CMU-CS-80-102

Two Papers on Graph Embedding Problems

James B. Saxe

January 1980

CMU-CS-80-102

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-80-102

Two Papers on Graph Embedding Problems

James B. Saxe

January 1980

Abstract

This report contains two independent papers on problems concerned with *graph embedding—i.e.*, assignment of the vertices of a graph to points in a metric space subject to specified constraints. The first paper in this report, "Embeddability of weighted graphs in k-space is strongly NP-hard," examines the problem of assigning the vertices of a *weighted* graph to points in a k-dimensional Euclidean space subject to the constraint that any two vertices connected by an edge must be assigned to points whose distance is the weight of that edge. We prove (by reduction from 3-satisfiability) that it is NP-hard to determine whether such an assignment exists, even when k=1 and the edge weights are restricted to take on the values 1 and 2. The same reduction used in this proof forms the basis of proofs of the NP-completeness of several variants of the original problem. The second paper, "Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time," deals with the problem of *bandwidth minimization*, in which we are given a graph, G, and a positive integer, k, and asked whether it is possible to assign the vertices of G to distinct integers subject to the constraint that no edge of G may have its endpoints mapped to integers which differ by more than k. Although the general problem has been proven (by C. H. Papadimitriou) to be NP-complete, we show that it can be solved in polynomial time for any fixed value of k. As in the first paper, the methods used to achieve the principal result are extended to a number of related problems.

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370.

Embeddability of Weighted Graphs in k-Space is Strongly NP-Hard

James B. Saxe
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this paper we investigate the complexity of *embedding* edge-weighted graphs into Euclidean spaces: Given an (incomplete) edge-weighted graph, G, can the vertices of G be mapped to points in Euclidean k-space in such a way that any two vertices connected by an edge are mapped to points whose distance is equal to the weight of the edge? We prove (by reduction from 3-satisfiability) that this problem is strongly NP-hard. Indeed, it is NP-complete even when k=1 and the edge weights are restricted to take on the values 1 and 2. We also investigate the related problems of *approximate embeddability* (in which G is accepted if its vertices can be embedded in k-space so that the distances between connected vertices match the corresponding edge weights within some small tolerance but G is rejected if there is no mapping which meets some other, larger tolerance) and the problem of *ambiguous embedding* (in which we are given both a graph, G, and an embedding for G and asked whether a second embedding exists which is not congruent to the first). We show that these related problems are just as hard as the ordinary embeddability problem.

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370.

1. Introduction

In many applications of distributed sensor networks[1] there arises the problem of determining the locations of sensors from incomplete (and possibly errorful) information about their distances from each other and from fixed landmarks. This prompts us to ask the following geometric questions:

- Given an incompletely specified distance matrix for a set of points in k-space,[2] when is the complete distance matrix uniquely determined?

- Assuming the distance matrix to be uniquely determined, what is the computational complexity of actually finding the unspecified distances?

In this paper we consider the closely related problem of embeddability:

- Given a (purported) incompletely specified distance matrix for a set of points in k-space, determine whether there can actually exist a set of points satisfying that matrix.

In Section 2 we introduce definitions that will allow us to phrase several forms of the embeddability problem in terms of edge-weighted graphs. In Section 3, we give a simple proof that a 1-dimensional version of the embeddability problem is NP-complete. In Section 4, we show the more difficult and surprising result that this same 1-dimensional problem is strongly NP-complete in the sense of Garey and Johnson [1979] and extend this result to higher dimensions. In Section 5 we address some naturally-arising questions concerning the suitability of the Turing Machine model for a problem that inherently involves real numbers, and show that the proofs used in Section 4 have relevance to an "approximate embeddability" problem on the reals. In Section 6 we discuss versions of the problem in which one way to complete an incompletely specified distance matrix is known and it is desired to determine whether a second solution exists. We show that these versions are no

[1]See, for example, *Distributed Sensor Nets* [1978].

[2]For practical purposes the most interesting cases are k=2 and k=3.

easier than corresponding versions studied earlier in the paper. Finally, the contributions of the paper are summarized in Section 7.

2. Fundamental Concepts

We begin by introducing the concepts of weighted graph and embedding:

Definitions:

A weighted graph, $G = \langle V,E,W \rangle$, is an ordered triple such that each element of E is an unordered pair of distinct elements of V and W is a function mapping E into $[0,\infty)$. The elements of V are called the vertices of G. The elements of E are called the edges of G. For each edge, e, of G, the real number $W(e)$ is called the weight of e in G (or simply the weight of e).

Definitions:

Let $G = \langle V,E,W \rangle$ be a weighted graph, and let k be a positive integer. Then an embedding of G in k-space is a function, f, mapping V into the k-dimensional Euclidean space, $\mathbb{R}^k$, such that, for each edge, $e = \{v,w\}$, of G, $|f(v)-f(w)| = W(e)$. G is said to be embeddable in k-space, or k-embeddable, iff there exists an embedding of G in k-space.

For any positive integer, k, the problem of k-embeddability may now be stated as follows:

Problem (k-Embeddability):

Given an arbitrary weighted graph, G, determine whether G is k-embeddable.

In Sections 3 and 4 we will wish to restrict the class of weighted graphs under consideration, so that the notion of NP-completeness (which is defined in terms of Turing machines) will make sense in relation to Embeddability. We therefore introduce the following definition.

