored and admits $K - map$ form $\mathbf{K}(\mathbf{L}, \mathbf{R})$ for some $\mathbf{L}$ and $\mathbf{R}$. substituting colors for the orthogonal basis units i, j, k immediately gives a solution to an equation $\mathbf{L} = \mathbf{R}$.

First of all, one needs to analyze a $K - map$ form $\mathbf{K}(\mathbf{L}, \mathbf{R})$ for some equations $\mathbf{L}$ and $\mathbf{R}$ in n variables. Each tree. $\mathbf{T}(\mathbf{L})$ and $\mathbf{T}(\mathbf{R}^*)$, has $(n-1)$ vertices each. the total of

$$|V| = 2(n-1) = 2n-2 \text{ vertices.}$$

Since the graph is cubic. there are

$$|E| = \frac{3|V|}{2} = \frac{3(2n-2)}{2} = 3(n-1) \text{ edges.}$$

According to the Euler formula. there are

$$|R| = |E| - |V| + 2 = 3(n-1) - (2n-2) + 2 = n+1 \text{ regions.}$$

I.e.. if there is a K-form for a graph on $|V|$ vertices. it corresponds to some equation on $n = \frac{(|V|+2)}{2}$ variables.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Further, a $K - map$ is made of two trees tied together through leaves and roots only. It means that there is always a circuit passing through all $n + 1$ regions exactly once, crossing $n + 1$ edges. not touching any vertex. and leaving exactly $n - 1$ vertices on each side. If one puts vertices on the intersections of the circuit and the edges it crosses. two original trees would be restored. It follows from Section 3 and Lemma 10.1 below that. from an algebraic point of view, given a graph and such a circui t it does not matter what vertices are picked up as a root vertices since all eligible (according to Section 3) $n + 1$ pairs of vertices would produce $n + 1$ equivalent equations.

Hence, one may ask the following two questions:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Question 1:

What planar cubic graphs on $|V|$ vertices do have a circuit that:

P1:passes through all its regions exactly once;

P2:leaves exactly $\frac{|V|}{2}$ vertices on each side?

Question 2:

Are conditions P1 and P2 necessary and sufficient for a cubic planar graph to admit a K-form?

Lemma 10.1: *Given a planar cubic graph $G(V,E)$ on $|V|$ vertices with a circuit C satisfying conditions $P1$ and $P2$ as described, there is a corresponding K-form $K(L,R)$ isomorphic to G for some equation $L = R$ on $\frac{(|V|+2)}{2}$ variables.*

Proof (constructive): Given a planar cubic graph $G(V,E)$ on $|V|$ vertices and a circuit C satisfying conditions $P1$ and $P2$, consider a map $M(G)$ - a planar embedding of G on a sphere.

C divides a sphere into 2 simple regions. Let G_L be a part of G inside one of two regions, including the edges crossing C, and G_R - its complement in G plus the same edges crossing C.

According to P2 there are $\frac{|V|}{2}$ vertices in each G_L and G_R. Besides, there are no vertex cycles inside G_L and G_R since all the regions are crossed by C. Hence, C divides the graph G into two overlapping trees G_L and G_R.

Since G is cubic and $|V|$ is even, one can set $|V| = 2n-2$ for some integer n. Hence, there are $|E| = \frac{3|V|}{2} = 3(n-1)$ edges and $|R| = \frac{|V|}{2} + 2 = n+1$ regions. Since each region is cut by C exactly once, there are $n+1$ edges

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

crossed by C.

Pick any edge (v_i, v_k) crossing C. Set vertices v_i and v_k to be root vertices of trees G_L and G_R respectively and (v_i, v_k) to be a root edge. (Then two regions having (v_i, v_k) on their boundaries become outside regions for the remaining tied trees.) If necessary, starting from the roots. rearrange vertices so that at each vertex one of three edges closer to a root (in terms of the shortest path between an edge and a root) lies geometrically closer to a root.

It is clear that on a sphere this corresponds to a geometrical rearrangement that creates a planar map isomorphic to $M(G)$ without any edge crossings.

Q.E.D.

Hence, the answer to *Question 2* is Yes.

As for *Question 1*, one needs to go back to the Whitney's theorems on graphs and maps. A theorem on maps deducible from Whitney's Theorem I for triangulations (see Section ...) states:

Theorem 10.2 (H.Whitney):

Given a map on the surface of a sphere containing at least three regions in which:

(A_1) The boundary of each region is a single closed curve without multiple point,

(B) Exactly three boundary lines meet at each vertex,

(A_2) No pair of regions taken together with any boundary lines separating them forms a multiply connected region.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(A_3) No three regions taken together with any boundary lines separating them form a multiply connected region,

we may draw a closed curve which passes through each region of the map once and only once, and touches no vertex.

Note. For a graph G as described in Whitney's Theorem II. a dual graph $D(G)$ satisfies Whitney's Theorem I, i.e. is a triangulation with no circuits of one, two, or three edges other than the elementary triangles.

By Theorem II such a G has a closed curve (or a cycle) which passes through each region of the map once and only once, and touches no vertex. If the curve divides a set of vertices in halves, it allows one to build a corresponding $K - map$. Consider a dual $D(G)$ of a given graph G that is a triangulation. Since a set of vertices of $D(G)$ is in one-to-one correspondence with a set of regions of G. a curve in G described in Theorem 10.2 corresponds to a Hamiltonian cycle in $D(G)$.

Hence, if there is a Hamiltonian cycle in $D(G)$ that divides the vertices of $D(G)$ in halves. then there is a $K - map$ dual to $D(G)$.

Given a planar triangulation $D(G)$ on the surface of a sphere together with a Hamiltonian cycle in it, put it in a *Whitney polygonal form* and consider only the inside of a polygon.

Lemma 10.2: *Given a triangulated surface bounded by a polygon with p sides and p vertices* ($p > 3$)*, there are* ($p - 2$) *triangles and* ($p - 3$) *edges not counting p boundary edges.*

Proof: Consider a triangulated surface bounded by an n-polygon $(v_1, v_2, \ldots, v_p)$,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$p > 3$. In any such triangulation:

- any pair of adjacent triangles share exactly one edge:
- there are at least two triangles each having two edges on the boundary and only one adjacent triangle;
- all other triangles have two or three adjacent triangles.

Step 1: Delete a triangle (v_1, v_2, v_3) with two edges (v_1, v_2) and (v_2, v_3) on the boundary. Hence, a vertex v_2 is deleted.

There remains a triangulated surface bounded by an $(n-1)$-polygon $(v_1, v_3, \ldots, v_n)$ on $(n-1)$ vertices that has all the properties described above.

Repeat **Step 1** k times till $(p-k) = 3$, i.e., $(p-3)$ times. The remaining polygon is also a triangle. Hence, *there were originally* $(p-2)$ *triangles.*

Let the *inside edges* denote all the edges of a triangulation except for the boundary edges. Each pair of triangles shares an inside edge.

Hence,

$$(a\ number\ of\ boundary\ edges) + 2(a\ number\ of\ inside\ edges) =$$

$$3(a\ number\ of\ triangles)$$

and

$$(a\ number\ of\ inside\ edges) = \frac{3(p-2)-p}{2} = p-3.$$

Q.E.D.

The Whitney theorems and both lemmas above imply the following two

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

statements:

Theorem 10.3: *Given a map on $|V|$ vertices on the surface of a sphere containing at least three regions in which conditions* (A_1), (A_2), *and* (A_3) *of Theorem 10.2 are satisfied, there is a K-form isomorphic to the given map corresponding to some association equation on* $\frac{|V|}{2} + 1$ *variables.*

Theorem 10.4: *If the edges of any map as described in Theorem 10.2 can be colored in three colors, then any association equation in n variables such that a corresponding K − map satisfies conditions (A2) and (A3) of Theorem 10.2 has a solution.*

Due to Lemma 10.2 condition *(P2)* can now be deleted from *Question 1* and the question be restated as following:

Question 1-A: What planar cubic graphs have a circuit that passes through all their regions exactly once?

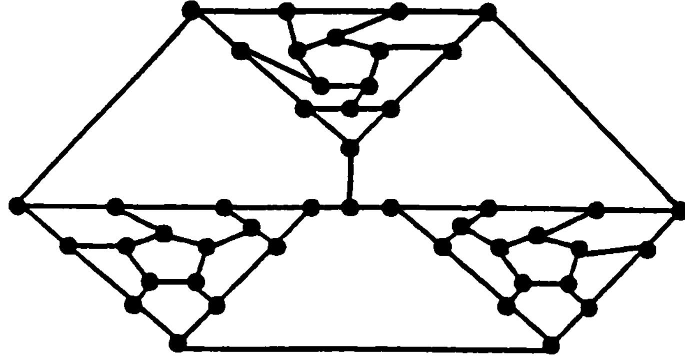
or equivalently:

Question 1-B: What planar triangulations have a Hamiltonian cycle?

Note that the Whitney's theorems only partially answer both questions 1-A and 1-B. A famous example of a graph that does not satisfy all the conditions of Theorem 10.2 is a Tutte graph (let's call it T) on 46 vertices (see Fig.10.1). In Tutte's graph there are three triples of regions such that each triple taken together with any boundary lines separating them forms a multiply connected region. Correspondingly, a dual graph $D(T)$ to Tutte's graph has three circuits of three edges othe r than elementary triangles.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.10.1 Tutte graph on 46 vertices.



