From the Even Cycle Mystery to the L-Matrix Problem and Beyond

by

Michael Brundage

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

Master of Science

University of Washington

1996

Approved by _____

Chairperson of Supervisory Committee

_____

Program Authorized
to Offer Degree _____

Date _____

Master's Thesis

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature _____

Date _____

TABLE OF CONTENTS

TABLE OF CONTENTS (*continued*)

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author wishes to express sincere appreciation to Professor Klee for recommending this research topic and for his tremendous help. Thanks also to Professor Grünbaum for his assistance in the preparation of this manuscript, and to Yvonne Quirmbach and Sandra Williams for their support.

Introduction

Computational complexity is a rapidly growing field of mathematics. With new knowledge, many new problems have arisen; of these, those which are NP-complete are of singular interest. This thesis discusses several problems which are in some sense near the "boundary" of NP-completeness. We focus on two related problems: the Even Cycle Problem for directed graphs, and the L-Matrix Problem. We explore relationships among these two and other problems in computational complexity, both solved and unsolved. Appendices on the theory of NP-completeness and directed graphs are provided for readers unfamiliar with these topics, and an additional appendix lists the complexities of many relatives of the Even Cycle Problem, including those known to be polynomial-time equivalent to it.

An even cycle is a simple cycle of even length (one that uses an even number of edges). The Even Cycle Problem for directed graphs asks if there an efficient (polynomial-time) algorithm to answer the following:

Instance: A directed graph.
Question: Does the digraph have an even cycle?

We will discover that the Even Cycle Problem is equivalent to many other problems, such as the L-Matrix Problem from mathematical economics:

Instance: A square matrix.
Question: Is the matrix an L-matrix?

An L-matrix is a matrix with signed entries such that every real matrix with the same size and sign pattern as it has linearly independent columns.

These problems are all close to the boundary of NP-completeness in the sense that they strongly resemble certain problems known to admit polynomial-time algorithms, yet they also strongly resemble other problems known to be NP-complete. For example, it is easy to construct a polynomial-time algorithm to test a directed graph for the presence of odd cycles and also for the presence of even closed walks. However, localized version of these problems (e.g., looking for cycles of either parity passing through a given vertex) are NP-complete.

Recognizing L-matrices is similar to recognizing S-matrices and matrices with signed permanent — both problems admit efficient algorithms — yet in the rectangular case (even for "almost square" matrices), recognizing L-matrices is NP-complete. The complexity of the Even Cycle Problem and the L-Matrix Problem for square matrices remain unknown.

In the first chapter, we explore the Even Cycle Problem in greater depth, and also describe a few related graph-theoretic problems. The second chapter focuses on the L-Matrix Problem, affiliated concepts, and connections with the Even Cycle Problem for directed graphs. We journey beyond these topics in the third chapter, examining questions about permanents, Pfaffians, and labelled digraphs, and extending results of the previous two chapters to problems with a polytopal flavor. This final chapter concludes with a discussion of open problems and suggested topics for future research.

Chapter 1: Even Cycles in Directed Graphs

Introduction to: Even Cycles in Directed Graphs

We will use terminology and notation consistent with those defined in Appendix B. Unless otherwise stated, we will use **digraph** to mean a finite, simple, directed graph. First, we examine the similarities and differences of walks, trails, and cycles in digraphs, discussing algorithms to detect them. Next, we investigate conditions on the number of edges, connectivity, and/or vertex degrees of a digraph which guarantee the presence of an even cycle. We then describe the class of even digraphs and conclude the chapter with a summary and a few other results on certain restricted versions of the Even Cycle Problem.

Walks, Trails, and Cycles

For an arbitrary digraph G, consider the following questions:

(1e) Does G contain an even closed walk?
(2e) Does G contain an even closed trail?
(3e) Does G contain an even cycle?

By replacing the word "even" with "odd," we obtain similar questions (1o), (2o), and (3o). We will also explore localized versions of each, obtained by specifying a vertex v of G and asking (for example):

(1o-loc) Does G have an odd closed walk passing through the vertex v?

The difficulty of detecting even cycles in digraphs apparently was observed first by D.H. Younger. [Younger, 1973] It is still not known whether an efficient algorithm exists for this problem, or whether it is NP-complete. The same is true for the corresponding problem for even closed trails (2e), and in fact, these two problems are equivalent. First let us prove:

Theorem (Klee, Ladner, and Manber, 1984): (1o), (2o), and (3o) all admit polynomial-time solutions.

Proof: Klee, et al. describe an efficient algorithm which solves these and (1e) by solving (1e-loc) and (1o-loc). First we demonstrate the equivalence of (1o), (2o) and (3o) by observing that a digraph has an odd cycle if and only if it has an odd closed walk.

One direction is clear: Every cycle is a closed walk. On the other hand, if W is an odd closed walk $x_0\, x_1 \ldots x_k$ ($x_k = x_0$ and k is odd), and if W is not already an odd cycle, then there are two vertices repeated in this sequence. Choose i as large as possible subject to the condition that $x_i = x_j$ for some $i < j \leq k$. There is a unique such i and j, and now $x_i\, x_{i+1} \ldots x_{j-1}\, x_j$ is a cycle. If it is an odd cycle, we are done. If not, then j-i is even and the sequence $x_0\, x_1 \ldots x_{i-1}\, x_i\, x_j\, x_{j+1} \ldots x_k$ formed by removing the cycle is a shorter odd closed walk. Repeated application of this procedure eventually produces an odd cycle because the digraph is finite. Notice that this proof hinges on the fact that no odd integer can be represented as a sum of two even integers. In contrast, an even integer may be written as a sum of two odd integers, and so a digraph may contain an even closed walk but have no even cycles — for example, the digraph formed by the one-point union of two odd cycles.

Consequently, to solve any of (1o), (2o), and (3o) in polynomial time it suffices to construct an efficient algorithm to determine whether a graph has an odd closed walk. In fact, we will show the stronger result that (1e-loc) and (1o-loc) admit polynomial-time solutions. However, the remaining localized versions are all NP-complete.

In this algorithm, each edge enters the computation at most twice (once for each label that can be "active" on the head vertex of the edge). Thus the computational complexity for solving the localized problems (1o-loc) and (1e-loc) is linear in the number of edges of the digraph (quadratic in the number of vertices). Therefore, determining whether a digraph has an odd cycle (3o) can be solved by an algorithm with time complexity cubic in the number of vertices.

The algorithm is as follows: We are given a digraph G and a vertex v, and ask whether there is an odd (or even) cycle passing through v. Starting at v, breadth-first search through the digraph, keeping track of the parities of lengths of walks discovered. Begin by applying the label 0 to vertex v. Apply the label 1

to the vertex w to indicate the discovery of an odd walk from v to w; similarly, label w with 0 when there is an even walk from v to w. Labels are never removed, and a vertex may have both labels. When attaching a new label to a vertex, the label remains active until the vertex has been "scanned." Scanning a vertex x consists of inspecting each neighbor y of x to determine whether the recent addition of a new label to x justifies the attachment of a new label to y. The list of all vertices with active labels is kept in a queue Q, which is empty at the beginning of our computation and at the end. When the algorithm terminates, each vertex w carries a label if and only if there is a walk of that parity starting at v and ending at w.

Figure 1 gives the algorithm in pseudo-code. It consists of two functions, the main program and a subsidiary function for applying labels to vertices. Denote the set of all [active] labels of the vertex x by L(x) [A(x)]. The queue Q contains all vertices with active labels, and there are functions add_to_Q(x) and remove_from_Q() which add x to the rear of Q or remove and return the front element of Q, respectively. The functions add_element(element, set) and is_element(element, set) add the element to the set, or return 1 [0] if the element is [not] a member of the set, respectively. Each label is treated as a boolean value (0 or 1), and !s denotes the logical complement of s (read "not s").

We assume that we are given the source vertex v and the digraph in a manner such that the neighbors of a vertex can be computed efficiently. Table I lists the contents of the queue as the algorithm is run on the sample digraph in figure 2. Each vertex is followed by its currently active labels.

```
# subsidiary function to add labels to vertices
label(x, y, s) {
     if(is_element(!s, A(x)) and !is_element(s, L(x))) {
          add_element(s, A(y));
          add_element(s, L(y));
          if(!is_element(y, Q)) {
               add_to_Q(y);
          }
     }
}

# main algorithm
main() {
     for each neighbor y of v {
          add_element(1, A(y));
          add_element(1, L(y));
          add_to_Q(y);
     }
     while Q is nonempty {
          x = remove_from_Q();
          for each neighbor y of x {
               label(x, y, 0);
               label(x, y, 1);
               A(x) = empty set;
          }
     }
     if(is_element(0, L(v))) {
          print “Vertex v lies on an even closed walk”;
     }
     if(is_element(1, L(v))) {
          print “Vertex v lies on an odd closed walk”;
     }
}
```

Figure 1: An efficient algorithm for solving (1e-loc) and (1o-loc).

Table I: Active labels of vertices in the queue in a sample run.