Definition:

Let S be any subset of $[0,\infty)$. Then, an S-weighted graph is a weighted graph, G, such that the weight of each edge of G is an element of S. We will generally refer to $\mathbb{Z}^+$-weighted graphs as integer-weighted graphs.

In Section 5 we will return to the question of graphs with real edge weights.

3. The Weak NP-completeness of 1-Embeddability

In this section, we demonstrate the weak NP-completeness of the problem of 1-Embeddability of integer-weighted graphs. To do this, we first show constructively that 1-Embeddability is in NP. We then use a reduction from Partition[3] to show completeness.

Theorem 3.1:

1-Embeddability of integer-weighted graphs is in NP.

Proof:

To check the 1-embeddability of any integer-weighted graph, a NDTM need only

1. Partition the graph into disjoint connected subgraphs,
2. Guess the direction of each edge of the graph, and
3. Check the consistency of each disjoint connected subgraph.

These operations can clearly be carried out in (nondeterministic) polynomial time. □

Theorem 3.2:

1-Embeddability of integer-weighted graphs is NP-complete.

Proof:[4]

We will show the NP-completeness of 1-Embeddability by reduction from Partition. Let $S = \{a_1, a_2, \ldots, a_n\}$ be a multiset of positive integers. In polynomial time we may construct from S a description of the cyclic graph

[3]The Partition problem calls for partitioning a (multi-)set of integers into two subsets with equal sums, and is known to be NP-complete; see Garey and Johnson [1979].

[4]The construction used in this theorem and that used in the proof of Lemma 4.4 were independently developed by Yemini [1978], who used them to show the (weak) NP-completeness of 2-Embeddability of integer-weighted graphs.

$G = \langle V,E,W\rangle$ whose edge weights are the a_i, that is

$$V = \{v_0, \ldots, v_{n-1}\},$$
$$E = \{\{v_i, v_{(i+1 \bmod n)}\} \mid 0 \le i < n\}, \text{ and}$$
$$W = \{(\{v_i, v_{(i+1 \bmod n)}\}, a_i) \mid 0 \le i < n\}.$$

If f is an embedding of G in the line, then the multisets

$$S_1 = \{a_i \mid f(v_i) < f(v_{(i+1 \bmod n)})\} \text{ and}$$
$$S_2 = \{a_i \mid f(v_i) > f(v_{(i+1 \bmod n)})\}$$

constitute a partition of S into two pieces whose sums are equal. Similarly, any such partition of S yields a 1-Embedding of G. □

4. The Strong NP-completeness of 1-Embeddability

We now come to our key theorem, which asserts that the problem of determining whether an integer-weighted graph is embeddable in the line remains NP-complete even if the edge weights are restricted to be no greater than four.

Theorem 4.1

1-Embeddability of $\{1,2,3,4\}$-weighted graphs is NP-complete.

Proof:

Our proof consists of a reduction from 3-Satisfiability (which was shown to be NP-complete by Cook [1971]) to 1-Embeddability of $\{1,2,3,4\}$-weighted graphs. Let E be any Boolean expression in conjunctive normal form with three literals in each clause. Our goal will be to construct a $\{1,2,3,4\}$-weighted graph, G, which is embeddable iff E is satisfiable. We let n be the number of variables occurring in E and m be the number of clauses in E. Throughout this proof, we will use the convention that the variables of E will be indexed by "i" (which will therefore range from 1 through n), the clauses of E will be indexed by "j" (ranging from 1 through m), and the literals within each clause will be indexed by "k" (ranging from 1 through 3). Thus E has the form

$$E = \prod_{1 \le j \le m} C_j,$$

where each clause, C_j, has the form

$$C_j = \sum_{1 \le k \le 3} L_{j,k},$$

and each literal, $L_{j,k}$, has the form

$$L_{j,k} = X_i \quad \text{or} \quad L_{j,k} = \bar{X}_i$$

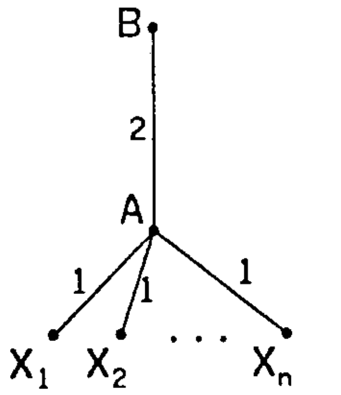
for some i, $1 \leq i \leq n$. We will also use throughout the proof the convention that "f" represents a hypothetical 1-embedding of G (or of the part of G we have constructed so far).

To construct G, we will use the "building blocks" shown in Figure 4.1. We begin with the subgraph shown in Figure 4.1(a). We assume without loss of generality that $f(A) = 0$ and $f(B) = 2$. This assumption constrains f to assign each of the X_i (which we identify with the variables of E) to 1 or -1 (which we identify with the Boolean values TRUE and FALSE, respectively). Note that each possible mapping of the X_i into $\{1,-1\}$ corresponds to some assignment of truth values to the X_i. In the remaining steps of the construction, we will add edges which have precisely the effect of constraining f to map the X_i to $\{1,-1\}$ in such a way that the corresponding assignment of the X_i satisfies E.