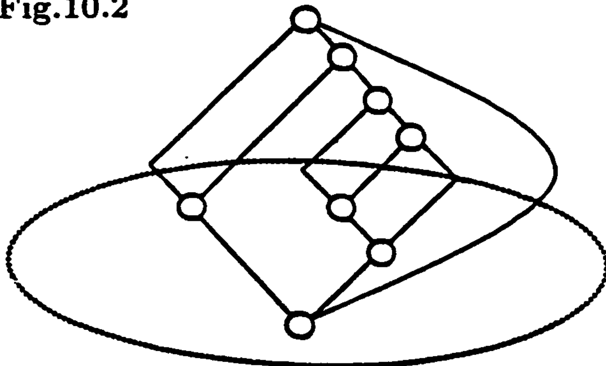
However, $D(T)$ has a Hamiltonian cycle and T has a corresponding closed curve which passes through each region of the map once and only once, and touches no vertex.

Hence,

- condition (A_3) is not a necessary condition for Whitney's Theorem II (10.2) ;
- forbidding triangles other than elementary ones is not a necessary condition for the Whitney's Theorem I.

Consider another, smaller example on 8 vertices (which is a $K-map$):

Fig.10.2



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

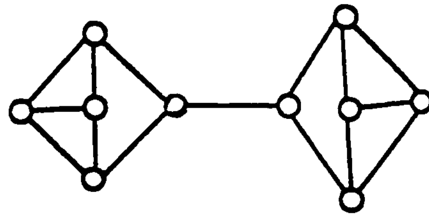
There is a closed curve passing through all regions exactly once. However, in the hypothesis of Theorem II, neither (A_2) nor (A_3) are satisfied.

Hence,

- neither (A_2) nor (A_3) is a necessary condition for Theorem II.

However, there is obviously no way to draw a closed curve passing through all regions exactly once and touching no vertices in a graph with a bridge, or an edge connecting two otherwise disconnected graph components (See Fig.10.3 below).

Fig.10.3



11 Example.

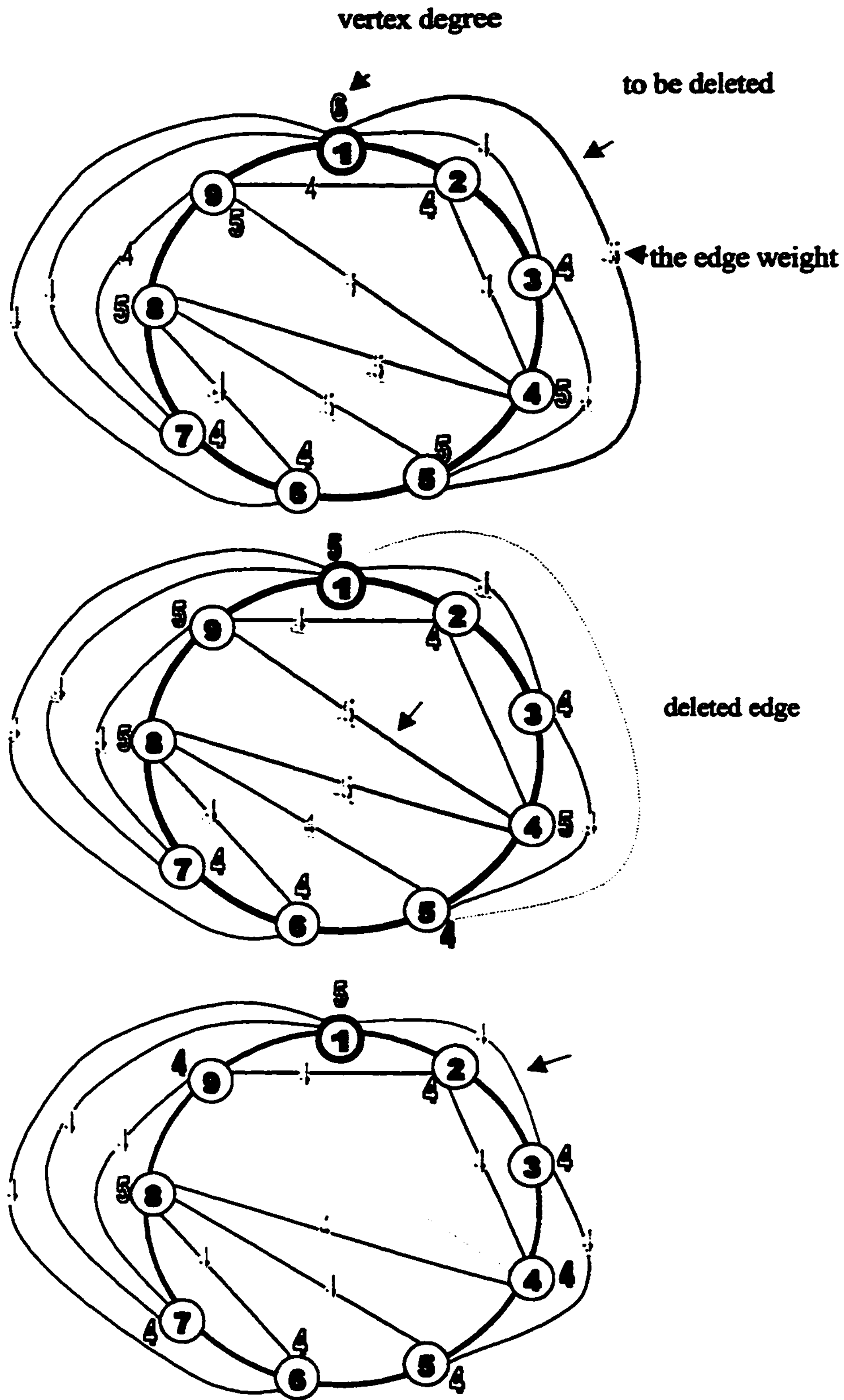
11.1 Reducing a graph to an almost cubic graph.

Consider a planar triangulation on 9 vertices (see Fig.11.1) which is the dual of the cubic planar graph on 14 vertices (see Fig.11.3). Figure 11.1 shows an example of reducing the graph to an almost cubic graph deleting the edges of weight 4 and higher (the red edges on the figures). The edge weight labels

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

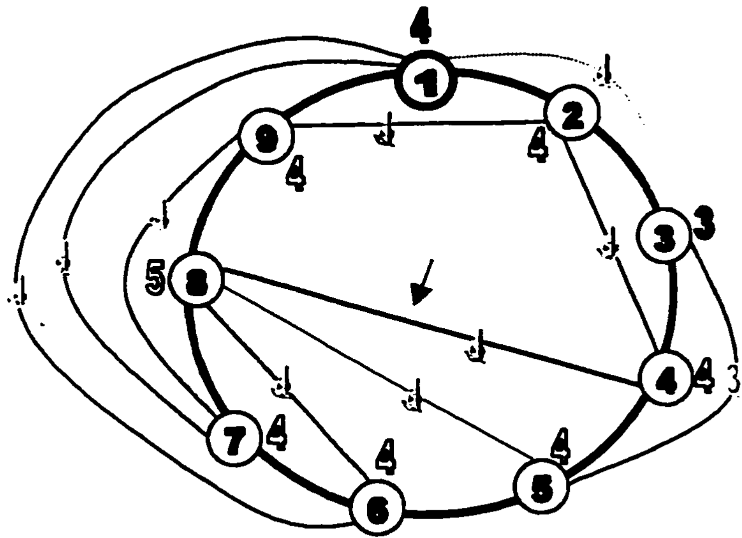
are shown right on the graph edges. The vertex degrees are shown next to vertices. The vertices are numbered 1 through 9.

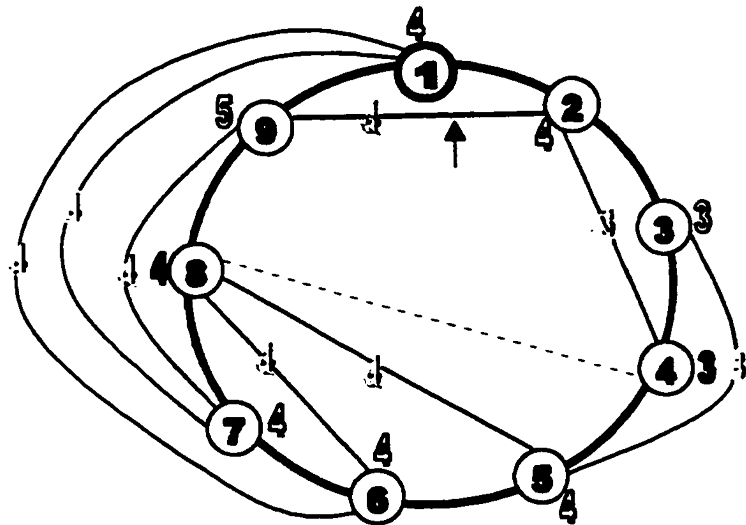
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