Step	Vertices in queue (with active labels)			
0	empty			
1	1 (1)	2 (1)	3 (1)	
2	2 (1)	3 (1)	4 (0)	5 (0)
3	3 (1)	4 (0)	5 (0)	1 (0)
4	4 (0)	5 (0)	1 (0)	
5	5 (0)	1 (0)	v (1)	
6	1 (0)	v (1)		
7	v (1)	4 (1)	5 (1)	
8	4 (1)	5 (1)	2 (0)	3 (0)
9	5 (1)	2 (0)	3 (0)	v (0)
10	2 (0)	3 (0)	v (0)	
11	3 (0)	v (0)		
12	v (0)			
13	empty			

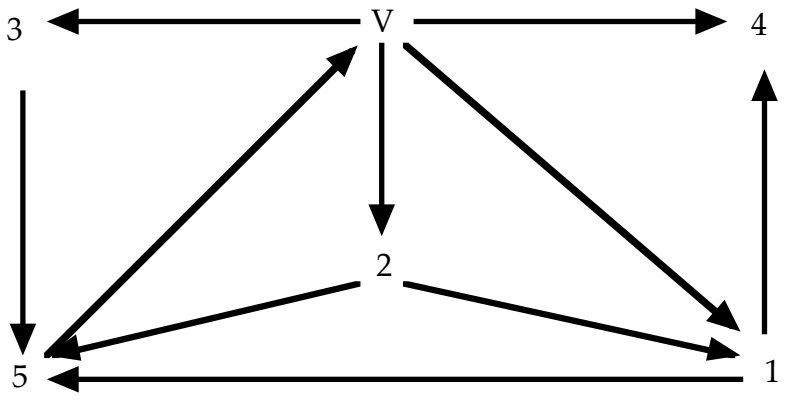

Figure 2: An example digraph with source vertex v.

As already mentioned, the remaining localized problems are all NP-complete. This situation is in stark contrast to the analogous questions for **un**directed graphs, all of which (local and global) admit polynomial-time solutions! [La Paugh and Papadimitriou, 1984] For digraphs, the computational complexity of (2e) and (3e) remains unknown, but these two problems are equivalent:

Theorem (Klee, Ladner, and Manber, 1984): Even cycles can be detected efficiently if and only if even closed trails can.

Proof: We describe a polynomial-time algorithm A which takes a digraph G and constructs another digraph H which has an even closed trail if and only if G has an even cycle. Similarly, we describe an algorithm B which constructs a digraph with an even closed trail if and only if the original graph contains an even cycle. These algorithms are illustrated in figure 3. Because each algorithm runs in polynomial time, detecting even cycles in directed graphs is equivalent to detecting even closed trails in directed graphs.

In algorithm A, each vertex of G is replaced by three vertices x_0, x_1 and x_2, and two edges (x_0, x_1) and (x_1, x_2). Each edge (x, y) of G is replaced by the edge (x_2, y_0) in H. Clearly, each cycle of length k in G is transformed into a cycle of length 3k in H. When k is even, this cycle is an even closed trail in H corresponding to the even cycle in G.

On the other hand, suppose T is a closed trail in H. The successive edges of T form a sequence of pairwise edge-disjoint paths of length three. The first two edges of each such path arise from a vertex of G, and the third edge from an edge of G. We claim that no vertex of T is ever repeated, and hence T is actually an even cycle. Consider three successive edges (x_0, x_1), (x_1, x_2), (x_2, y_0) in T, corresponding to an edge (x, y) of G. Now, x_2 is not repeated in T because (x_1, x_2) is the only edge of H into x_2, and no edge of T is repeated (T is a trail, by assumption). Similarly, x_1 is not repeated. But then the vertex x_0 cannot be repeated either, because (x_0, x_1) is the only edge issuing from x_0. Thus, each closed trail T in H is, in fact, a cycle of length 3k, and arises from a cycle of length k in G. Because k and 3k have the same parity, we have constructed the desired algorithm A.

Algorithm B produces the **edge graph** of G: For each edge (x, y) of G, there is a vertex xy in its edge graph, and (uv, xy) is an edge in the edge graph of G only when v = x. Each closed trail of length k in G involves k distinct, successively adjacent edges, thus leading to a cycle of length k in its edge graph. Conversely, each cycle of length k in the edge graph of G is a sequence of k distinct, adjacent vertices, and hence must come from a sequence of k distinct, successively adjacent edges in G (that is, a closed trail of length k in G).

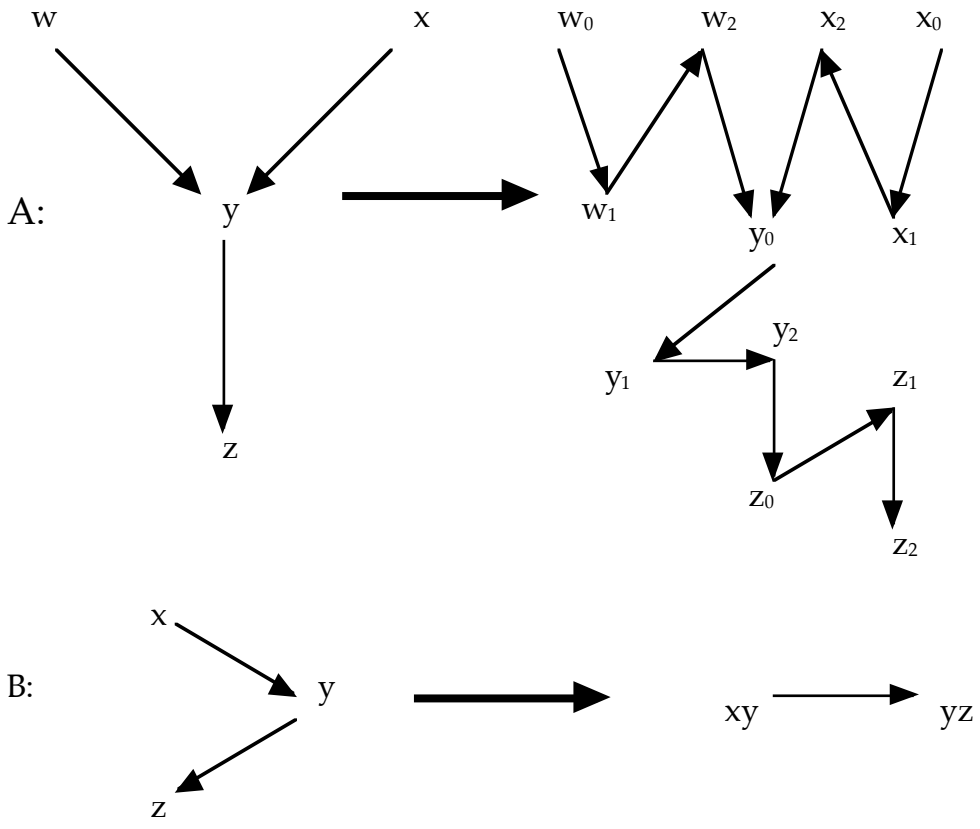

Figure 3: Algorithms demonstrating the equivalence of detecting even cycles and even closed trails in directed graphs.

Connectivity Results

Although the complexity of finding even closed trails and even cycles remains unknown, there are many results of interest which provide easily checked conditions for a digraph to have an even cycle or an even closed trail. Intuitively, one would expect that a large number of edges (with respect to the number of vertices) would guarantee the existence of an even cycle. That this is indeed the case was established by Chung, Goddard, and Kleitman in 1994:

Theorem: Every strongly connected directed graph with n vertices and at least $\lfloor (n+1)^2/4 \rfloor$ edges has an even cycle, and this bound is best possible in the sense that there exist strong digraphs with fewer edges and no even cycles.

Of course, this theorem allows for the possibility that many edges are clumped together. In order to obtain better bounds, we need to exlude these cases by better distributing the edges among the vertices. One way to accomplish this task is to place lower or upper bounds on the vertex degrees. For example,

Theorem (Koh, 1976): If each vertex of a digraph G has outdegree at least two, then G contains two edge-disjoint cycles with at least one common vertex. Hence G contains an even closed trail.

Proof: If each vertex of a digraph has outdegree at least one, then the digraph has a cycle. To produce one, start from any edge (x_0, x_1) and continue to add edges (x_k, x_{k+1}) until some vertex is repeated in the sequence $x_0, \ldots, x_{k+1}$. Such edges exist because each vertex has outdegree at least one; until we repeat a vertex, there is always at least one outgoing edge. The graph is finite, so eventually some vertex must be repeated, and then we have found a cycle of the digraph.

Now if every vertex of G has outdegree at least two, start with some cycle C_1, and let G_1 denote the sugraph of G formed by removing all the edges of C_1. Each vertex of G_1 has outdegree at least 1, so G_1 also has some cycle, C_2. Now let G_2 be the sugraph of G_1 formed by removing the edges of C_2. If C_2 is vertex-disjoint from C_1, then every vertex of G_2 also has outdegree at least 1, and we can continue this process. Eventually we obtain two cycles with at least one vertex in common, but with no edges in common.