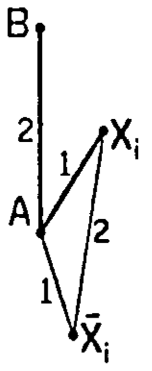
The next step in our construction is to augment G by adding the edges shown in Figure 4.1(b) for each i, $1 \leq i \leq n$. The heavy lines in that figure represent already-existing edges. We now have vertices $\bar{X}_i$ such that for each variable, X_i, f maps X_i to 1 (TRUE) iff it maps $\bar{X}_i$ to -1 (FALSE), and vice-versa. The possible mappings from $\{X_i\} \cup \{\bar{X}_i\}$ to $\{1,-1\}$ under f now correspond precisely to the possible (consistent) truth assignments of the X_i and $\bar{X}_i$, but still without regard to whether those assignments satisfy E.

For the final step of our construction, we add the edges indicated in Figure 4.1(c) for each j, $1 \leq j \leq m$. The vertices $L_{j,k}$ are identified with the X_i and $\bar{X}_i$ precisely as the corresponding literals, $L_{j,k}$, are formally identical with the X_i and $\bar{X}_i$. Once again, the heavy lines indicate edges which were present at earlier stages of the construction. Careful study of the graph in Figure 4.1(c) will reveal that it is impossible to embed it in the line in such a way that A is sent to 0, B is sent to 2, and all three of the $L_{j,k}$ are sent to -1 (FALSE), but if one or more of the $L_{j,k}$ are to be sent to 1 (TRUE), then an embedding is possible (in fact, exactly one such embedding is possible). Thus, for each j, $1 \leq j \leq m$, the effect of the edges in Figure 4.1(c) is precisely to constrain f to map the X_i to $\{1,-1\}$ in such a way that the corresponding truth assignment for the X_i satisfies clause C_j.

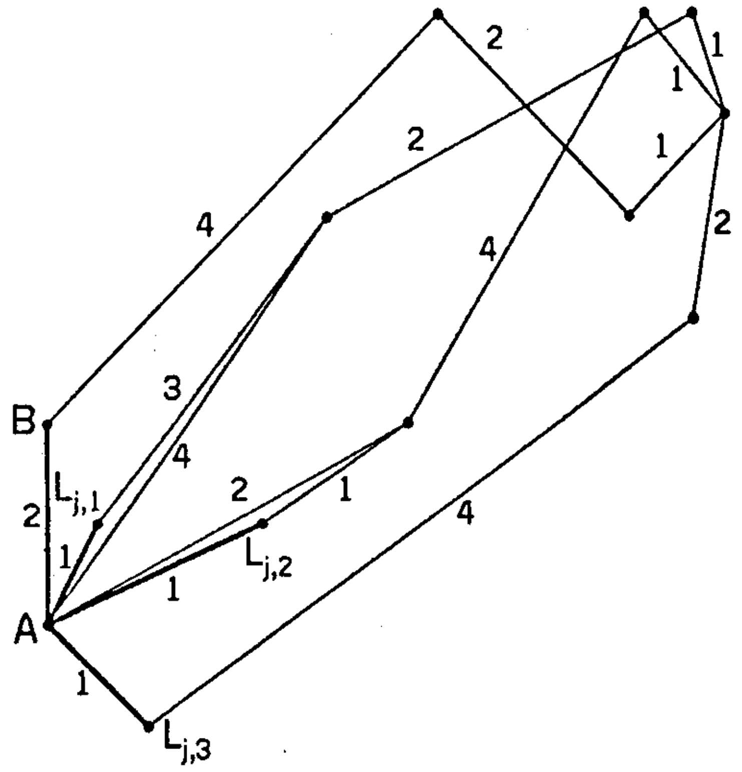
The effect of all the edges of G is therefore to constrain f to map the X_i to $\{1,-1\}$ in such a way that the corresponding assignment of truth values to the X_i satisfies E. If there is no such assignment then G is not 1-embeddable. If



(a) Implementation of variables.



(b) Implementation of a negative literal.



(c) Implementation of a disjunctive clause.

Figure 4.1 Building blocks for transforming an expression in 3-CNF to a graph.

there are any assignments satisfying E, then for each such assignment G can be (uniquely) 1-embedded by a function sending A to 0 and B to 2 and mapping the X_i to $\{1,-1\}$ in accordance with that assignment. Finally, it is clear that the preceding construction can be carried out in polynomial time. This completes the proof. □

For future reference, we note that the construction used in the preceding proof is such that the 1-embeddings of G are in one-to-one correspondence (up to translation and reflection) with the truth assignments that satisfy E. We note also that the preceding theorem immediately yields the following result:

Corollary 4.2:

1-Embeddability of integer-weighted graphs is strongly NP-complete.

Proof:[5]

It suffices to note that translation of a sequence of numbers in {1,2,3,4} from binary to unary can be accomplished in linear time and causes only a constant factor increase in the length of the input. □

We may also immediately derive:

Corollary 4.3:

1-Embeddability of {1,2}-weighted graphs is NP-complete.