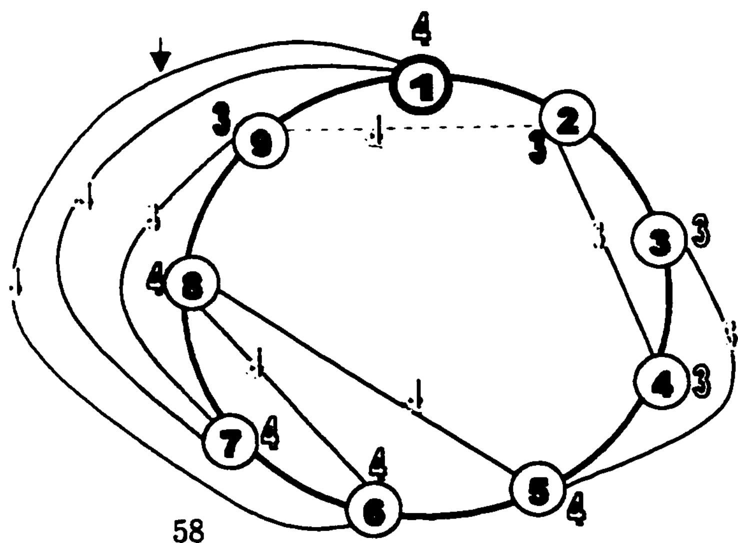


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.11.1 (cont)

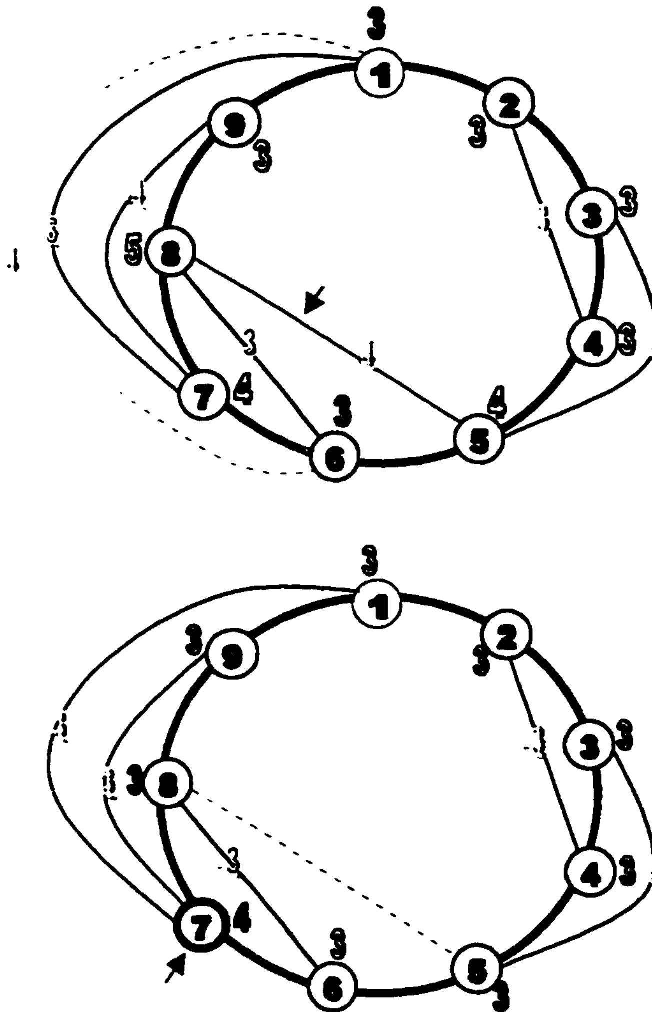






Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.11.1 (cont)



In this example, after the last edge of weight 4 is deleted, the only vertex of degree greater than three is vertex 7. All other vertices have degree 3. Hence, the graph is almost cubic with the given Hamiltonian cycle [1,2,3,4,5,6,7,8,9]. Fixing vertex 7 as a starting vertex and considering a Hamiltonian path [7,6,5,4,3,2,1,9,8] with a fixed edge (7,8) one may transform it into another unique cycle through the fixed edge using the transformation algorithm for cubic graphs or into any other cycle using the algorithm for general graphs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

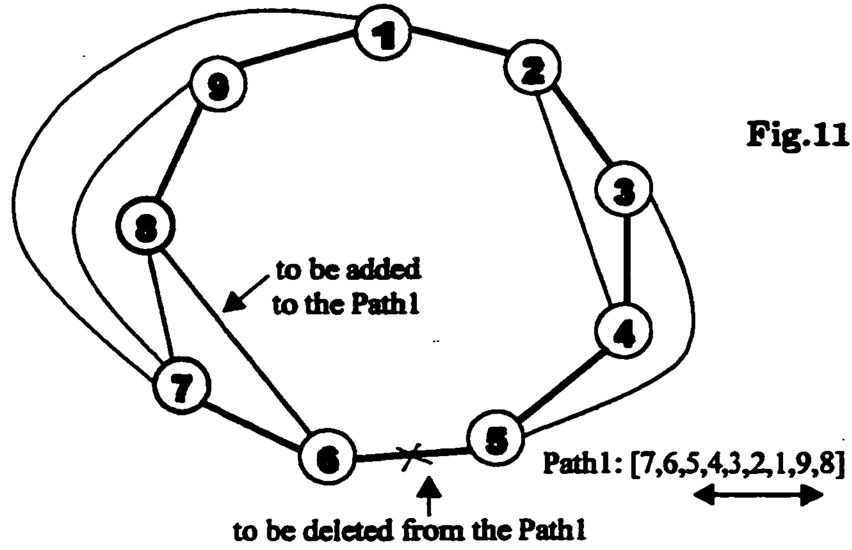
The algorithm for general graphs in this case is equivalent to the algorithm for cubic graphs since each transformation is completely determined (each degree except that of the starting vertex is equal to 3) except that it may be shorter.

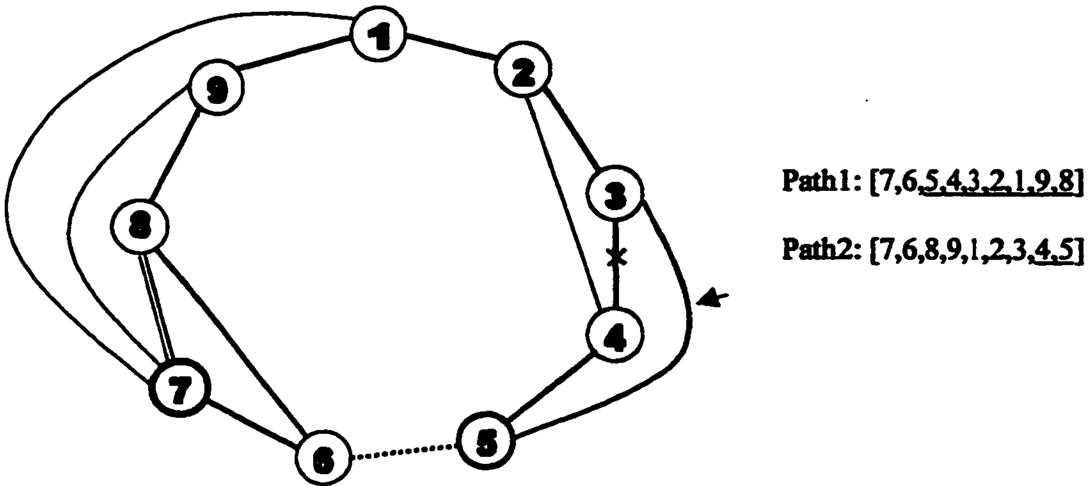
11.2 Transformation of a given Hamiltonian cycle in the reduced graph.

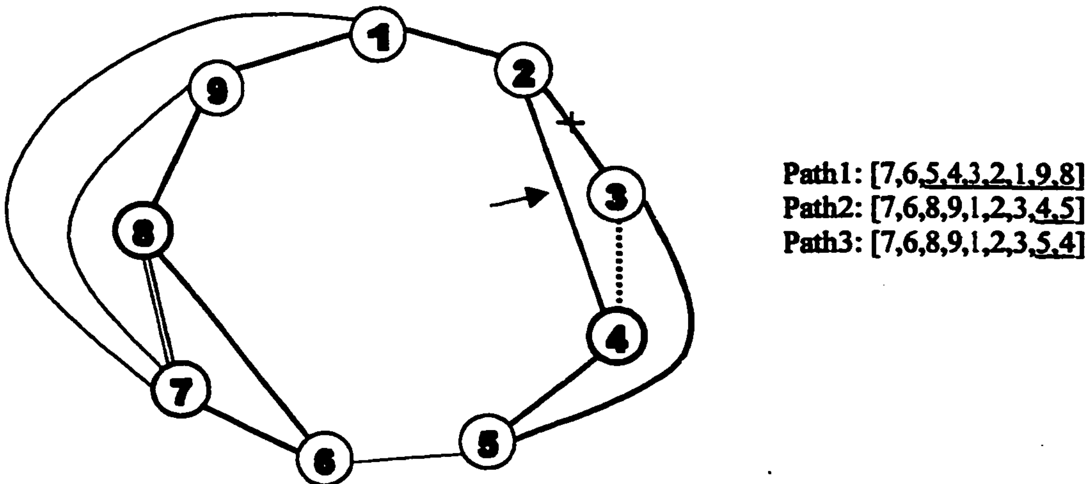
Figure 11.2 shows the transformation of a given Hamiltonian cycle [7,6,5,4,3,2,1,9,8] in the graph reduced to an almost cubic one in Fig.11.1. The edge (7,8) is fixed. Vertex 7 is used as the starting vertex for the transformations. The sequence of the Hamiltonian paths produced by the transformations is shown at each step.