Given any two such cycles, if either is even then it is an even cycle, and thus an even closed trail. Otherwise, produce an even closed trail by starting at a vertex they have in common and then following first one cycle and then the other.

However, the analogous result for even cycles is not true. In fact, there exist digraphs with arbitrarily large in- and outdegrees, yet no even cycles:

Theorem (Thomassen, 1985): For each positive integer n there are digraphs D_n and G_n having no even cycles, even though

(i) D_n is strong and each vertex of D_n has outdegree n

(ii) each vertex of G_n has indegree n and outdegree at least n

Proof: Let D_1 be an odd cycle, and create D_n from D_{n-1} by induction as follows: For each vertex v in D_{n-1} add a set Y_v of n vertices and one more additional vertex v'. Add the edge (v, v') and for each y in Y_v add the edges (v', y) and (y, v). Also, for each neighbor z of v in D_{n-1} and for each member y of Y_v, add the edge (y, z). Then D_n is strongly connected by construction, and each vertex of D_n has outdegree exactly n. Every cycle in D_n corresponds to a cycle of the same parity in D_{n-1}, so D_n has no even cycles (because D_{n-1} doesn't by the induction hypothesis).

Now form the digraph E_n by reversing all edges of D_n, so each vertex of E_n has indegree exactly n. Form G_n by taking vertex-disjoint copies of E_n and D_n and adding all the edges (e, d) for each vertex e in E_n and d in D_n. Thus, each vertex of G_n has outdegree and indegree at least n. G_n has no even cycles because any cycle must lie entirely in one of its strong components, E_n and D_n, but neither of these has any even cycles.

Any cycle of a digraph lies in one of its strongly connected components, and efficient algorithms exist for finding these components. [Tarjan, 1972] Therefore, we lose nothing if we restrict our attention to strongly connected digraphs. In fact, Thomassen asked whether the properties of D_n and E_n in this theorem can be combined into a single graph: Does there exist a positive integer n such that every strongly connected digraph with indegree and outdegree at least n has an even cycle? Partial answers to this question exist. For example,

every d-regular digraph, d≥7, has an even cycle. [Friedland, 1989] Except when d=7, this theorem is subsumed by a result proved in an entirely different manner:

Theorem (Alon/Linial, 1989): If G is a digraph with minimum outdegree at least d and maximum outdegree at most D, and if the inequality $e(Dd + 1)(1 - 1/k)^d < 1$ holds, then G has a cycle whose length is divisible by k. In particular, if $e(dD + 1) < 2^d$, then G has an even cycle.

Recently, Thomassen improved this bound still further:

Theorem (Thomassen, 1992): Every d-regular digraph, d≥3, has an even cycle. As a (nontrivial) consequence, every 3-connected digraph has an even cycle.

However, this is the best possible in the sense that there exists a 2-regular, 2-connected digraph with no even cycles. [Koh, 1976; also Boyd, cited in Thomassen, 1986] This digraph, pictured in figure 4, is neatly described as the union of two 7-cycles 0123456 and 0362514. A computer search has shown that every other 2-connected, 2-regular digraph with rotational symmetry and fifteen or fewer vertices has an even cycle. [McKay, 1989, cited in Thomassen, 1993]

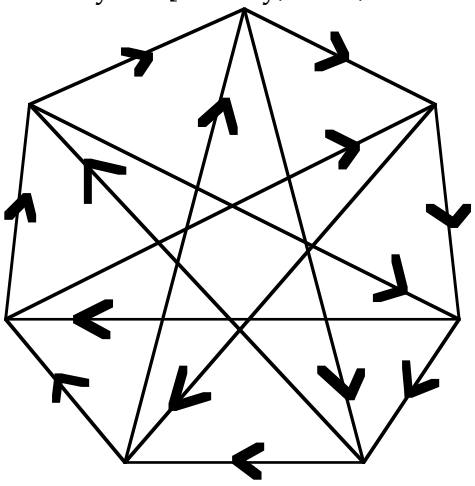

Figure 4: The only known 2-connected digraph with no even cycles.

Even Digraphs

Many results, including the proof of Thomassen's theorem for 3-connected digraphs, rely on another class of digraphs. A digraph is **even** if, whenever each of its edges is assigned a weight 0 or 1, the digraph has a cycle of even weight. [Vazirani and Yannakakis, 1989] Equivalently, every subdivision of the digraph has an even cycle. In fact, there is the following characterization of even digraphs, in terms of a class of "forbidden" subgraphs:

Theorem (Thomassen, 1992): A digraph G is even if and only if G has a weak odd double-cycle. (A **weak odd double-cycle** is any digraph obtainable from an odd dicycle by splitting vertices or subdividing edges.)

The recognition problem for even digraphs is equivalent to the Even Cycle Problem. [Seymour/Thomassen, 1987] As with even cycles, there are many theorems asserting that digraphs with certain connectivity or vertex degrees are even. One such theorem is that every strongly connected digraph with minimum in- and outdegrees at least three is even. In contrast to the situation for even cycles, it is known that there are infinitely many 2-connected digraphs which are not even. [Thomassen, 1992]

Restricted Versions

Other restricted versions of the Even Cycle Problem admit polynomial time solutions. For example, Thomassen describes an efficient algorithm for determining whether a planar digraph has an even cycle. [Thomassen, 1993] Galluccio and Loebl found a different polynomial-time algorithm for planar graphs. [Galluccio and Loebl, 1994]. They recently extended their algorithm in several ways, coming tantalizingly close to solving the Even Cycle Problem in its full generality.

First, they described a polynomial-time algorithm for detecting even cycles in digraphs which are $K_{3,3}$ or K_5 free. [Galluccio and Loebl, 1995] (An undirected graph is **H-free** when it does not contain a subgraph contractible to H. A digraph is **H-free** when its underlying undirected graph is H-free.) Their algorithm actually extends to detecting cycles of any modularity in a directed

graph, similar in spirit to the theorem due to Alon and Linial previously mentioned.

They followed up with a proof that, given a fixed but arbitrary surface, one can determine in polynomial time whether a digraph embeddable on that surface has an even cycle. [Galluccio and Loebl, 1996] Thus this result includes the previous one for planar digraphs, and constitutes a significant step toward detecting even cycles in arbitrary digraphs.

Conclusion

The complexity of the Even Cycle Problem remains unknown, although many similar problems for directed graphs admit polynomial-time solutions. By placing bounds — either on the number of edges in the digraph, or the number of edges entering or leaving each vertex, or the number of paths between any two vertices, or some combination of these — we can guarantee the existence of an even cycle. When these conditions are easily checked, we thus have an efficient algorithm for solving the Even Cycle Problem in these cases. Finally, we remark that algorithms for determining whether a digraph has an even cycle exist, and although not efficient, such algorithms do have advantages for our purposes over those that find every cycle in the digraph. [Karmarkar, 1984]

Chapter 2: L-Matrices and Sign-solvability

Introduction to: L-Matrices and Sign-solvability

Certain qualitative problems (including classic supply and demand scenarios in economics; for examples see Samuelson, 1947) lead naturally to questions of sign-solvability, in which one considers only whether the values of the variables in a linear system are positive, negative, or zero. First, we will define sign-solvable linear systems; then we introduce L- and S-matrices, which play important roles in studying sign-solvability. We continue with a foray into the complexity of the recognition problem for sign-solvability, and conclude this chapter with a proof of the equivalence of the square L-Matrix Problem and the Even Cycle Problem.

Sign-solvability

Sign-solvable systems were first studied in 1962 by the economist Lancaster (inspired in part by Samuelson's book). Basset, Maybee, and Quirk (1968) demonstrated that sign-solvability translates into certain properties of directed graphs. Building upon their work, Klee, Ladner, and Manber (1984) were the first to study in depth the relations between the even cycle problem and the recognition problem for sign-solvable systems. Their paper established several fundamental computational complexity results and spurred widespread interest in these problems. Key to their study of sign-solvable systems are matrices known as L- and S- matrices (for "linearly independent" and "simplex," respectively).

Definition: Two matrices A, B have the same **sign pattern** when they are of the same size and corresponding entries in the same row and column have the same sign. In this case we write sgn(A) = sgn(B), and write Q(A) for the **qualitative class** of matrices, all of which have the same sign pattern as the matrix A.

We can also speak of qualitative classes of vectors (thought of as nx1 matrices), and then we have:

Definition: The linear system Ax = b is **sign-solvable** when

(1) For every matrix C in Q(A) and every vector d in Q(b), the system Cy = d is solvable,

and

(2) every solution vector y is in the same qualitative class, which we write Q(Ax = b).

Although it is not immediately clear from the definition, every vector in Q(Ax = b) is a solution to some system satisfying the first condition. (Composing the matrix with any nonnegative, invertible diagonal matrix does not change its qualitative class, and hence every solution vector can be achieved.) Moreover, sign-solvable systems do exist; one example is depicted in figure 5. Every matrix A with the same sign pattern as the matrix on the left, and every vector b with the same sign pattern as the vector on the right give rise to a linear system Ax = b with a solution vector x whose sign pattern is $(+, +)^T$.

$$\begin{pmatrix} + & - \\ - & - \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ - \end{pmatrix}$$

Figure 5: A sign-solvable system.