Proof:

Consider the graphs shown in Figure 4.2. By replacing edges of weights 3 and 4 with configurations T_3 and T_4, respectively, we can reduce any {1,2,3,4}-weighted graph, G, to a {1,2}-weighted graph, H, that is 1-embeddable iff G is 1-embeddable. □

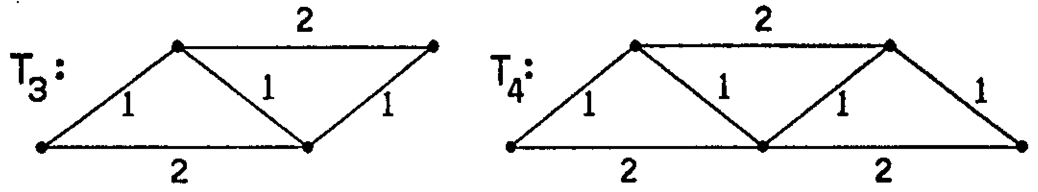


Figure 4.2. Building long "edges" from short edges.

In fact, for any positive integer, k, the graph H so constructed will be k-embeddable

[5]Another NP-complete problem involving a form of graph embedding is the Bandwidth Minimization Problem (see Papadimitriou [1976]). In Appendix I we exhibit a reduction from Bandwidth Minimization to Embeddability. That reduction suffices to show the strong NP-completeness of 1-Embeddability of integer-weighted graphs, although it is somewhat less economical than the construction given in Theorem 4.1. Also, it cannot be used as a basis for deriving the results presented in Sections 5 and 6.

iff G is k-embeddable. We may use this fact to prove our next lemma.

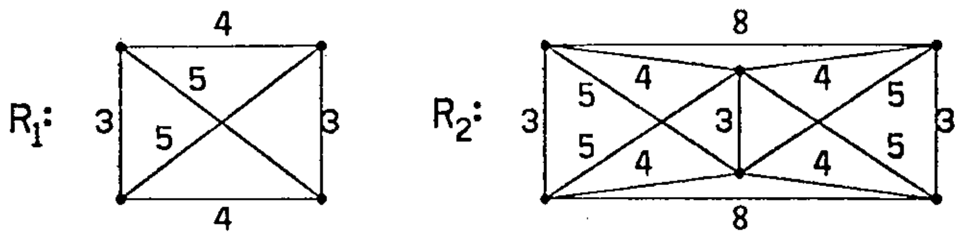


Figure 4.3. Gadgets for adding a dimension.

Lemma 4.4:

For every positive integer, k, k-Embeddability of {1,2}-weighted graphs is NP-hard.

Proof:

Consider the graphs shown in Figure 4.3. Given any {1,2}-weighted graph, G, each edge of G having weight 1 may be replaced by the R_1 and each edge of weight 2 by R_2, yielding a graph, H, which, for any positive integer, k, is embeddable in (k+1)-space iff G is embeddable in k-space. By the methods of Theorem 4.3, H may be transformed into a {1,2}-weighted graph, J, that is embeddable in precisely those spaces in which H is embeddable. The transformation from G to J involves only a constant factor increase in the length of a specification of the graph and can clearly be accomplished in polynomial time. It follows by mathematical induction that, for any positive integer, k, 1-Embeddability is polynomial-time reducible to k-Embeddability for {1,2}-weighted graphs. □

Once again, we note that the (k+1)-embeddings of J will be in one-to-one correspondence (up to translation, rotation and reflection) with the k-embeddings of G. Finally, Theorem 4.4 gives us the following result.

Corollary 4.5:

Let k be any positive integer. Then k-Embeddability of integer-weighted graphs is strongly NP-hard.

Proof:

This result follows from Lemma 4.4 and the same reasoning used in the proof of

Corollary 4.2. □

5. Graphs with Real-Valued Edge Weights

We will now discuss the applicability of NP-completeness to problems whose inputs are real numbers in general, and to embedding problems in particular. A number of reasons for doubting the relevance of the Turing Machine model seem naturally to present themselves.

- NP-completeness is defined for language recognition problems on Turing Machines, which inherently can deal only with integers and not with arbitrary reals.

- Given a "random" embedding of an unweighted graph into a Euclidean space, any two of the edge weights induced by the embedding will be incommensurable with probability 1. Moreover, if the graph is overconstrained and the dimension of the space is at least two, then rounding the induced edge-weights to multiples of some small distance will almost always produce a weighted graph that is not embeddable in the space.

In order to deal with these issues, we introduce the notion of approximate embeddings.

Definitions.

Let G be a weighted graph and ϵ be a non-negative real number. Then an ϵ-approximate k-embedding of G is a function, f, that maps the vertices of G into Euclidean k-space such that for every edge, {u,v}, of G, $1-\epsilon < |f(u)-f(v)|/W(\{u,v\}) < 1+\epsilon$. If such an embedding exists, then G is said to be ϵ-approximately k-embeddable.

Given a positive integer, k, and two reals, ϵ_1 and ϵ_2, such that $0 \leq \epsilon_1 \leq \epsilon_2$, we may now define the following more "robust" embeddability problem:

Problem (ϵ_1,ϵ_2-Approximate k-Embeddability):
Given a weighted graph, G, assert correctly either (1) that G is ϵ_2-approximately k-embeddable (this is called accepting G) or (2) that G is not ϵ_1-approximately k-embeddable (this is called rejecting G).