Since none of the intermediate transformation paths happened to have a final vertex adjacent to a starting vertex 7, the only Hamiltonian cycle found is the final one [7,6,5,3,4,2,1,9,8].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

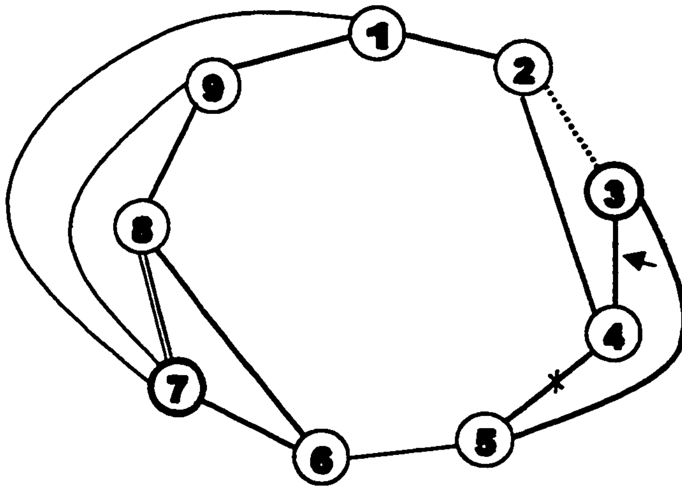




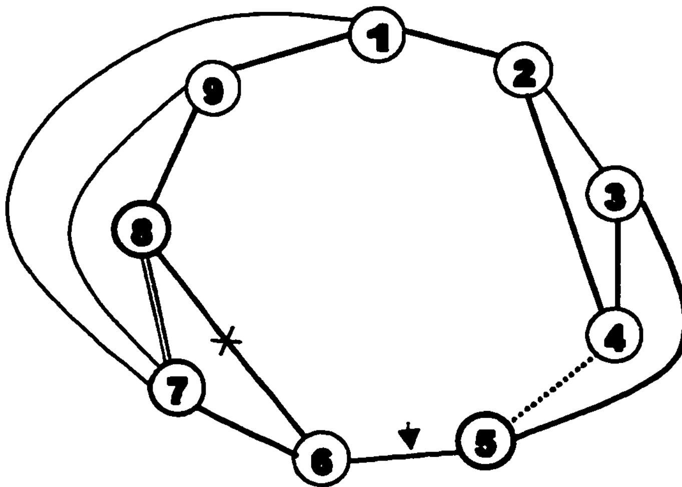


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

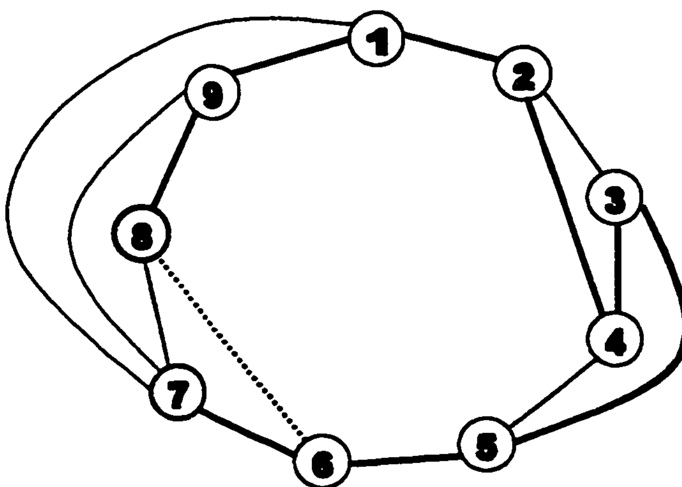
Fig.11.2 (cont.)



Path1: [7,6,5,4,3,2,1,9,8]
Path2: [7,6,8,9,1,2,3,4,5]
Path3: [7,6,8,9,1,2,3,5,4]
Path4: [7,6,8,9,1,2,4,5,3]



Path1: [7,6,5,4,3,2,1,9,8]
Path2: [7,6,8,9,1,2,3,4,5]
Path3: [7,6,8,9,1,2,3,5,4]
Path4: [7,6,8,9,1,2,4,5,3]
Path5: [7,6,8,9,1,2,4,3,5]



Path1: [7,6,5,4,3,2,1,9,8]
Path2: [7,6,8,9,1,2,3,4,5]
Path3: [7,6,8,9,1,2,3,5,4]
Path4: [7,6,8,9,1,2,4,5,3]
Path5: [7,6,8,9,1,2,4,3,5]
Path6: [7,6,5,3,4,2,1,9,8]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

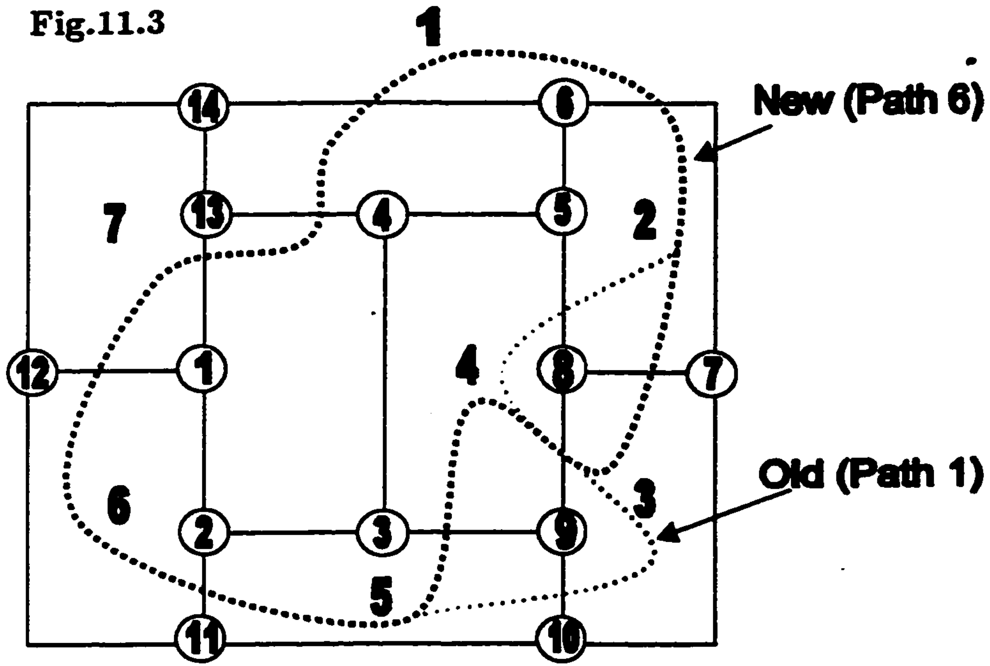
11.3 Creating K-maps and solving the corresponding association equations.

Consider the original graph on 14 vertices with no edges deleted. Both the old and the new Hamiltonian cycles generate closed curves in the graph passing through all regions exactly once and crossing no vertex (see Figure 11.3).

Using both closed curves we create two different K-maps for the graph. K-Map 1 is shown in Figure 11.4 and K-Map 2 is shown in Figure 11.5.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.11.3



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fig.11.4

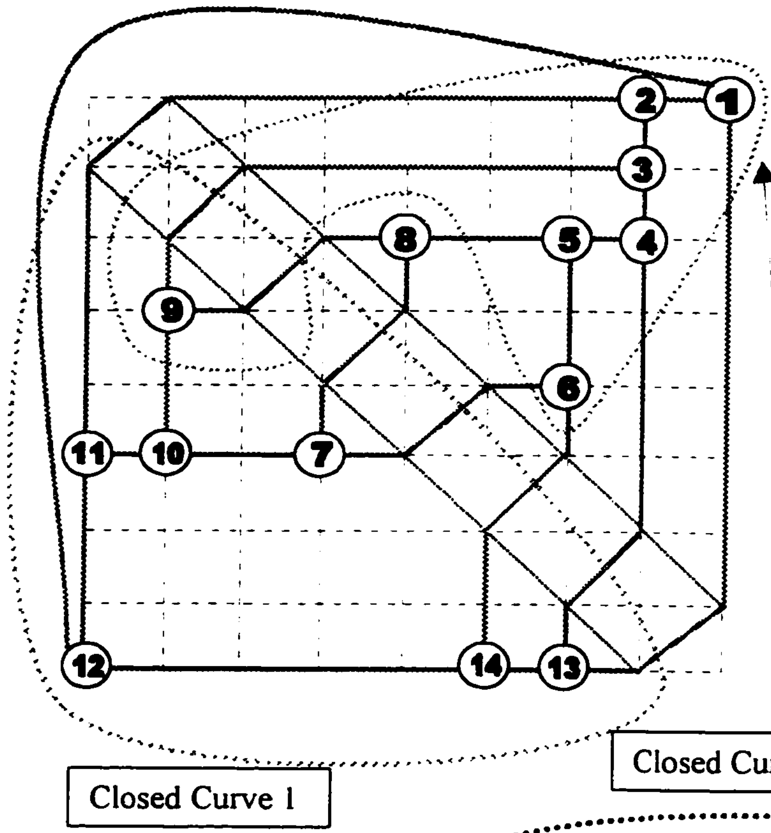
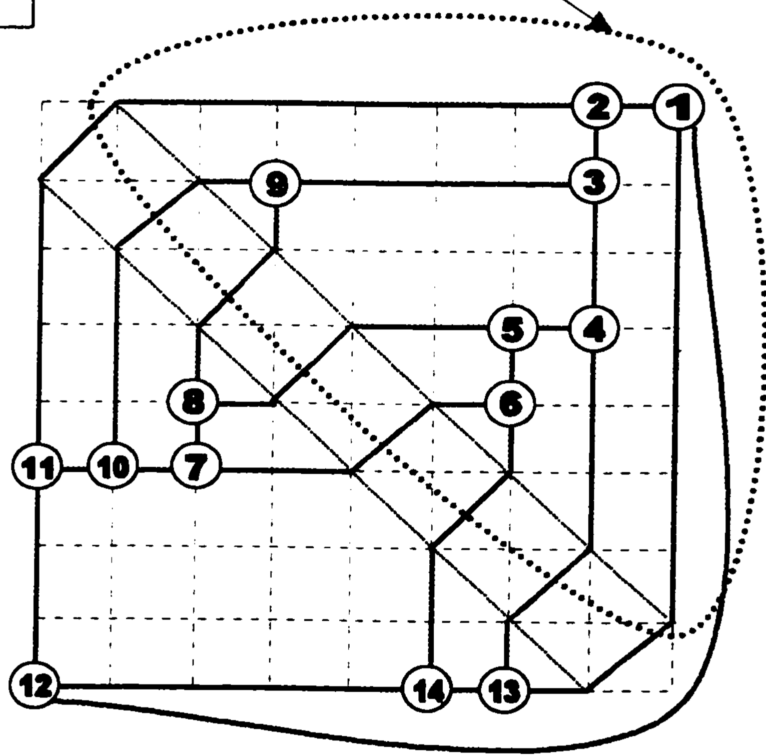