When the solution vectors in Q(Ax = b) have no zero coordinates, we say the system is **strongly sign-solvable**. [Klee, 1987] Throughout our discussion, we will assume the coefficient matrices have no rows which are entirely zero, since the addition or removal of such rows does not change whether a matrix is an L- or S-matrix, and all-zero rows only make the statements of results more complicated.

L-matrices

L-matrices arise naturally from sign-solvable homogenous systems in the following way: Because 0 is a solution vector to Ax = 0 and is the only vector with its sign pattern, the system Ax = 0 is sign-solvable if and only if Q(Ax = 0) consists only of the vector 0. But Q(Ax = 0) = {0} if and only if every matrix in Q(A) has linearly independent columns, motivating the following definition:

Definition: A is an **L-matrix** if every matrix in Q(A) has linearly independent columns.

Some authors define an L-matrix as one having linearly independent rows, a definition equivalent to ours via transposition.

$$\begin{pmatrix} + & + & + & - \\ + & + & - & + \\ + & - & + & + \end{pmatrix}$$

Figure 6: A 3 x 4 L-matrix.

As we will soon explain, square L-matrices are of particular interest because of their connections to even cycles in directed graphs. From the definition, it follows that every square L-matrix (also called a **sign-nonsingular** matrix) is invertible, and hence has nonzero determinant. The sign of the determinant of every matrix in the qualitative class of an L-matrix must be the same, or else there would exist some matrix in the class with determinant zero, hence singular. Therefore, square L-matrices are precisely those matrices with **signed determinant**.

For example, the lower Hessenberg nxn matrices — with positive ones on the super-diagonal, negative ones on and below the main diagonal, and zeros everywhere else — are square L-matrices with signed determinant $(-1)^n$.

Because every square L-matrix has some nonzero term in its determinant expansion, each row and column has at least one nonzero entry. Consequently,

there is a permutation of the columns and rows such that these nonzero terms are on the diagonal. Permuting rows and columns preserves the property of being an L-matrix, so without loss of generality we may assume every L-matrix has only nonzero diagonal entries.

Moreover, if we simultaneously multiply by -1 any row of the matrix A and the corresponding entry in b for the sign-solvable system Ax = b, we obtain a new sign-solvable system such that every diagonal element of the matrix has the same sign. We will use this "standard form" of an L-matrix in the next chapter to establish the equivalence of the Even Cycle Problem and the square L-Matrix Problem.

S-matrices

Proceeding to sign-solvable inhomogeneous systems Ax=b for some nonzero b, we first observe that A must be an L-matrix: If not, then there would be a nonzero vector y in the null space of A so that the equation A (x + cy) = Ax + 0 = b holds for all real numbers c. However, x+cy is not in Q(x) for all values of c, contradicting sign-solvability.

Now suppose [A –b] z = 0 is sign-solvable, where [A –b] is the augmented matrix obtained by appending the column vector b to the matrix A. Because A is an L-matrix, if the last coordinate of z is zero then all of z is. Otherwise, Az′ = cb, where z′ is the shortened vector obtained by removing the last coordinate of z. Call this coordinate c, and observe cz′ is in Q(Ax = b). We are concerned only with the signs of these vectors, so we conclude that Ax = b is sign-solvable if and only if the null space of [C –d] for every C in Q(A) and d in Q(b) is a line through the origin.

When A is a square nxn matrix, the augmented matrix [A –b] has dimensions nx(n+1). Solutions to the system can be found by applying Cramer's Rule, computing determinants of each of the submatrices obtained by deleting a column in the augmented matrix. Some routine but tedious calculations show that the system is strongly sign-solvable if and only if A is an L-matrix and every one of these submatrices is an L-matrix. This discovery motivates the following definition:

Definition: Let A be an nx(n+1) matrix. A is an **S^*-matrix** if and only if the null space of each matrix in Q(A) is a line through the origin and an open orthant in R^n, the same orthant for all matrices in Q(A).

Equivalently, we could say that A is an S^*-matrix if and only if there exists a vector z such that the Q(Ax = 0) is the union of Q(-z), Q(z), and the zero vector. Every S^*-matrix and every submatrix obtained by deleting a column of an S^*-matrix is an L-matrix. The inhomogeneous linear system Ax = b is strongly sign-solvable if and only if the matrix [A –b] is an S^*-matrix.

When the null space is a line passing through the open positive orthant in particular (that is, every vector in the null space has all its entries the same sign), the matrix is called an **S-matrix**. Thus, S-matrices are a special type of S^*-matrix, and every S^*-matrix can be converted into an S-matrix by replacing some of its columns with their negatives. We can think of S-matrices as a "normal form" for S^*-matrices. As with square L-matrices, there are several equivalent conditions for an nx(n+1) matrix to be an S-matrix. For example, a matrix is an S-matrix if and only if its columns form the vertices of an n-simplex with the origin in its (relative) interior (inspiring the "S" in S-matrix). [Klee and Ladner, 1981]

Although we will consider these matrices primarily in terms of their relevance to the Even Cycle Problem, sign-solvable systems, L-, and S-matrices form an area of study in their own right. The interested reader can find additional details in the recent book by Brualdi and Shader (1995). However, we mention a few additional facts before continuing with our investigation of the recognition problem for L- and S-matrices.

If Ax = b is sign-solvable, then so is Ax = c, where c is formed from b by possibly replacing some of its nonzero entries with 0. As already noted, the addition or deletion of all zero rows does not alter whether a matrix is an L- or S-matrix. Many results focus on the placement of zeros in these matrices, in part because their structure turns out to be quite sensitive to them — after all, 0 is a point of discontinuity in passing from the positive sign cone to the negative one. For example, the fact that every S^*-matrix has at least one row with precisely two nonzero entries is used to establish their recursive structure and show that every nx(n+1) S^*-matrix has at least 2n and no more than n(n+3)/2 nonzero entries (both bounds are sharp). [Klee, 1987]

Recognition

Now that we have been introduced to the major players in this arena, let us prove that recognizing sign-solvability is equivalent to recognizing L- and S-matrices. Standard notation is to write A[a, b] for the submatrix of A with rows indexed by the set of indices a and columns indexed by b. We will also write A(a, b) for A[a′, b′] (where a′ denotes the complement of the set a).

Theorem (Manber, 1982): Let A be an mxn matrix, b an mx1 column vector, and suppose x is a solution of Ax = b. Define the index sets

$j = \{k : x_k \neq 0\}$ and $i = \{k : a_{kl} \neq 0 \text{ for some } l \text{ in } j\}$

Then Ax = b is sign-solvable if and only if

(1) the matrix (A[i, j] –b[i, {1}]) formed by appending a subvector of –b to a submatrix of A is an S^*-matrix,

and (2) A(i, j) is an L-matrix. (Every 0xk matrix is vacuously an L-matrix).

Proof (adapted from Brualdi and Shader, 1995):

Without loss of generality, suppose i is the set of indices from 0 to r inclusive, and j is the set from 0 to s. Note that if s is zero then r is also. By the definitions of i and j, A can be written in block form as

$$\begin{pmatrix} A_1 & A_3 \\ 0 & A_2 \end{pmatrix}$$

where A_1 is an rxs matrix with no zero rows. Let $b' = (b_1\ b_2\ \ldots\ b_r)^T$. Then the linear system Ax = b is equivalent to the simultaneous equations

$$A_1 y + A_3 z = b'$$
$$A_2 z = 0$$

and it is necessary to show that Ax = b is sign-solvable if and only if the matrix $[A_1\ -b']$ is an S^*-matrix, and that A_2 is an L-matrix.

First, suppose Ax = b is sign-solvable, so A is an L-matrix. Then every vector $(y\ z)^T$ in Q(Ax = b) must have z=0. Then the first equation is just $A_1\ y = b'$ and is sign-solvable, so A_1 is also an L-matrix. Because the columns of A_1 are linearly independent, the number r of columns of A_1 must exceed the number s of rows. In fact, if r were strictly greater than s, then because the set of linearly independent rows of A_1 has cardinality at most s, we could find a matrix in $Q(A_1)$ which agrees with A_1 in these independent rows but when multiplied by y does not yield a vector in the qualitative class of b′ (which has r rows), thus contradicting the sign-solvability of $A_1\ y = b'$. So r = s, and consequently $[A_1\ \ -b']$ is an S^*-matrix. As already noted, z = 0 for every vector in Q(Ax = b). To show A_2 is an L-matrix, it remains to show that there is a solution to C u = 0 for every matrix C in $Q(A_2)$. But the fact that A_1 is a square matrix implies that the equation $A_1\ y = b' - A_3\ u$ always has a solution. Therefore, A_2 is an L-matrix.