Note that if the least ϵ for which G is ϵ-approximately k-Embeddable lies in the interval $(\epsilon_1,\epsilon_2]$, then it is permissible either to accept or to reject G. In this problem definition, we have attempted to capture, without introducing inordinately many complexities of detail, the essential problem of embedding as it would apply to real computers given inexact data.

We now wish to investigate the computational complexity of ϵ_1,ϵ_2-Approximate Embeddability problems. Is it possible, for example, to solve all such problems where ϵ_1 is strictly less than ϵ_2 in time polynomial in the size of a specification of G (where the degree of the polynomial, or even just the "constant" factor, depends on $(\epsilon_2-\epsilon_1)^{-1}$)?

It turns out that such polynomial solutions are not possible in the general case (assuming that $P \neq NP$). In particular, we have the following result.

<u>Theorem</u> <u>5.1</u>:

Let ϵ_1 and ϵ_2 be real numbers such that $0 \leq \epsilon_1 \leq \epsilon_2 < 1/8$. Then ϵ_1,ϵ_2-Approximate 1-Embeddability of integer-weighted graphs is NP-complete.

<u>Proof</u>:

We note that the embeddability properties of the graphs used in the proof of Theorem 4.1 depend only on cycles of length no greater than 16 having edges whose lengths are multiples of 1. It follows from this that, for any $\epsilon < 1/8$, any such graph is ϵ-approximately 1-embeddable iff it is (exactly) 1-embeddable, and the desired result is at hand. □

It is interesting to examine Theorem 5.1 to see just what it is saying in terms of language recognition. For each non-negative real number, ϵ, let L_ϵ be the language consisting of all descriptions (in some agreed-upon form) of ϵ-approximately 1-embeddable integer-weighted graphs. For each ϵ in the interval $[0,1/8)$, the language L_ϵ is a superset of L_0 and a strict subset of $L_{1/8}$. There are also many other languages which contain L_0 and are contained in L_ϵ, for some $\epsilon < 1/8$, but which are not equal to L_ϵ for any ϵ. Theorem 5.1 says that every one of these languages is strongly NP-hard.

It is interesting to note that Approximate 1-Embeddability problems restricted to graphs consisting of a single cycle (such as were used in the proof of Theorem 3.2) are always solvable in polynomial time if e_2 is positive.[6] This shows that the weak NP-completeness result given in Section 3 does not say all there was to say about the difficulty of the practical (*i.e.*, with inexact data, etc.) form of the problem. Loosely speaking, we could say that we have shown the notion of strong *vs.* weak NP-completeness to be significant even for problems that naturally involve reals rather than integers. It should be noted, however, that Theorem 5.1 followed not from Theorem 4.4 but rather from the particular construction used in the proof of Theorem 4.4.

The proof of Theorem 5.1 depended on the fact that, for sufficiently small ϵ, ϵ-approximate 1-embeddability is equivalent to ordinary 1-embeddability for the class of weighted graphs we constructed in our proof of Theorem 4.1. By making this same observation regarding approximate k-embeddability of the weighted graphs constructed in the proof of Theorem 4.4, we arrive at the following result.

<u>Theorem</u> <u>5.2</u>:

Let k be any positive integer. Then there exists a positive real number, ϵ, such that $0,\epsilon$-Approximate k-Embeddability of integer-weighted graphs is NP-hard.

<u>Proof</u>:

The argument is outlined in the above text. Details are left to the reader. □

It has also been pointed out[7] that ϵ_1,ϵ_2-Approximate k-Embeddability of

[6]This follows from the existence of fast approximation algorithms for Partition. See, for example, Lawler [1977].

[7]The author regretfully cannot recall which participant at the 1979 Allerton conference made this observation; he is willing and eager to accept reminders or clues.

integer-weighted graphs is in NP[8] whenever $\epsilon_2 > \epsilon_1$. This may be seen by considering an algorithm which nondeterministically assigns vertices to points in k-space whose coordinates must all be multiples of $(\epsilon_2-\epsilon_1)/k^{1/2}$.

6. Ambiguous Embedding Problems

Another variation on the embeddability problem that may arise in practical applications is that of "ambiguity of solution." Given an incomplete weighted graph and some embedding of that graph into a Euclidean space, we may wish to know whether the given embedding is unique. For example, are the nodes of our sensor network really where we think they are, or might they be in some very different configuration? To pose the problem more precisely, we introduce the following definitions.

Definitions:

Let G be a weighted graph and k be a positive integer. Then two k-embeddings, f and g, of G are said to be congruent iff for each two vertices, u and v, of G, $|f(u)-f(v)| = |g(u)-g(v)|$. A k-embedding, f, of G is said to be unique (up to congruence) iff every k-embedding of G is congruent to f, and in this case G is said to be uniquely k-embeddable. If G has two or more non-congruent k-embeddings, then G is ambiguously k-embeddable.

For any positive integer, k, we may now define the problem of Ambiguous k-Embedding as follows:

Problem (Ambiguous k-Embedding):

Given a weighted graph, G, and a k-embedding, f, of G, determine whether G is ambiguously k-embeddable (*i.e.*, whether there exists a k-embedding of G which is not congruent to f).

In this section, we will show that the Ambiguous Embedding problems defined above are just as hard as the ordinary Embeddability problems we studied in

[8] Strictly speaking, at least one language including all descriptions of ϵ_1-approximately k-embeddable integer-weighted graphs and containing only descriptions of ϵ_2-approximately k-embeddable integer-weighted graphs is in NP.