Fig.11.5



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$K-map$ 1 is equivalent to the association equation:

$$[x_1(x_2[(x_3x_4)(x_5x_6)]x_7)]x_8 = x_1[(x_2x_3)(x_4x_5)][x_6(x_7x_8)]$$

$K-map$ 2 is equivalent to the association equation:

$$[x'_1((x'_2x'_3)\{[x'_4(x'_5x'_6)]x'_7\})]x'_8 = (x'_1\{x'_2[(x'_3x'_4)x'_5]\})[x'_6(x'_7x'_8)]$$

where

$$x'_i = x_i \; for \; i = 1,3,5,6,7,8 \; and \; x'_2 = \pm x_2x_3; \;\; x'_4 = \pm x_3x_4$$

Rewritten with the first variables, the second equation becomes:

$$[x_1(((x_2x_3)x_3)\{[(x_3x_4)(x_5x_6)]x_7\})]x_8 = (x_1\{(x_2x_3)[(x_3(x_3x_4))x_5]\})[x_6(x_7x_8)]$$

Given the 3-edge coloring of the graph (hence, the 3-edge coloring of both its K-forms), one immediately obtains solutions to both equations. Different 3-edge colorings correspond to different solutions. See Figures 11.6 - 11.8 for an example.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

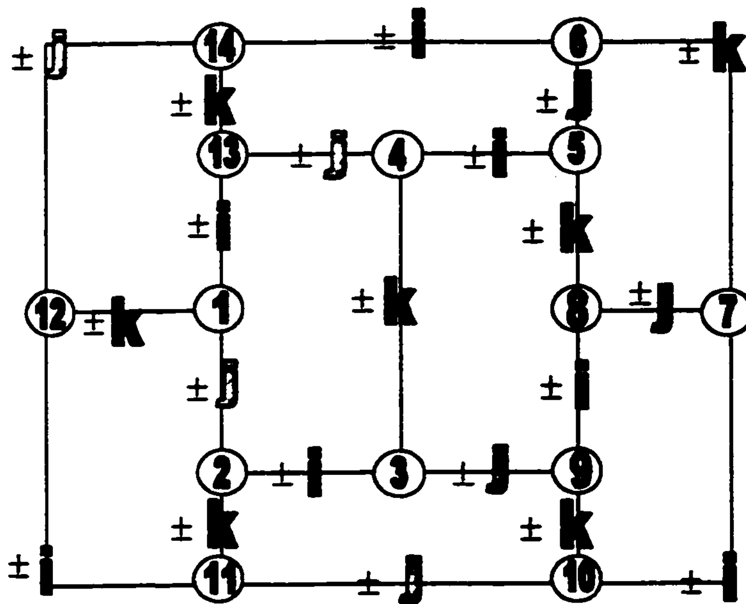


Fig.11.6. $x_2 = (3,9) = j$

$x_3 = (8,9) = i$

$x_4 = (7,8) = j$

$x_5 = (6,7) = k$

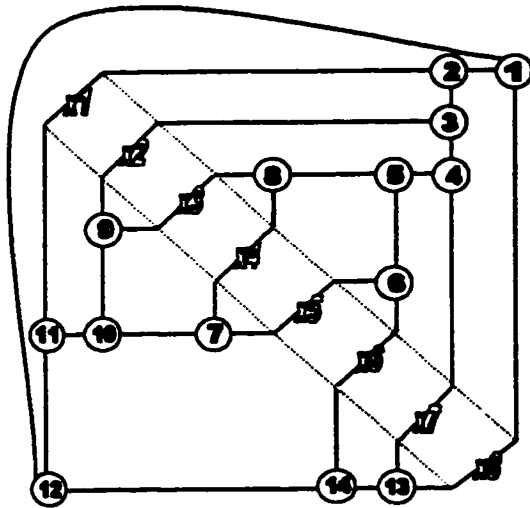
$x_6 = (6,14) = i$

$x_7 = (4,13) = j$

$x_8 = (1,13) = i$

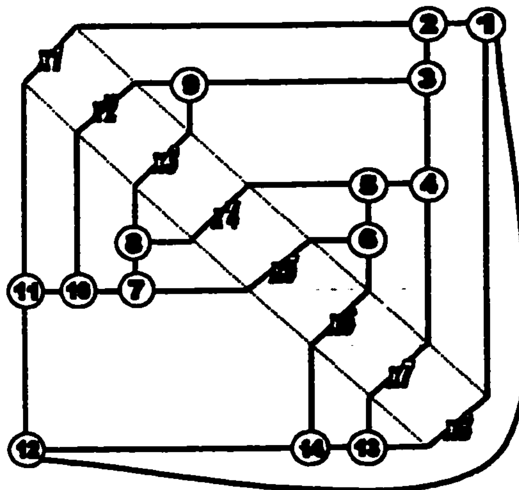
$x_2' = (9,10) = k$

$x_4' = (5,8) = k$



K-Map 1

Fig.11.7



K-Map 2

Fig.11.8

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

12 Unconventional approach to the Hamiltonian cycle problem: molecular computing.

12.1 Introduction.

There was not much said so far about the situation when a Hamiltonian cycle is required for further manipulations (such as transformations into another Hamiltonian cycle or construction of the corresponding K-map) but it is not found yet. Many algorithms exist that can find Hamiltonian cycles. In practice, however, given a graph with a few dozens of vertices, the number of steps required to complete these algorithms becomes too large for any conventional computer.

A possible solution may come from the biolabs where all the necessary information about a given graph is translated into certain DNA sequences in such a way that one may take advantage of the famous Watson-Crick complementarity. The complementary sites of single-stranded DNA molecules bind together forming longer double-stranded molecules. The original DNA molecules are prepared in such a way that two graph edge-molecules bind together only if they are incident with the same vertex in the graph. As a resul t, the Hamiltonian path molecule is formed among other "DNA junk". The result is filtered and "read" using lab euqipment. See Appendix D for the details on the first experiment performed by Leonard Adleman with a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

graph on 7 vertices. See [1] for details and other references.

The real experiment became possible in practice due to, first of all, the microscopic size of DNA molecules that can be combined together in huge amounts. Secondly, modern biolab techniques finally allow the creation of predefined DNA sequences, control their proper binding (or *ligation*) and separate and read the resulting sequences. A lot of theoretical models of DNA computation for different problems were suggested since 1994. The real lab experiments are a little behind due to the technical problems. Howe ver, the field is developing very rapidly and finding new areas of application.

A lot of material was also published describing basic lab techniques as formal operations. Using a formal language theory, DNA computation models were presented as theoretical computation models and were proved to be universal, in other words, capable of solving any problem algorithmically. (See [3], [7], and [2].)

12.2 Possible conventional computer simulation of the biolab experiments.

The real process taking place in a test tube and afterwards during all steps of filtering can be partialy simulated on the conventional computer in a few ways. Simulation, of course, does not replace the ideal DNA lab experiment due to existence of a huge number of parallel processes in a tube. Besides, computer simulation most probably will not take into consideration all the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

chemical and physical factors affecting interaction of DNA molecules. However, it may significantly help to analyze the process showing its advantages or disadvantages. Depending on the purposes of computer simulation, one can consider different approaches restricted to certain tasks.

One of the approaches would be to simulate the whole process as close to the real experiment as possible. It would require the creation of a pseudo-random analogue of a ligation reaction (see step 1 of the algorithm) when complementary molecules link to each other forming all possible paths. Such a simulation may appear helpful in the analysis of the complexity of ligation reaction, in studying the increase in complexity arising from increase of the problem input size, in estimating a probability of getting a correct answer in different conditions, etc.

Besides giving a combinatorial picture of the ligation reaction, such a simulation may in its turn show ways of widening the range of problems that may be solved in a tube.

It may be argued that such an experiment is unnecessary for the following reason. The key argument of the DNA computing advocates is that *all the paths* are likely to be formed in a test tube before filtering, implementing the idea of "exhaustive search" in practice. However, the probability of forming all the paths depends mostly on the amount of available DNA material that, in its turn, significantly increases the amount of undesirable "junk" and "*false witnesses*" in a test tube, i.e. illegal graph paths formed due to the availability of oligos close to complementary, or due to not well

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

chosen encodings for a graph vertices and edges. As a result. the probability of filtering out molecules representing the correct answers decreases with the growth of possibly unnecessary input.