Conversely, suppose $[A_1\ \ -b']$ is an S^*-matrix, and A_2 is an L-matrix. Then the equations above have a unique solution for each choice of matrices and vectors in the same qualitative classes as A_1, A_2, A_3, and b′, and thus for each C in Q(A) and d in Q(b) there is a solution w in Q(Ax = b) to Cw = d. Therefore, Ax = b is sign-solvable.

We conclude that to determine whether a linear system is sign-solvable, it is necessary to examine only these two matrices (formed in polynomial time from the elements of this system) and determine whether or not one is an S^*-matrix and the other an L-matrix.

Unfortunately (for economists, at least!), the recognition problem for rectangular L-matrices is NP-complete, even in the "almost square" case. [Klee, Ladner, and Manber, 1984] The proof is too long for inclusion here, but involves transforming L-matrices to a satisfiability problem in propositional logic. In contrast, S-matrices can be recognized in polynomial time (in fact, there is a recursive algorithm for constructing them). [Klee, 1987] The complexity of recognizing square L-matrices (the square L-Matrix Problem) remains unknown, but next we will prove it is equivalent to the Even Cycle Problem.

Cycles in Matrices

At first, sign-solvable systems and the L- and S- matrices may appear to have nothing in common with the problem of finding even cycles in directed graphs discussed in the previous chapter. Let us now establish a connection between them.

Recall that square L-matrices are precisely those matrices with signed determinant, and that every square L-matrix can be put into a "standard form" with all its diagonal elements negative. Now, the determinant of any matrix can be written as a sum of **cycles** $a_{i1\ i2}\ a_{i2\ i3} \cdots a_{i3\ in}\ a_{in\ i1}$. Define the sign of a cycle to be the product of the signs of its elements, and observe that when the determinant is nonzero, some cycle must be nonzero. The matrix has nonzero signed determinant if and only if some cycle has nonzero sign and no two cycles have opposite sign. For an L-matrix in standard form, the cycle consisting of diagonal elements is nonzero, so every cycle in an L-matrix must have nonpositive sign.

In addition, Bassett, Maybee, and Quirk (1968) proved that

Theorem: The system Ax = b is sign-solvable if and only if both

(1) Every cycle in A is nonpositive, and

(2) whenever $b_j > 0$, every **chain** $a_{i1\ i2}\ a_{i2\ i3} \cdots a_{i(n-1)\ j}$ is nonnegative.

In particular, because Ax = 0 is sign-solvable if and only if A is an L-matrix, we conclude that A is an L-matrix if and only if A has no positive cycles.

Signed Digraphs

The term "cycle" is highly suggestive, and aptly chosen. Given any nxn square matrix, we can construct a digraph D(A) with adjacency matrix A by taking the vertices 1 through n and adding an edge (i, j) whenever the matrix element $a_{i\ j}$ is nonzero. Cycles in the matrix correspond to cycles in the digraph, and conversely so. We can make a **signed digraph** by also labelling each edge with the sign of the corresponding entry $a_{i\ j}$. Define the sign of a cycle in a signed digraph to be the product of the signs of the labels of the cycle. Every cycle in the matrix A is nonpositive if and only if every cycle in the signed digraph D(A) is negative.

Now that we know square L-matrices are precisely those with nonpositive cycles, signed digraphs provide a convenient graph-theoretical tool for

characterizing square L-matrices. However, an L-matrix in standard form has nonzero diagonal entries, and hence the digraph formed from it will have loops at each vertex. We want to work with simple digraphs, so remove these loops from the digraph. This is equivalent to working with the matrix B+I and the digraph D(B+I), where B is a matrix in Q(A) with all its entries –1, 0, or +1. (In fact, because we can always choose such a representative from the qualitative class of a matrix, the problem of recognizing L- and S-matrices is purely combinatorial in nature.) To avoid difficult wordings of the results presented in this chapter, we will henceforth write D(A) for this digraph, instead of the digraph with loops.

Now, apply our characterization of L-matrices A in terms of their cycles to the matrices –A with positive main diagonal. Then A is an L-matrix if and only if the sign of each cycle of length k is $(-1)^k$. Therefore,

Theorem: Let A be a square {–1, 0, +1} matrix with positive diagonal. Then A is an L-matrix if and only if the digraph D(A) has no even cycles.

This theorem establishes the firm connection between recognizing square L-matrices (the square L-Matrix Problem) and determining whether a directed graph has an even cycle (the Even Cycle Problem).

Conclusion

Graph theoretic problems are often easier to work with than matrix problems in terms of computational complexity. Thus, one significance of this theorem is that it provides a mechanism for establishing the computational complexity of certain matrix-theoretic problems by using graph-theoretic tools, thereby extending the reach of computational complexity. Another is that, as we have seen, many problems in proximity to the Even Cycle Problem for digraphs or the L-matrix Problem for square matrices admit efficient solutions, while many others do not. We conclude that these two (equivalent) problems are in some sense near the "boundary" of NP-completeness. This proximity to problems of widely differing complexity could have consequences for determining whether NP = P, and is an interesting phenomenon in its own right.

Chapter 3: Beyond

Introduction to: Beyond

The Even Cycle Problem leads to many other avenues of research, on such varied topics as labelled digraphs, Pfaffian orientations, permanents, polytopes and more. In this chapter, we briefly explore some of these generalizations and side streets, and conclude our tour with proposed topics for future research.

Balanced Labellings

A **balanced labelling** of a digraph is an assignment from {-1, 0, +1} to its vertices (one label per vertex) such that not all labels are 0, and every vertex labelled ±1 has at least one neighbor of opposite sign. Let us now prove a digraph has a balanced labelling if and only if it has an even cycle. Unfortunately, there are too many possible labellings to be checked for this result to lead to an efficient algorithm for detecting even cycles.

Theorem (Klee, n. d.): Given a digraph G, the following are equivalent:

(i) G has an even cycle

(ii) G admits a balanced labelling

(iii) G admits a balanced labelling in which the neighbors of every vertex labelled 0 are also labelled 0.

Proof: Given a balanced labelling, one can find an even cycle by starting with a nonzero label and following edges which alternate sign from one vertex to the next, until some vertex is repeated. The portion of the walk between the two appearances of the repeated vertex is an even cycle.

Conversely, given an even cycle, label its vertices alternatingly +1, -1 and then extend to a balanced labelling for the entire digraph as follows: While there is an edge (x, y) such that x is not yet labelled and y has a nonzero label, label x oppositely from y. When there are no more such edges (x, y), attach the label 0 to all remaining unlabelled nodes. The extended labelling is balanced (by

construction), and has the additional property that if a vertex is lablled 0, then so are all of its neighbors. Thus the three properties in the theorem are equivalent.

Distinguished Vertices

A vertex in a signed digraph is **distinguished** if every path ending at that vertex uses only positive edges. Any strong component of the graph which contains a distinguised vertex is also said to be distinguished. Manber proved several results concerning distinguished vertices. [Manber, 1982] For example, when the digraph is formed from an L-matrix, each distinguished component contains only one distinguished vertex and there are no paths starting in an undistinguished component and ending in a distinguished one. Armed with the concept of distinguished vertices and given any L-matrix A, one can find a vector b such that Ax = b is sign-solvable by choosing $b_i \geq 0$ if i is distinguished, and $b_i = 0$ otherwise.

Permanents

A formula for the **permanent** of a square matrix is deceptively similar to the well-known alternating sum for the determinant — simply remove the alternating sign from each term in the sum. However, this "minor" difference greatly increases the difficulty of computing the permanent. In fact, computing the permanent of a matrix is NP-hard, even for 0-1 matrices. [Valiant, 1979]

Because the formula for the permanent is superficially so similar to the determinant, many have sought an expression for one in terms of the other. Polya suggested that perhaps the permanent could be found by computing the determinant of another matrix obtained from the original by reversing the sign of some (possibly none) of its entries. [Polya, 1913] However, he also noted that this is not always possible.

In 1989, Vazirani and Yannakakis showed that determining whether the determinant and permanent of a matrix are equal is NP-hard. However, when we restrict the instances to only 0-1 matrices or matrices with nonnegative entries, they showed this problem is equivalent to the Even Cycle Problem. If we say that Polya's scheme, when applied to a 0-1 matrix A, is to change some +1 entries to -1, then Vazirani and Yannakakis also showed that determining

whether or not Polya's scheme can be applied to a square 0-1 matrix A to produce a matrix B such that det(B) = perm(A) is equivalent to the Even Cycle Problem.

Pfaffian Orientations

A [**perfect**] **matching** is a subset of the edges of a graph such that every vertex of the graph is incident to at most [exactly] one edge in the matching. In 1967, Kasteleyn described a polynomial time algorithm for computing the number of perfect matchings of any planar, undirected graph. His proof relied heavily on the idea of a **Pfaffian orientation** of the graph. Of course, any undirected graph can be converted to a directed one by arbitrarily assigning directions to its edges. To assign a Pfaffian orientation, first distinguish each cycle C in the graph G for which C has even length and G\C has a perfect matching. A Pfaffian orientation of G, if one exists, is an orientation such that when traversing each distinguished cycle, an odd number of edges are oriented in the direction of the traversal.