Sections 2 through 4. The methods we will use are of general interest in that they are potentially applicable to "ambiguous" versions of many other NP-complete problems.

We will begin by formalizing the idea of "ambiguous" versions of problems. Since we hope that the methods of this section will find more widespread application, we will work in a more general setting than is necessary for the task at hand. For this same reason, our presentation of these ideas will be somewhat more formal and more attentive to mathematical fine points than it would be otherwise.

For the purpose of relating a problem, X, to the language classes P and NP, we normally phrase X as a recognition problem; we identify X with a language, L, such that we may ask whether any *instance*, I, of X is in L. We are concerned here with cases in which the defining property of L is that I is in L iff there exists some object, O, such that P(I,O), for some fixed predicate, P, which we call a *defining predicate* for X.[9] In such a case, we refer to an O such that P(I,O) as a *solution* of I.

We sometimes wish to regard two solutions of (an instance of) a problem as essentially the same even if they are not actually identical. We may do this by introducing an equivalence relation, ≡, on the space of potential solutions. Note that ≡ must be such that if O ≡ O′ then for any problem instance, I, P(I,O) iff P(I,O′); such an equivalence relation is said to *respect* the predicate P. Given a problem, X, defined by a predicate, P, and given an equivalence relation, ≡, which respects P, we may define an "ambiguous" version of X as follows:

Problem (Ambiguous X up to ≡):

Given an instance, I, of X and a solution, O, of I, determine whether there exists a solution, O′, of I such that O′ ≢ O.

Note that the problem of Ambiguous k-Embedding defined above may now be

[9]For example, if X is the problem of 1-Embeddability of integer-weighted graphs, then an instance, I, of X is a description of an integer-weighted graph; the language, L, consists of all descriptions of 1-embeddable integer-weighted graphs; and the predicate, P, might be defined so that P(I,O) is TRUE iff O is a 1-embedding of the integer-weighted graph described by I.

described as "Ambiguous k-Embeddability up to congruence." A subtle point which may have escaped the reader's attention is that different predicates may define the same language,[10] and the definition of "Ambiguous X up to ≡" depends on the defining predicate, P, as well as on ≡. In the text below, the intended P should always be clear from context.

When we speak of "Ambiguous X" (without mention of any ≡), for some previously defined X, we will mean "Ambiguous X with respect to equality." Following this convention, we can embark on the path to showing the NP-hardness of Ambiguous k-Embedding problems, by defining the problems of Ambiguous 3-Satisfiability and Ambiguous 4-Satisfiability as follows:

Problems (Ambiguous 3-(4-)Satisfiability):

Given an expression, E, in 3-CNF (resp. 4-CNF) and an assignment of truth values for the variables of E which satisfies E, determine whether there exists any other assignment which satisfies E.

Lemma 6.1:

Ambiguous 4-Satisfiability is NP-complete.

Proof:

We will proceed by reduction from 3-Satisfiability. Consider an expression, E, in 3-CNF with variables $X_1,\ldots,X_N$ and clauses $C_1,\ldots,C_M$. We introduce a new variable, Y, and define a function, F, on Y and the X_i as follows:

$$\begin{aligned} F &= (Y \wedge \prod_{1\le i\le N} X_i) \vee (\bar{Y} \wedge E) \\ &= (\bar{Y} \vee \prod_{1\le i\le N} X_i) \wedge (Y \vee E) \\ &= \prod_{1\le i\le N} (\bar{Y} \vee X_i) \wedge \prod_{1\le j\le M} (Y \vee C_j). \end{aligned}$$

[10]This fact is used to great advantage in the recent work on fast probabilistic tests for primality (see, for example, Rabin [1976]). Briefly, the usual defining predicate for the problem of Compositeness (given a positive integer, I, is I composite?) is given by $P(I,O) \equiv O$ is an integer divisor of I such that $1<O<I$. Unfortunately, by this definition solutions for a given instance may be very rare and hard to find, as in the case where I is the product of two large primes. The fast probabilistic tests rely on other "defining" predicates for Compositeness for which solutions (called "witnesses" in the literature) are guaranteed to be common.

Note that F may be satisfied by assigning the value TRUE to Y and to all the X_i. Any other assignment can satisfy F iff it makes Y FALSE and assigns truth values to the X_i in a way that satisfies E. Finally, it is clear that a 4-CNF expression for F can be constructed from E in polynomial time. □

Lemma 6.2:
Ambiguous 3-Satisfiability is NP-complete.