Hence, taking also into consideration the difficulties with running real DNA computing experiments, it seems reasonable to attempt to simulate them on the conventional computer estimating the amounts of necessary and unnecessary input and selecting the best strategies and problems for later applications in a lab.

The main purposes of the current algorithm are to analyze:

- the number of desired Hamiltonian Paths in the output vs. other paths of length N that are likely to pass all algorithm steps except for the last one (see "Magnetic Beads" step in Appendix D);
- complexity and reliability of simulation.

Future goals:

- the dependence between the number of available edge oligos in the bio experiment and likely formation of the desired paths;

Different simulation approaches.

Consider a graph G on N vertices. The most straightforward method would be to actually create N distinct strings of length L, one for each edge, and "put" them together simulating all possible random interactions between molecules and linking them together in paths according to their complementarity. However, at this point it seems to be too complicated and unnecessary. Using the very idea of randomness one may consider an

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithm more suitable for a sequential computer. i.e. creating all such random linked paths sequentially. The simulation algorithms **Simula I** and **Simula II** considered below represent the latter approach.

The basic algorithm. Analysis.

Since all ligations (concatenations) of molecules in a test tube happen almost randomly ("almost" refers to possible negligible bio-chemical bounds other than Watson-Crick complementarity) in favorable bio-chemical conditions, each ligation depends mostly on the number of available complementary oligos that gradually decreases. Since the key idea of the lab experiment is a DNA exhaustive search, one has to provide sufficient amount of raw DNA material so different molecules corresponding to all graph paths of length N are likely to form. Taking this into consideration, it seems reasonable to attempt to simulate, first of all, the ideal process (Simulation I) with an unlimited amount of available complementary oligos. In terms of a conventional computer it would mean considering sequential random path generation where the only restrictions are the adjacency of vertices and a fixed Start vertex. To make this simulation more meaningful the program may count the number of times each edge oligo is used before a Hamiltonian Path is found. In further simulations certain restrictions will be added, such as a fixed Finish vertex and a finite number of available edge oligos. Such experiments are believed to help in estimating the expected number of "bad" paths formed along the Hamiltonian Path.

Description.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The program is based on a randomized algorithm producing random paths of length N in a Cubic graph G on N vertices (i.e., Hamiltonian Paths) starting from the fixed vertex Start. In Simulation II the final vertex Finish will also be fixed. At each step the next vertex is picked randomly from at most three available adjacent vertices using a random number generator. In Simulation I, for an undirected graph the number of available outgoing edges is two only for the vertices adjacent to the Start vertex, since a random walk is not allowed to return to the Start vertex. The only input is an adjacency matrix for a graph of at most (Maximum Vertex Degree) $\times N$ ($3N$ in the case of undirected Cubic graphs) numbers that is entered from a text file. The algorithm is split into a user set number of *Runs* that depends on particular statistical goals. Inside each *Run* the random paths of length N are being generated until a Hamiltonian Path is found. The output for a *Run* is the number of found paths and the number of times each edge ("oligo") was used. Separate output is formed collecting all formed paths for future analysis.

The programs have to be slightly changed to be applied to any graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Results of simulating L.Adleman's experiment:

Per 100 Runs

Length	Paths	% of Total
2	4.567	32.33%
3	-	0.00%
4	-	0.00%
5	385	2.73%
6	178	1.26%
7	8,995	63.68%
All:	14.125	100.00%

Hamiltonian Paths: 0.71%

Per 400 Runs

Length	Paths	% of Total
2	20.334	32.98%
3	-	0.00%
4	-	0.00%
5	1,755	2.85%
6	833	1.35%
7	38.734	62.82%
All:	61.656	100.00%

Hamiltonian Paths: 0.65%

Per 1000 Runs

Length	Paths	% of Total
2	488	31.38%
3	0	0.00%
4	0	0.00%
5	48	3.09%
6	28	1.80%
7	991	63.73%
All:	1555	100.00%

Hamiltonian Paths: 0.10%

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Other results.

The programs were tested also on various cubic graphs. The largest graph considered had 24 vertices. The statistical results received will be used in the future analysis of DNA computing algorithms. The author also plans to use the known results on the number of Hamiltonian cycles in various graphs to make the searching algorithms more efficient.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Program Code for the Randomized Transformation Algorithm.

Language: C++
Compiler: MicrosoftVisual C++ 5.0 under WindowsNT
Processor: Intel Pentium

Note:

The program uses pseudo-random number generator **rand()** supplied with MicrosoftVisual C++ 5.0 compiler function library. Generator is seeded once before multiple program runs start with current time.

```
# include <string.>
# include <stdio.h>
# include <stdlib.h>
# include <iomanip.h>
# include <fstream.h>
# include <math.h>
# include <time.h> // Provides Pseudo-Random function seeding.

// Change manually "Graph_Size" and "Max_Degree" for current graph:
const int Graph_Size = [Type in number of vertices in a graph] ;
const int Max_Degree = [Type in Max Vertex Degree in a graph] ;

// To hold graph's adjacency matrix:
int Adjacent_Vertex[Graph_Size][Max_Degree];
int Vertex_Degree[Graph_Size];

// To hold current and transformed paths:
int Path_Vertex[Graph_Size];
int Tr_Path_Vertex[Graph_Size];

int i,j,t,step=1,Run;
char in_file_name[50];
char out_file_name[50];
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
ifstream in_file;
ofstream out_file1;
/***MAIN****************************************************************/
void main() {
void Random_Transform();

// Change a path and a name of the INPUT FILE manually:
in_file.open("Type a path and a file name here");
// Reading graph adjacency info from the file:
for ( i=0; i<Graph_Size; i++ )
 {
 for ( j=0; j<Max_Degree; j++ )  in_file >> Adjacent_Vertex[i][j];
 in_file >> Vertex_Degree[i];
 }

 in_file.close ();

// Change a path and a name of the OUTPUT FILES manually:
// (out_file1 to trace all transformations;
//  out_file2 to keep only the new cycles)

out_file1.open("[Type a path and a output file 1 name here]");
out_file2.open("[Type a path and a output file 2 name here]");

// Seeding pseudo-random function with current time:
srand( (unsigned)time( NULL ) );
// Running Random_Transform procedure certain number of times
// (for statistical purposes):

for (Run=1; Run<50; Run++)
{  out_file1 << endl << endl << "Run # " << Run << endl;
Random_Transform();
}
out_file1.close();
}
//*********************************************************************
// Checking whether a new path is a Hamiltonian Cycle by
// checking whether the last vertex is adjacent to Start:

int Is_It_A_Cycle()
{
int m=0, Last_v;
Last_v = Tr_Path_Vertex[Graph_Size-1];
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
while ( (Last_v != Adjacent_Vertex[0][m]) && (m<Max_Degree) ) m++;
if (m<Max_Degree) return 1;
        else return 0;
}
//*****************************************************************
// Checking whether a new cycle is distinct from the original cycle:

int Is_It_Different_Cycle()
{
int m=0;
while ( (Tr_Path_Vertex[m] == m) && (m<Graph_Size) ) m++;
if (m<Graph_Size) return 1;
        else return 0;
}
//*****************************************************************
// Pseudo-Random transformations of a given Hamiltonian Cycle:

void Random_Transform(){
step = 1;
int Guess, a, Previous_a, Last_Vertex;
out_file1 << "HC1 =     ";
for (i=0; i<Graph_Size; i++)
{
Tr_Path_Vertex[i]=i;
out_file1 << setw(3) << Tr_Path_Vertex[i];
}
out_file1 << endl;
Previous_a = 0;

do {
//**************************************
  for (i=0; i<Graph_Size; i++)
  {
   Path_Vertex[i] = Tr_Path_Vertex[i];
  }
  Last_Vertex = Path_Vertex[Graph_Size-1];

// Step 4 ****************************
//
// Guess = (pseudo-random number) mod (Vertex Degree)
// "randomly" chooses the next adjacent vertex 'a'
// used in the next transformation.
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Guess = (rand())%(Vertex_Degree[Last_Vertex]);
a = Adjacent_Vertex[Last_Vertex][Guess];

// Checking whether 'a':
// - is a vertex adjacent to the Last_Vertex in the current path.
// - is a starting vertex
// or
// - was already used in the previous step.
// * If any answer is "Yes" then the next adjacent vertex is picked:

while (( a == Path_Vertex[Graph_Size-2] ) || (a == 0) || (a == Previous_a)
{       Guess = (Guess+1)%(Vertex_Degree[Last_Vertex]);
        a = Adjacent_Vertex[Last_Vertex][Guess];
}
Previous_a = a;

// Step 5 ****************************
t=0;
while ( Path_Vertex[t] != a ) t++;
for (i=1; (t+i)<(Graph_Size-i); i++)
{
  Tr_Path_Vertex[Graph_Size-i] = Path_Vertex[t+i];
  Tr_Path_Vertex[t+i] = Path_Vertex[Graph_Size-i];
}
out_file1 << "Step " << setw(3) << step << " - ";
for (i=0; i<Graph_Size; i++)
{
  out_file1 << setw(3) << Tr_Path_Vertex[i];
}
out_file1 << endl;

//*************************************

step++; }
while ((Is_It_A_Cycle() == 0) || (Is_It_Different_Cycle() == 0));

// Prints out a new cycle into out_file2:

for (i=0; i<Graph_Size; i++)
 { out_file2 << setw(3) << Tr_Path_Vertex[i]; } out_file2 << endl; }

//****************************** THE END ********************************
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

B Detailed program output 1.

Note. Each run starts with the same Hamiltonian cycle and terminates when the new cycle is found.

Run # 1
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 2
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 3
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 4
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 5
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 6
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 7
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 8
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 9
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Run # 10
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 11
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 12
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 13
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 14
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 15
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 16
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 17
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 18
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 19
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 2 - 0 1 8 7 6 5 2 3 4
Run # 20
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 21
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 22
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 23
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 24
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 25
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 26
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 27
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 28
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 29
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Run # 30
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 31
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 32
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 33
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 34
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 35
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 36
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 37
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 38
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 39
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 40
$HC_1 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 41
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5
Run # 42
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 43
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 44
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 45
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 46
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 2 3 4
Run # 47
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 2 3 4 5 6 8 7
Run # 48
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 5 4 2 3
Run # 49
HC_1 = 0 1 2 3 4 5 6 7 8
Step 1 - 0 1 8 7 6 5 4 3 2
Step 2 - 0 1 8 7 6 2 3 4 5

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

C Detailed program output 2.

Run # 1

HC1=

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 -

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19

Step 2 -

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22

Run # 2

HC1=

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 -

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23

Step 2 -

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 23 24 22 21

Run # 3

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 23 24 18 17

Run # 4

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22

Run # 5

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19

Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24

Run # 6

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22

Run # 7

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17

Run # 8

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 23 22 21
Run # 9
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17
Run # 10
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17
Run # 11
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24
Run # 12
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 13
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 14
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 15
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 22 21 20 19
Step 4 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22
Run # 16
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22
Run # 17
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 18
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 23 24 18 17
Run # 19
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22
Run # 20
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 21
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17
Run # 22
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 22 21 20 19
Step 4 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22
Run # 23
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 24
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 23 22 21
Run # 25
HCl= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 26
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 22 21 20 19
Step 4 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22
Run # 27
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22
Run # 28
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 22 21 20 19
Step 4 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22
Run # 29
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17
Run # 30
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24
Run # 31
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 24 23 18 17
Run # 32
H 1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 23 24 22 21 20
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 23 20 21 22 24
Run # 33

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24
Run # 34
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 35
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 23 24 22 21 20
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 23 24 22 20 21
Run # 36
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 37
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 38
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 39
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 24 23 18 17
Run # 40
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 22 21 20 19
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 24 23 19 20 21 22
Run # 41
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 17

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Run # 42
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24
Run # 43
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 24 23 18 17
Run # 44
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 18 19 20 21 22
Run # 45
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24
Run # 46
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 22 21 20 19
Step 4 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 23 19 20 21 22
Run # 47
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 23 22 21 20 19 18
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 21 22 23
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 24 18 19 20 23 22 21
Run # 48
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 24 23 18 17
Run # 49
HC1= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Step 1 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 23

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

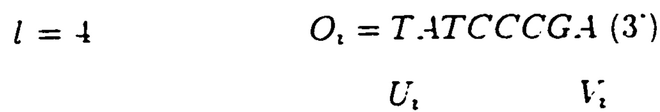
Step 2 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 24 22 21 20 19
Step 3 - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 23 19 20 21 22 24

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

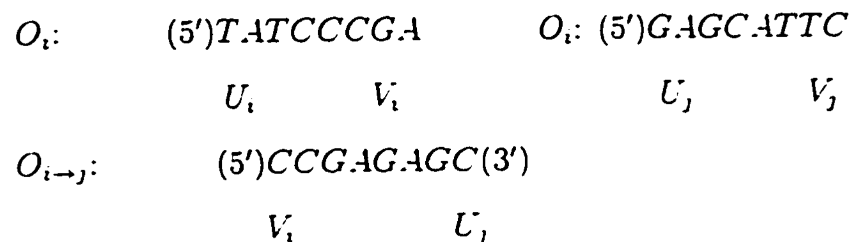
D Mr. L. Adleman's Algorithm ("Mr. Adleman's Seven Days").

Material.

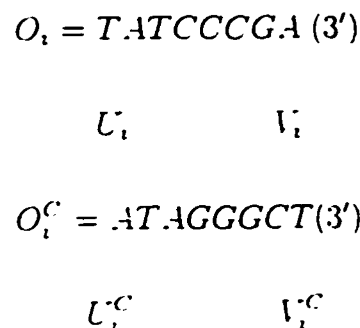
1. For each vertex i choose a random $2l$-base (20-base in experiment) strand of DNA. O_i (*oligonucleotide* or *oligo*), e.g.:



2. For each edge (i, j) in the graph create a $2l$-base DNA strand that consists of the last l bases of $O_i(V_i)$, followed by the first l bases of $O_j(U_j)$, *e.g.*:



3. For each vertex i in the graph. O_i^C is the Watson-Crick complement of O_i, *e.g.*:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

O_i^C serves as a splint to bind $O_{k\to i}$ and $O_{i\to j}$:

$$O_i^C$$

$$ATAGGGCT$$

$$TTAATATCCCGAGAGC$$

$$O_{k\to i} \qquad\qquad O_{i\to j}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Algorithm.

Step 1. Generate a large number of paths through the graph.

"*Ligation reaction*" (or *concatenation*): Approximately 3 × 1013 copies of each oligo for edges $O_{i \to j}$ and vertex complements O_i^C are combined in a test tube. In certain conditions complimentary oligos link to each other, creating DNA double-helix.

Result: DNA molecules encoding random paths through the graph 1.

Step 2. Keep only those paths that start at $Vertexstart$ and finish at $Vertexfinish$.

"PCR" (*Polymerase Chain Reaction*): Only DNA strands (→ paths) with certain starting and finishing l-oligos (→ vertices) are reproduced repeatedly.

Result: The liquid in the tube is dominated by double-stranded oligos encoding paths through the graph that start and finish at the given vertices.

Step 3. Keep only those paths entering exactly N vertices, i.e. paths of length N.

"*Gel Electrophoresis*": DNA molecules (all charged negatively) of different length exposed to electric field in an agarose gel migrate from the negative end to the positive and cover different distances in certain period of time, leaving a spectrum of different molecular weighs (→ lengths). The band of molecules of length N was later excised and soaked to extract DNA.

Result: Molecules of length N are separated.

Step 4. Keep only those paths that pass through each vertex at least once.

"*Magnetic beads*": Check if a single-stranded DNA molecule possess a certain

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

oligo (which correspond to a certain vertex).

First, single-stranded DNA was generated from Step 3 double-stranded product.

Second, the single-stranded DNA was incubated with O_1^C conjugated to magnetic bead.

Result: Mostly only those single-stranded DNA molecules that contained the sequence O_1 annealed to the bound O_1^C and were retained.

The second part was repeated successively with all other $N-1$ O_1^C beads.

Result: Mostly only Hamiltonian path strands will pass through all N different magnetic beads.

Step 5. Read the output. Check it using conventional techniques.

In the experiment the product was amplified by PCR and run on a gel.

Mr. Adleman's Algorithm adopted for a Hamiltonian cycle instead of a Hamiltonian path.

Hamiltonian cycle is a special case of a Hamiltonian path when there is an edge connecting *Finish* vertex to a *Start* vertex. Since the algorithm starts with fixing certain *Start* and *Finish* vertices for a path to be found, it can be as well applied for finding a Hamiltonian cycle.

Hence:

- to find all cycles in a graph on N vertices, one would need either to run one experiment without step 2, "PCR" (since the *Start* and *Finish* vertices for a path would not be important) or to separate a solution at certain step (possibly from the very beginning) to separate Hamiltonian paths according

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to their *Start* and *Finish* vertices.

- to find a Hamiltonian cycle through given fixed edge the algorithm may stay the same as for finding Hamiltonian Path with given *Start* and *Finish* vertices.

Note: Since each Hamiltonian cycle of length N can be actually transformed into N distinct Hamiltonian Paths of length N removing one of N edges at a time, the problem of finding any Hamiltonian Cycle can be formally replaced by a problem of finding at least one Hamiltonian Path such that a *Start* vertex is adjacent to a *Finish* vertex.

A problem of finding a Hamiltonian cycle through a given fixed edge $(Start, Finish)$ can be transformed in the following way. Let $\{Finish, a, b\}$ be the vertices adjacent to the *Start* vertex. Then original problem can be transformed into a problem of finding at least one of possibly three paths described below:

i. one with a fixed *Start* and *Finish* vertices;

ii. one with a fixed *Start* vertex and a fixed ending vertex a that has *Finish* as a second vertex;

iii. one with a fixed *Start* vertex and a fixed ending vertex b that has *Finish* as a second vertex.

Adding then an edge $(Start, a)$ to a path (ii) and an edge $(Start, b)$ to a path (iii) one obtains the desired cycle.

Of course, paths (ii) and (iii) may not exist at all but looking for them along with the main objective (i) may significantly increase a probability of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

having correct answer in the final filter. Using the properties of particular graphs considered for input, one may find other possibilities of extending the type of useful output.

E Example of a simulation program code.