One reason Pfaffian orientations are of interest is that given a graph G with a Pfaffian orientation, if the matrix B is obtained from its adjacency matrix A by replacing with -1 those entries corresponding to edges which are reversed in the orientation (i.e., Polya's scheme), then the determinant of B is the square of the number of perfect matchings in G.

Every $K_{3,3}$-free graph has a Pfaffian orientation that can be found in polynomial time. [Little, 1974] On the other hand, deciding whether or not a bipartite graph has a Pfaffian orientation is equivalent to the Even Cycle Problem. [Vazirani and Yannakakis, 1989]

Interval Matrices

We can generalize sign-solvability by a modification of the qualitative class. When the qualitative classes are viewed as collections of matrices whose entries belong to one of three intervals, the two open intervals $(-\infty, 0)$ and $(0, \infty)$ together with the degenerate interval $\{0\}$, a natural generalization of sign-solvability (and other concepts related to sign patterns) is to consider **interval matrices**, whose entries belong to certain intervals. That is, given a set of

intervals or types of intervals, we define the "qualitative class" (with respect to that set) to be those matrices whose entries lie in those [types of] intervals.

The usual type of interval matrix found in the literature is a restriction to finite closed (or open) intervals. Then we can describe qualitative classes succinctly as $[A - B, A + B] := \{ M \in R^{nxn} \mid a_{ij} - b_{ij} \leq m_{ij} \leq a_{ij} + b_{ij}\}$ where A is the matrix whose entries are the midpoints of each interval and B is a nonnegative matrix determining the length of each interval (using strict inequalities when we wish to consider open intervals). In general, the computational complexity of recognizing whether every matrix in such a "qualitative class" is nonsingular is NP-hard. [Poljak and Rohn, 1993]

Cone-Systems

We mentioned in passing that an equivalent definition of an nx(n+1) S-matrix is that when its columns are treated as vertices in R^n, they determine an n-simplex with the origin in its interior. Continuing in this direction, observe that the columns of any matrix in the qualitative class Q(A) (for some matrix A) all lie in convex cones. This follows from the fact that for any vector x, Q(x) is a **sign cone** of vectors with the same sign as x in each of its coordinates.

When viewed in this way, the recognition problem for S-matrices becomes a question of determining whether the origin belongs to certain convex sets derived from the sign-cones. Klee, von Hohenbalken and Lewis proved the more general result that, given any mxn system $(A_1, A_2, \ldots, A_n)$ of finitely-presented cones, there are polynomial time algorithms to recognize whether the origin is in the vector sum of the cones and whether it is in their convex hull. As a consequence, the recognition problem for S-matrices admits a polynomial-time solution.

Their geometric approach is beautiful, and ties in with interval matrices as well. Unfortunately, interval matrices have upper and lower bounds for each entry, so each cone could require up to $O(2^m)$ generators. Sign-cones need only O(m) generators, so their algorithm solves the recognition problem for sign-cones in polynomial time, but not for general interval matrices. In general, the complexities of algorithms for polytopes depend on the representations of the polytopes — either as the convex hull of a set of vertices, or as the intersection of half-spaces determined by linear inequalities. Interval matrices, for example, can

be represented using O(m) linear inequalities. Perhaps their result can also be proved for cones presented in this way.

Also, their algorithm is not as efficient as the $O(m^2)$ algorithm for recognizing S-matrices [Klee, 1987], so other improvements may be possible. What happens for matrices whose columns belong to more general polytopes than just cones? What characterizations might these matrices have in terms of digraphs? What connections do they have to sign-solvability, or more general kinds of solvability?

Open Problems

Many of the generalizations mentioned in this chapter have not been fully investigated. Klee (personal communication) has suggested examining, for example, interval matrices using five types of intervals: $(-\infty, -1]$, $(-1, 0)$, $\{0\}$, $(0, 1)$, and $[1, \infty)$. In general, what happens when we fix a set of intervals and then define the qualitative classes using these intervals? When are all the matrices in such a class solvable? Stable? Are there analogs of L- and S-matrices for these classes?

Especially promising are the generalizations from sign-cones to polytopes. How can the considerable machinery from the theory of polytopes be brought to bear on this subject? Klee, von Hohenbalken, and Lewis (1993) successfully employed results from linear programming to establish the computational complexity of some of these systems, but this is only a beginning.

Even when we constrain ourselves to remain within the confines of the original problems concerning L-matrices, sign-solvability, and even cycles in directed graphs, there are many interesting unexplored avenues and unsolved problems. (Of course, there is always the main one: Determine the complexity of the Even Cycle Problem!) Many results for L-matrices and their ilk can be translated into results for directed graphs, and vice-versa. [e.g., Brualdi and Shader, chapter 6] What other results can be so translated? For example, what implications do k-connectedness or planarity of a digraph have for its adjacency matrix, and how do these implications affect our analysis of the square L-matrix problem? Can an efficient algorithm for one restricted problem (such as the polynomial-time algorithm for recognizing planar digraphs without even cycles)

be transformed into efficient algorithms for the others (perhaps an efficient algorithm for recognizing a certain subclass of L-matrices)?

It would be very surprising if the only 2-connected digraph with no even cycles is the one given in the first chapter. Perhaps there is only a finite list of such graphs; even if there are infinitely many, perhaps they can be described succinctly enough to allow a polynomial-time algorithm to test whether a given 2-connected digraph is one of them. If nothing else, surely upper bounds on the number of edges can be found, analogous to the bound for (1-)connected digraphs. [Chung, Goddard, and Kleitman, 1994]

Conclusion

The Even Cycle Mystery remains unsolved. In the course of investigating this thorny case, sleuths have discovered leads into many disparate areas of mathematics, and proximity to problems of widely varying difficulty and complexity. Although the case may never be completely solved, progress continues to be made as many exciting answers are found, and even more unanswered questions are raised.

Appendix A: Computational Complexity

Computational complexity is the study of algorithms. For an in-depth treatment of computational complexity, we refer the reader to the definitive text by Garey and Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Here we will provide just a shallow background and establish the terminology used in this paper.

Computational complexity has only a little to do with computers, and a great deal to do with decision problems and algorithms. A **decision problem** consists of a collection of **instances,** and a yes/no (or true/false) **question** regarding these instances. A typical decision problem is:

Instance: A finite, simple, undirected graph G.
Question: Does G have a Hamiltonian cycle?

The **size** of this problem might be regarded as the number of edges of the graph, or the number of vertices, or possibly the sum of these.

An **algorithm** is a process for answering the question, given any one of the instances. That is, the same algorithm should correctly solve the question for any of the possible instances. We say an algorithm is **efficient** or **polynomial-time** (sometimes shortened to **polytime**) if it runs in polynomial time with respect to the size of the instance. This characterization is mathematically useful because combinations of efficient algorithms are still efficient. It is also practically useful when the coefficients and exponents of the polynomial are not too large, because such algorithms will fairly quickly produce an answer (or prove that none exists).

There are two aspects to solving any decision problem: Producing an answer (**solving** the problem), and **verifying** an answer. Sometimes one of these tasks may be significantly easier than the other. For example, it may be difficult to find a Hamiltonian cycle in a graph, but once we are given a cycle it is straightforward to verify whether or not it really is Hamiltonian. Decision problems which can be solved with an efficient algorithm belong to the class **P** (for **polynomial time**). Problems for which candidate solutions can be verified with an efficient algorithm belong to the class **NP** (for **nondeterministic**

polynomial time). "Nondeterministic" refers to the fact that although we may be able to verify a "yes" answer to the question, verifying a "no" answer basically involves solving the problem itself. Thus the algorithm cannot provide an answer (i.e., is not deterministic), but can decide whether an answer works. So, although every decision problem which can be solved in polynomial time can also be verified in polynomial time, the converse may not be true. Said another way, P is a subset of NP, but it is not known whether the two classes are equal.

Most researchers believe that P ≠ NP because of the inherent asymmetry between verifying solutions and producing them. For example, verifying the claim that a particular graph has no Hamiltonian cycle often appears to require searching through every possibility unless the graph has some special property (or properties) which guarantee or exclude the existence of a Hamiltonian cycle (or otherwise simplify our search), and we can check in polynomial time whether or not a graph has that property. Thus, verifying a negative answer is tantamount to solving the problem in the first place (exhaustive searches can take a long time). In contrast, verifying whether a given set of edges really is a Hamiltonian cycle in the graph is easy.

One important area of research involves determining whether a problem is in P or NP. Proving a problem is in P is fairly straightforward: Either produce a polynomial time algorithm for solving it, or else find a **polynomial time transformation** of it into another problem which is already known to be in P. Because adding and multiplying polynomials results in another polynomial, such transformations do not destroy the polynomial time complexity of the algorithm. When there are polynomial time transformations from each problem to the other, we say the problems are (**polynomial-time**) **equivalent** to each other.