Proof:
We will show a polynomial-time reduction from Ambiguous 4-Satisfiability to Ambiguous 3-Satisfiability. Consider any expression, E, in 4-CNF. For each clause, $C_j = L_{j,1}+L_{j,2}+L_{j,3}+L_{j,4}$ (where $L_{j,1}$, $L_{j,2}$, $L_{j,3}$, and $L_{j,4}$ are literals of E), of E we introduce a new variable, Q_j, and define C'_j as the following conjunction of clauses:

$$C'_j = (L_{j,1} \vee L_{j,2} \vee Q_j) \wedge (L_{j,3} \vee \bar{L}_{j,4} \vee Q_j) \wedge (\bar{L}_{j,3} \vee L_{j,4} \vee Q_j) \wedge (\bar{L}_{j,3} \vee \bar{L}_{j,4} \vee Q_j) \wedge (L_{j,3} \vee L_{j,4} \vee \bar{Q}_j).$$

Note that for each assignment of truth values to $L_{j,1}$, $L_{j,2}$, $L_{j,3}$, and $L_{j,4}$ such that C_j is satisfied there is exactly one assignment for Q_j such that C'_j is satisfied.[11] We define E' as the conjunction of all the C'_j. It follows that for each assignment, A, to the variables, X_i, of E which satisfies E there is exactly one assignment, B, of the Q_i such that E' is satisfied by $A \cup B$. Finally, it is clear that E' and B can be computed in polynomial time from E and A. □

In the previous proof, we reduced Ambiguous 4-Satisfiability to Ambiguous 3-Satisfiability by exhibiting a reduction from ordinary 4-Satisfiability to ordinary 3-Satisfiability in such a way that there exists a polynomial-time-computable 1-1 correspondence between the solutions (*i.e.*, satisfying assignments) of instance of 4-Satisfiability (*i.e.*, an expression in 4-CNF) and the solutions of the instance of 3-Satisfiability to which it is reduced. We may generalize this technique by defining

[11]Drawing a 5-variable Karnaugh map for the terms of C'_j (with Q_j as the fifth variable) will make the truth of this assertion immediately clear.

the concept of a *solution-preserving* reduction:

Definitions:

Let X and Y be problems defined by predicates P_X and P_Y, respectively, and let $\equiv_X$ and $\equiv_Y$ be equivalence relations respecting P_X and P_Y respectively. Then a *reduction* from X to Y is a polynomial-time function from instances of X to instances of Y such that for any instance, I_X of X, solutions for $f(I_X)$ exist iff solutions for I_X exist. A reduction is said to be solution-preserving from $\equiv_X$ up to $\equiv_Y$ if there exists a 1-1 function, G, sending equivalence classes under $\equiv_X$ to equivalence classes under $\equiv_Y$ and having the following properties:

1. For any instance, I_X, of X, the restriction of G to the set of all equivalence classes whose elements are solutions to I_X is onto the set of all equivalence classes whose elements are solutions to $f(I_X)$.

2. There exists a polynomial-time computable function, g, such that, for any instance, I_X, of X and any[12] solution, O, of I_X,

   $$g(I,O) \in G([O]),$$

   where [O] is the equivalence class (under $\equiv_X$) of which O is a representative.

We may now state a lemma which will be of use in proving the NP-hardness of the "ambiguous" versions of various problems.

Lemma 6.3:

Let X and Y be problems defined by predicates P_X and P_Y, respectively, and let $\equiv_X$ and $\equiv_Y$ be equivalence relations which respect P_X and P_Y, respectively. Let f be a reduction from X to Y which is solution-preserving from $\equiv_X$ up to $\equiv_Y$. Suppose that Ambiguous Y up to $\equiv_Y$ is NP-hard. Then Ambiguous X up to $\equiv_X$ is NP-hard.

[12]Please go back and finish reading the definition before looking at this footnote. There is a subtle point being glossed over here. The function g must operate on representations of solutions rather than actual solutions, and not all solutions will necessarily be representable (for example, any weighted graph of three vertices and two edges has uncountably many 2-embeddings which are distinct with respect to congruence). Note that the definition of "Ambiguous X up to $\equiv_X$" depends on the selection of some scheme for representing solutions to instances of X. We only require g(I,O) to be defined in the case that O is representable under the chosen scheme.

Proof:

Produce functions G and g as given by the preceding definition. We note that any instance, (I_X,O_X) of Ambiguous X up to $\equiv_X$ can be reduced in polynomial time to (I_Y,O_Y), where $I_Y = f(I_X)$ and $O_Y = g(I_X,O_X)$. Moreover I_Y has solutions which are not equivalent to O_Y under $\equiv_Y$ iff I_X has solutions which are not equivalent to O_X under $\equiv_X$. □

Observe that our proof of Lemma 6.2 depends simply on the fact that the transformation from E to E′ is a reduction from 4-Satisfiability to 3-Satisfiability which is solution-preserving from equality up to equality. We now employ Lemma 6.3 to show the results claimed at the beginning of this section.

Theorem 6.4:

Ambiguous 1-Embedding of {1,2}-weighted graphs is NP-complete and Ambiguous k-Embedding of {1,2}-weighted graphs is NP-hard for any positive integer k.

Proof:

This result is a consequence by Lemma 6.3 of the following easily verified facts:

1. The reduction used in the proof of Theorem 4.1 is solution-preserving from equality up to congruence in 1-space.

2. For any positive integer, k, the reduction used in the proof of Corollary 4.3 is solution preserving from congruence in k-space up to congruence in k-space.

3. For any positive integer, k, the reduction used in the proof of Lemma 4.4 is solution preserving from congruence in k-space up to congruence in (k+1)-space.

□

Corollary 6.5:

Ambiguous 1-Embedding of integer-weighted graphs is strongly NP-complete and Ambiguous k-Embedding of integer-weighted graphs is strongly NP-hard for any positive integer, k.