```
/* Simulation II
 * Randomly finds HP's starting at the specified vertex Start.
 * FEATURES:
 * i.    A walk is not allowed to return to Start vertex.
 * ii.  The program keeps track of the maximum number of times
 *        any edge was used in any random walk.
 * For more details see Simula_I.doc.
 * Created:     July 4, 1998
 * Last Modified:       July 7, 1998
 */
# include <string.h>
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <iomanip.h> // Include setw(x) output formatting function.
# include <fstream.h> // Header file necessary for file manipulation.
# include <math.h>

const int Graph_Size = 24;
int Path_Vertex[Graph_Size];  // To hold generated paths.
int Adjacent_Vertex[Graph_Size][3]; // To hold graph's adjacency matrix.
int i,j;

// To store a number of times each edge was used in a walk.
long Used_Copies[Graph_Size][3];
long Max_Used_Copies = 0;
char in_file_name[50];          // Variable to hold the external
char out_file_name[50];         // (WIN 95) names of the In/Output files.
ifstream in_file;               // In/Output file stream variables – internal (C++) names
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
ofstream out_file1, out_file2;
//______________________________________________________________
void main()
{
void Random_Paths();
in_file.open("C:\\MSDev\\Projects\\Simula_1\\S4.txt");

// Reads graphs adjacency info from the file:
   for ( i=0; i<Graph_Size; i++ )
   {
     in_file >> Adjacent_Vertex[i][0] >> Adjacent_Vertex[i][1] >> Adjacent_Vertex[i][2];
   }
   in_file.close ();
   out_file1.open("C:\\MSDev\\Projects\\Sim_II_b\\S4_out.txt");
  out_file2.open("C:\\MSDev\\Projects\\Sim_II_b\\S4_HPs.txt");
  Random_Paths();
  out_file1.close();
  out_file2.close();
}
// Function checking whether a path of length N is Hamiltonian:
int Is_It_Hamiltonian()
{
  i = 1;
  j = 0;
  while ((Path_Vertex[i] != Path_Vertex[j]) && ((i+1)<Graph_Size))
  {
    i++;
    j = i;
    do {j--;} while ((Path_Vertex[i] != Path_Vertex[j]) && (j>0));
  }
  if (Path_Vertex[i] != Path_Vertex[j]) retum 1;
    else retum 0;
};
/* Random_Paths() creates 'Graph_Size'-long random paths through
 * the graph starting with vertex Start specified by the user
 * until a paths happens to be Hamiltonian, i.e.
 * it passes through each vertex exactly once:
 */
void Random_Paths()
{
int Step, Start, Finish, Next, Run;
long Path_Number;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
/* Seed the random-number generator with current time so that
 * the numbers will be different every time it runs.
 */
srand( (unsigned)time( NULL ) );
Start = 0;
out_file1 << "HP # in a Run| Run # | Max Used Copies Per Runs" << endl << endl;
// Loop runs till the program is terminated:
for (Run = 1;; Run++)
{
    cout << "a " << Run << endl;
    Path_Number = 0;
    // Sets the number of used edge copies for each edge equal to 0:
    for (i=0; i<Graph_Size; i++)
    {
        for (j=0; j<3; j++)
        Used_Copies[i][j] = 0;
    }
    // Creating paths:
    do
    {
        Path_Vertex[0] = Start;
        for (Step=1; Step < Graph_Size; Step++ )
        {
        do { Next = (rand())%3; }
        while (Adjacent_Vertex[Path_Vertex[Step-1]][Next]==0);
        Used_Copies[Path_Vertex[Step-1]][Next]++;
        Max_Used_Copies = __max ( Max_Used_Copies,
        Used_Copies[Path_Vertex[Step-1]][Next]);
        Path_Vertex[Step] = Adjacent_Vertex[Path_Vertex[Step-1]][Next];
        }
        Path_Number++;
    } while (Is_It_Hamiltonian() == 0);
    out_file1 << setw(10) << Path_Number << setw(10) << Run << setw(10) <<
Max_Used_Copies << endl;
    out_file2 << "Run # "<< setw(5) << Run << " | ";
    for (i=0; i<Graph_Size; i++)
        out_file2 << setw(3) << Path_Vertex[i];
    out_file2 << endl;
}
}
//********** T H E    E N D ***************************************************************
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

[1] L. M. ADLEMAN. On constructing a molecular computer. *Draft* (Jan. 1995).

[2] L. KARI. Dna computers: tomorrow's reality. *Tutorial in the Bulletin of EATCS* #59 (1996), 256–266.

[3] ———. Dna computing - the arrival of biological mathematics. *Mathematical Intelligencer* **19** #2 (1997), 9–22.

[4] L. H. KAUFFMAN. Map coloring and the vector cross product. *Journal of Combinatorial Theory, Series B* **48** (1990), 145–154.

[5] C. H. PAPADIMITRIOU. The complexity of the parity argument. *Journal of Computer And System Sciences* **48** (1994), 498–532.

[6] ———. *Computational Complexity*. U.S.A.: Addison-Wesley Publishing Company, Inc., 1994.

[7] G. PAUN R. FREUND. L. KARI. Dna computing: the existence of universal computers. *Theory of Computer Science (to appear)* .

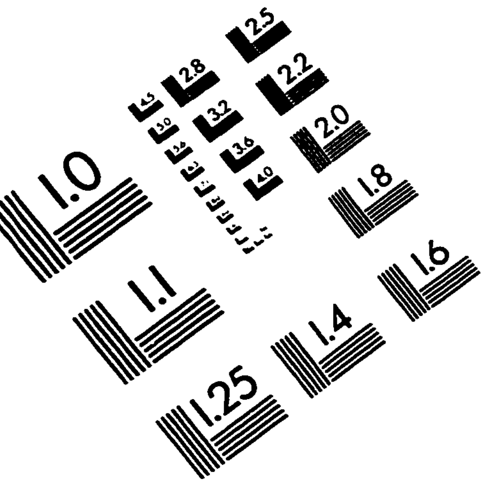
[8] T. L. SAATY & P. C. KAINEN. *The Four-Color Problem: Assaults and Conquest*. New York: Dover Publications. Inc.. 1986.

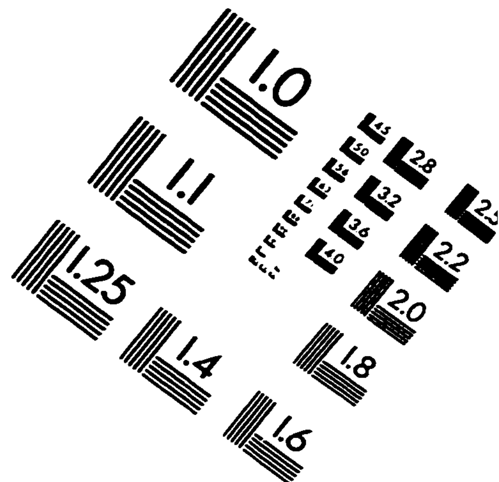
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

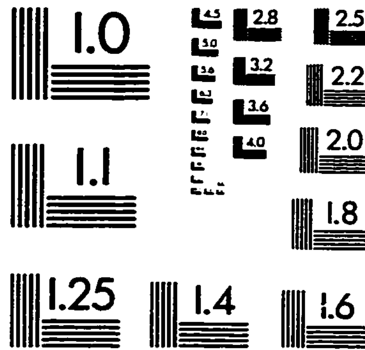
[9] R. THOMAS. An update on the four-color theorem. *Notices of the AMS* **45** #7 (1998). 848–859.

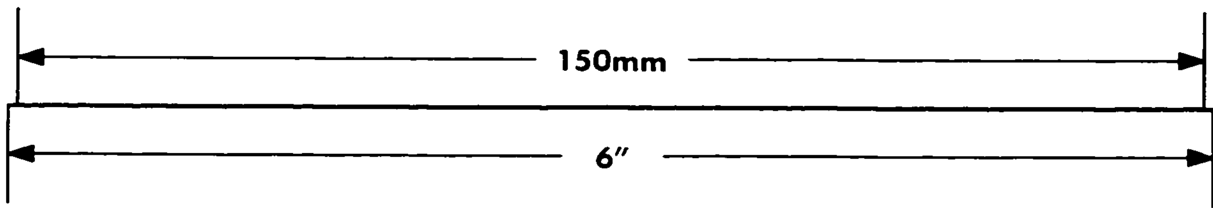
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

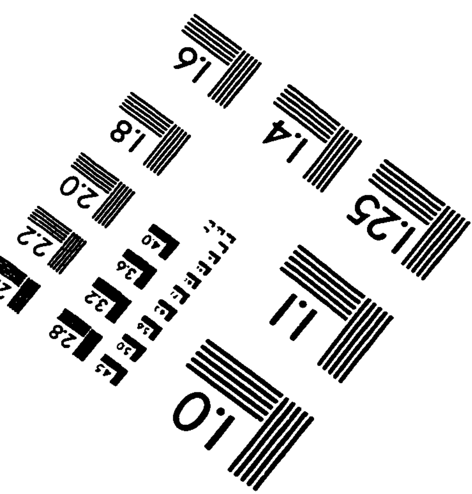
IMAGE EVALUATION TEST TARGET (QA-3)

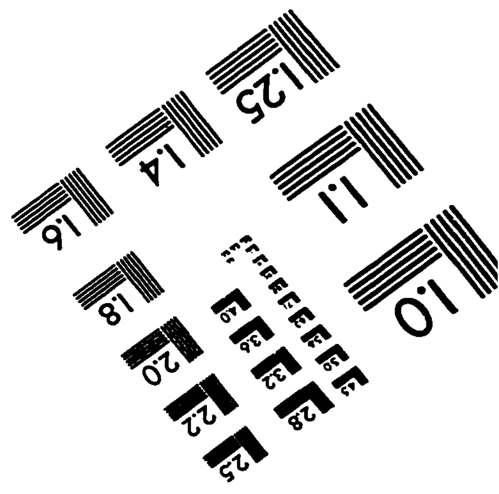












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.