Proving a problem is not in P is usually not as simple. After all, the inability to find an efficient algorithm does not imply no such algorithm exists. However, there is a third class of decision problems which play a very important role in this paper, namely the class of **NPC** (**NP-complete**) problems. These problems are in a sense "the hardest problems in NP," because every problem in NP can be transformed (in polynomial time) to one of them. More generally, a problem which is at least as hard as every problem in NP (but not necessarily in NP) is called **NP-hard**. Thus, the NP-complete problems are precisely those problems which are both in NP and NP-hard.

In order to show a problem is NP-complete, it suffices to find a polynomial time transformation from an existing NP-complete problem to it. Of course, this requires an NP-complete problem to begin with, and it is not trivial to show that one exists. However, Cook (who introduced the notion of NP-completeness) proved that the satisfiability problem of propositional logic is in fact, NP-complete. Briefly, this problem concerns elements from logic:

- **literals**, which are Boolean variables or their negations
- **clauses**, which are disjunctions of literals (that is, a clause is true if and only if at least one literal in it is true)
- **formulas**, which are conjunctions and disjunctions of clauses

We collectively refer to these elements as **Boolean expressions**. A Boolean expression is **satisfiable** if there is an assignment of true and false values to its variable(s) such that the expression is true.

A Boolean formula is in **conjuctive normal form** when it is purely a conjunction of clauses (that is, the formula is true if and only if all of its clauses are true). Then the satisfiability problem (SAT) is:

Instance: A Boolean formula in conjunctive normal form.
Question: Is it satisfiable?

A proof that SAT is NP-complete can be found in Garey and Johnson.

Since the introduction of NP-completeness, literally hundreds of problems have been proven NP-complete by transforming known NP-complete problems into them. But, for every problem whose computational complexity is determined, it seems two (or more) problems whose computational complexity is unknown are found.

The two problems with which this paper is mostly concerned are the Even Cycle Problem for directed graphs (EVCY) and the square L-matrix problem (LMAT). These two problems are equivalent, but their complexity remains unknown.

Instance: A finite simple directed graph G.
Question: Does G have an even cycle?

Instance: A square real matrix A.
Question: Is A an L-matrix?

Appendix B: Directed Graph Theory

Abstractly, a **graph** is a set V of **vertices** (or nodes, or points) together with a set E of **edges** (or arcs, or lines), where each element of E is an unordered pair (u, v) of elements in V. The vertices u and v are the **endpoints** of the edge (u, v). A **directed graph** (or **digraph**) is superficially similar to an undirected one; the only change to the definition above is that now edges have a direction to them; each element of the edge set E of a directed graph is an *ordered* pair of elements in V. However, this small difference in their definitions produces large (and sometimes surprising) differences in their structures.

Often, and in this paper in particular, we restrict V and E to be finite sets. We also require the graphs to be **simple**, which means we disallow **loops** (edges of the form (v, v) for some vertex v) and **multiple edges** from one vertex to another.

For directed graphs we do not count edges (u, v) and (v, u) as multiple edges because they have opposite directions. In fact, we say the edge (u, v) is **incident from** u, and **incident to** v. Sometimes we then also say u **dominates** v. The vertices incident to (respectively, from) v are called the **in-neighbors** (respectively, **out-neighbors**) of v. The **indegree** (resp., **outdegree**) of a vertex is the number of its in-neighbors (resp., out-neighbors). When there is no confusion, we may refer to the out-neighbors of v as simply its **neighbors**.

A **subgraph** [**subdigraph** for directed graphs] has as its vertex set some subset of the vertex set of the original graph, and every edge in the subgraph is an edge in the original. A maximal subgraph is a subgraph with the maximum possible number of edges (every edge which is in the original and has both endpoints in the vertex set of the subgraph).

The removal of a vertex from a graph results in the subgraph formed by removing the vertex and all the edges incident to it. Removing an edge does not remove any vertices. An edge (u, v) may be **subdivided** by adding one or more new vertices $w_1, \ldots w_n$ to the vertex set, removing the edge (u, v), and in its place adding the edges $(u, w_1), \ldots (w_{n-1}, w_n), (w_n, v)$.

We can also speak of longer sequences of vertices joined by edges. A sequence $x_1\ e_1\ x_2\ e_2 \ldots x_n\ e_n\ x_{n+1}$ is called a **walk** (of length n) whenever all the x_i are vertices and each edge $e_i = (x_i, x_{i+1})$ joins the two vertices immediately

surrounding it in the sequence. For a simple graph, it is enough to list just the vertex sequence, since the edges are then uniquely determined.

A **trail** is a walk in which no edge is repeated; a **path** is a walk in which no vertex (and hence no edge) is repeated. A walk or trail is **closed** if the first and last vertices are the same ($x_{n+1} = x_1$). A **cycle** of length n is a closed path $x_1\ e_1\ x_2 \ldots x_n\ e_n\ x_1$, which we will sometimes write as $(x_1, x_2, \ldots x_n)$ to emphasize that it is a cycle and not merely a path. When we speak of the parity of any of these, we mean the parity of the number of edges in the walk, trail, path or cycle in question. A digraph with no cycles at all is **acyclic**.

Another useful concept in analyzing the structure of graphs is that of connectedness. Loosely speaking, the more connected a graph is, the easier it is to travel along edges in the graph from one vertex to another. A directed graph is (**strongly**) **connected** (or just **strong**) if, for any two vertices x and y of the digraph, there is a path from x to y, and a path from y to x. More generally, a digraph is (**strongly**) **k-connected** if the removal of any k-1 or fewer vertices results in a digraph which is still connected. If the underlying undirected graph (formed by disregarding the orientation of each edge) is connected, then the digraph is **weakly connected** (or just **weak**). We make no use of this concept in this paper, but mention in passing that a digraph which is not even weakly connected is said to be **disconnected**, and a weakly connected acyclic directed graph is called a directed tree.

Strongly connected digraphs are of particular interest in directed graph theory. Often it suffices to prove a theorem only for strong digraphs, and then the other cases follow. Each maximal strongly connected subdigraph is called a (**strong**) **component** of the digraph. It is not difficult to prove that every vertex of a digraph lies in a unique strong component, and in fact these components can be found in polynomial time.

Each graph has an **adjacency matrix** associated with it. The adjacency matrix has rows and columns indexed by the vertex set of the graph. The element of this matrix in row u, column v is 1 whenever (u, v) is an edge in the graph, and 0 otherwise. Consequently, the adjacency matrix of an undirected graph is symmetric, but this need not hold for directed graphs.

The adjacency matrix provides an alternative form for representing a digraph from the usual vertex and edge sets. Because this representation can be

obtained from the other in polynomial time, the two are equivalent from our algorithmic point of view.

The relationships between digraphs and matrices are deep and complex. For example, directed graph theory can be used to calculate the eigenvalues and inverse (when it exists) of a square matrix, and is especially effective when the associated digraph has few edges, which translates into the matrix being **sparse** (i.e., the matrix has many zeros). [Harary, p. 205]

The relationship between directed graphs and L- and S-matrices is the focus of chapter two. For an excellent introduction to undirected and directed graph theory, the interested reader is directed (no pun intended) to Graph Theory with Applications, by Bondy and Murty.

Appendix C: Relatives of the Even Cycle Problem

References are provided when possible, as well as the name used in the literature to describe the problem. We first list those problems whose complexity is known, and conclude the section with those problems which are equivalent to the Even Cycle Problem, but whose complexity remains unknown. See Appendix A for details on the meaning of "equivalent" and the format of the decision problems presented here.

When convenient, several problems with similar wordings are grouped together by putting the alternate wordings in square brackets. Unless explicitly stated otherwise, all graphs and digraphs are finite and simple, and all matrices are real.

Polynomial-Time (P) Problems

Odd Cycle [Walk, Trail]
Instance: A directed graph G.
Question: Does G have an odd cycle [walk, trail] ?
(Klee, Ladner, and Manber, 1984)

Local Odd [Even] Walk
Instance: A directed graph G and one of its vertices v.
Question: Does G have an odd [even] walk through v?
(Klee, Ladner, and Manber, 1984)

S[*]-Matrix
Instance: A matrix A.
Question: Is A an S[*]-Matrix?
(Klee, 1987)

Undirected Even [Odd] Cycle
Instance: An undirected graph G.
Question: Does G have an even [odd] cycle?
(LaPaugh and Papadimitriou, 1984)

Planar Even Cycle

Instance: A planar, directed graph G.
Question: Does G have an even cycle?
(Thomassen, 1993; Galluccio and Loebl, 1994)

Even Cycle on Fixed Surface

Instance: An arbitrary fixed surface and any direceted graph G embedded on that surface.
Question: Does G have an even cycle?
(Galluccio and Loebl, 1996)

Even Cycles in H-Free Graphs

Instance: A $K_{3,3}$ -free or K_5 -free directed graph G.
Question: Does G have an even cycle?
(Galluccio and Loebl, 1995)

Planar Perfect Matching

Instance: A planar, undirected graph G.
Question: Does G have a perfect matching?
(Kasteleyn, 1967)

NP-Complete (NPC) Problems

Local Odd [Even] Cycle (LOCODD[EV]CY)

Instance: A directed graph G and one of its vertices v.
Question: Does G have an odd [even] cycle through v?
(Klee, Ladner, and Manber, 1984)

General L-Matrix

Instance: An mxn matrix A.
Question: Is A an L-matrix?
(Klee, Ladner, and Manber, 1984)

Restricted L-Matrix

Instance: An $(n+\lfloor n^{1/k} \rfloor)$xn matrix A.
Question: Is A an L-matrix?
(Klee, Ladner, and Manber, 1984)

NP-Hard Problems

Permanent Equals Determinant

Instance: A square matrix A.
Question: Does perm(A) = det (A)?
(Vazirani and Yannakakis, 1989)

Nonsingular Interval Matrix

Instance: An interval of matrices $[A + cB, A - cB]$, $0 \leq c \leq 1$.
Question: Is every matrix in the interval nonsingular?
(Poljak and Rohn, 1993)

The Even Cycle Problem and Equivalent Ones

Even Cycle (EVCY)

Instance: A directed graph G.
Question: Does G have an even cycle?

Even Trail

Instance: A directed graph G.
Question: Does G have an even trail?

Square L-matrix Problem (LMAT)

Instance: A square matrix A.
Question: Is A an L-matrix?
(Bassett, Maybee, and Quirk, 1968)

Even Digraph

Instance: A directed graph G.
Question: Is G even?
(Seymour and Thomassen, 1987)

Balanced Labelling

Instance: A directed graph G.
Question: Does G admit a balanced labelling?
(Klee, n. d.)

Perfect Matching

Instance: An undirected, bipartite graph G.
Question: Does G have a perfect matching?
(Vazirani and Yannakakis, 1989)

Restricted Version of Permanent Equals Determinant

Instance: A square matrix A with non-negative entries.
Question: Does perm(A) = det (A)?
(Vazirani and Yannakakis, 1989)

Polya's Problem

Instance: A square 0-1 matrix A.
Question: Can some entries be changed from +1 to –1 to obtain a new matrix B for which det(B) = perm(A)?
(Vazirani and Yannakakis, 1989)

Bibliography

Alon, Noda and N. Linial, Cycles of Length 0 Modulo k in Directed Graphs. *J. Combinatorial Theory B*, 47 (1989) 114-119.

Bassett, Lowell, John Maybee, James Quirk, Qualitative Economics and the Scope of the Correspondence Principle. *Econometrica*, 36 (1968) 544-563.

Bondy, John A. and U. S. R. Murty, Graph Theory with Applications. MacMillan, London, 1976.

Brualdi, Richard A. and Bryan L. Shader, Matrices of Sign-Solvable Linear Systems. Cambridge University Press, 1995.

Bermond, J. C. and Carsten Thomassen, Cycles in Digraphs — A Survey. *J. Graph Theory*, 5 (1981) 1-43.

Chung, F. R. K., Wayne Goddard, and Daniel J. Kleitman, Even Cycles in Directed Graphs. *SIAM J. Discrete Math.*, 7 (1994) 474-483.

Fortune, Steven, John Hopcroft, and James Wyllie, The Directed Subgraph Homeomorphism Problem. *Theoretical Computer Science*, 10 (1980) 111-121.

Friedland, Shmuel, Every 7-Regular Digraph Contains an Even Cycle. *J. Combinatorial Theory B*, 46 (1989) 249-252.

Garey, Michael R. and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., San Francisco, 1979.

Gallucio, Anna and Marin Loebl, Even Cycles and H-Homeomorphisms. To appear, 1996.

Gallucio, Anna and Marin Loebl, Even Directed Cycles in H-Free Digraphs. *J. Algorithms*, to appear, 1995.

Gallucio, Anna and Marin Loebl, Even/Odd Dipaths in Planar Digraphs. *Optimization Methods and Software*, 3 (1994) 225-236.

Harary, Frank, Graph Theory. Addison-Wesley Publishing Co., Reading, Massachusetts, 1969.

Karmarkar, S. B., An Algorithm for Finding a Circuit of Even Length in a Directed Graph. *International J. Systems Sci.*, 15 (1984) 1197-1201.

Kasteleyn, P. W., Graph Theory and Crystal Physics. Graph Theory and Theoretical Physics, Frank Harary, ed. Academic Press, New York, 1967, 43-110.

Klee, Victor, Recursive Structure of S-Matrices and an $O(m^2)$ Algorithm for Recognizing Strong Sign-Solvability. *Linear Algebra and its Applications*, 96 (1987) 233-247.

Klee, Victor, Sign-patterns and Stability. Applications of Combinatorics and Graph Theory to the Biological and Social Sciences, F. Roberts, ed. IMA Volumes in Mathematics and Its Applications, Springer, New York, 17 (1989) 203-219.

Klee, Victor, The Even Cycle Mystery, the L-Matrix Problem, and Their Relatives. Unpublished manuscript, n.d.

Klee, Victor and Richard Ladner, Qualitative Matrices: Strong Sign-solvability and Weak Satisfiability. Computer-Assisted Analysis and Model Simplification, H. Greenberg and J. Maybee, eds. Academic Press, New York, 1981, 293-320.

Klee, Victor, Richard Ladner, Rachel Manber, Signsolvability Revisited. *Linear Algebra and its Applications*, 59 (1984) 131-157.

Klee, Victor, Balder Von Hohenbalken, Ted Lewis, On the Recognition of S-Systems. *Linear Algebra and its Applications*, 192 (1993) 187-204.

Knuth, Donald E., A Permanent Inequality. *American Math. Monthly*, (1981) 731-740.

Knuth, Donald E., Overlapping Pfaffians. *Electronic Journal of Combinatorics*, Foata Festschrift, 3 (1995) 1-13. Available at http://ejc.math.gatech.edu/Journal/Volume_3/foatatoc.html

Koh, Khee Meng, Even Circuits in Directed Graphs and Lovasz's Conjecture. *Bull. Malaysian Math. Soc.*, 7 (1976) 47-52.

Kuˇcera, Ludˇek, Combinatorial Algorithms. Adam Hilger, Philadelphia, 1990.

LaPaugh, Andrea S. and Christos H. Papadimitriou, The Even-Path Problem for Graphs and Digraphs. *Networks*, 14 (1984) 507-513.

Lint, J. H. van and R. M. Wilson, A Course in Combinatorics. Cambridge University Press, 1992.

Little, C. H. C., An Extension of Kasteleyn's Method of Enumerating the 1-factors of Planar Graphs. Combinatorial Mathematics, Proceedings 2nd Australian Conference, D. Holton, ed. Lecture Notes in Mathematics 403, Springer, Berlin, 1974, 63-72.

Manber, Rachel, Graph-Theoretical Approach to Qualitative Solvability of Linear Systems. *Linear Algebra and its Applications*, 48 (1982) 457-470.

Maybee, John and James Quirk, Qualitative Problems in Matrix Theory. *SIAM Review*, 11 (1969) 30-51.

Poljak, S. and J. Rohn, Checking Robust Nonsingularity is NP-Hard. *Math. Control Signals Systems*, 6 (1993) 1-9.

Polya, G., Aufgabe 424. *Arch. Math. Phys.*, 30 (1913) 271.

Quirk, James and R. Ruppert, Qualitative economics and the stability of equilibrium. *Rev. Economic Studies*, 32 (1965) 311-326.

Robinson, D. F. and L. R. Foulds, Digraphs: Theory and Techniques. Gordon and Breach Science Publishers, New York, 1960.

Rohn, Jiri, Systems of Linear Interval Equations. *Linear Algebra and its Applications*, 126 (1989) 39-78.

Rohn, Jiri, Interval Matrices: Singularity and Real Eigenvalues. *SIAM J. Matrix Anal. Appl.*, 14 (1993) 82-91.

Seymour, P. and Carsten Thomassen, Characterization of Even Directed Graphs. *J. Combinatorial Theory B*, 42 (1987) 36-45.

Tarjan, Robert, Depth-First Search and Linear Graph Algorithms. *SIAM J. Computing*, 1 (1972) 146-160.

Thomassen, Carsten, Even Cycles in Directed Graphs. *European J. Combinatorics*, 6 (1985) 85-89.

Thomassen, Carsten, Sign-Nonsignular Matrices and Even Cycles in Directed Graphs. *Linear Algebra and its Applications*, 75 (1986) 27-41.

Thomassen, Carsten, The Even Cycle Problem for Planar Digraphs. *J. Algorithms*, 15 (1993) 61-75.

Thomassen, Carsten, The Even Cycle Problem for Directed Graphs. *J. Amer. Math. Soc.*, 5 (1992) 217-229.

Valiant, L. G., The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8 (1979) 189-201.

Vazirani, V. J. and Mihalis Yannakakis, Pfaffian Orientations, 0-1 Permanents, and Even Cycles in Directed Graphs. *Discrete Applied Math*, 25 (1989) 179-190.