Proof:

Trivial from Theorem 6.4. □

7. Conclusions

The results of this paper fall into two classes, those of interest to persons concerned with embedding problems (such as the sensor positioning problem) and those that are of more general theoretical interest. To those concerned with finding efficient solutions to the Embedding problem (given a weighted graph, find "the" embedding), these results say what all NP-completeness results say: "You are trying to solve the wrong problem." Rather than looking for an efficient worst-case algorithm, it would be more promising to seek an algorithm that gives good performance in cases which arise in practice (for example, cases in which the graph is highly overconstrained). Pursuing this topic, we present in Appendix II a linear-time algorithm for determining whether any *complete* graph is k-embeddable (for any fixed k). Some other positive results are given by Yemini [1979].

The most specific result of theoretical interest is our discovery of some new *strongly* NP-hard geometric problems, and our use of some interesting gadgets to carry out the proofs of NP-hardness. Of more general interest are the two new classes of problems introduced in Sections 5 and 6. The "ϵ_1,ϵ_2-approximate" problems introduced in Section 5 offer a new way of looking at the notion of NP-completeness in the context of problems involving continuous variables. As we have seen, weak NP-completeness may not say all there is to say in this context. "Ambiguous solution" problems address the question of determining whether a known solution to a problem is in fact the unique solution. In Section 6, we exhibited a fundamental NP-complete problem, 3-Satisfiability, whose ambiguous version is also NP-complete, and exhibited a method for obtaining new NP-completeness results for "ambiguous" versions of other problems, namely the use of reductions that preserve uniqueness of solution.

Acknowledgements

The author would like to thank Steven E. Saunders for leading him to study the problem of Embeddability, Michael I. Shamos for helpful discussions, and Jon L. Bentley for his encouragement and advice during the preparation of this document.

References

Cook, S. A. "The complexity of theorem proving procedures." *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, New York (1971). pp. 151-158.

Distributed Sensor Nets. Proceedings of a conference sponsored by the Information Processing Techniques Office, Defense Advanced Research Projects Agency and hosted by Carnegie-Mellon University, December, 1978.

Garey, M. R. and D. S. Johnson. *Computers, Complexity, and Intractability*. Freeman, San Francisco (1979).

Lawler, E. L. "Fast approximation algorithms for knapsack problems." *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Long Beach, CA (1977). pp. 206-213.

Papadimitriou, C. H. "The NP-completeness of the bandwidth minimization problem." *Computing* 16 (1976). pp. 263-270.

Rabin, Michael O. "Probabilistic Algorithms." In *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, Ed. Academic Press. New York (1976). pp. 21-39.

Shamos, M. I. Personal communication (1978).

Yemini, Y. "On some theoretical aspects of position-location problems." *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*. IEEE. October 29-31, 1979. pp. 1-8.

Yemini, Y. "The positioning problem--A draft of an intermediate summary." In *Distributed Sensor Nets* [1978].

Appendix I: Reduction from Bandwidth Minimization

In this appendix we give a second proof of Theorem 4.2, using a reduction from the problem of Bandwidth Minimization.

Definitions:

Let G be a graph with vertex set V, and let N = |V|. A layout of G is a one-to-one mapping, f, from V onto {1,...,N}. The bandwidth of f is defined as the maximum distance between the images under f of any two vertices that are connected by an edge of G. That is,

$$\text{bandwidth}(f) = \max\{|f(u)-f(v)| \mid \{u,v\} \text{ is an edge of } G\}.$$

The bandwidth of G is defined as the least possible bandwidth for any layout of G. Thus,

$$\text{Bandwidth}(G) = \min\{\text{bandwidth}(f) \mid f \text{ is a layout of } G\}.$$

Problem (Bandwidth Minimization):

Given an arbitrary graph, G, and a positive integer, k, determine whether Bandwidth(G) $\leq$ k.

Bandwidth Minimization was shown to be NP-complete by Papadimitriou [1976]. Using this result, we can give a second proof of the following theorem:

Theorem I.1 (Corollary 4.2):

1-Embeddability of integer-weighted graphs is strongly NP-complete.

(Second) Proof:

We will proceed by reduction from Bandwidth Minimization. Let G be a graph of N vertices, and let k be a positive integer. We assume without loss of generality that k $\leq$ N. We now construct an edge-weighted graph, G′, as follows:

1. For each vertex, v, of G, let there be a distinct vertex, v′, of G′. G′ will also have some additional vertices as required by the remaining steps of the construction.

2. For each edge, {u,v}, of G, connect u′ and v′ by a chain of k edges, one (it doesn't matter which) having weight (k+1)/2 and the rest having weight 1/2.

3. For each two vertices, u and v, of G which are not connected by an edge of G, connect u′ and v′ by a chain of N-1 edges, one having weight N/2 and the rest having weight 1/2.

Note that for any bandwidth $\leq$ k layout, f, of G, there exists at least one 1-embedding, f′, of G′ such that f′(v′) = f(v) for every vertex, v, of G. Similarly from every 1-embedding of G′ we can derive a bandwidth $\leq$ k layout of G. If we now double the weights of all the edges of G′, we get an integer-weighted graph, G", having the following properties:

1. G" is 1-embeddable iff G has a layout of bandwidth $\leq$ k and

2. A representation of G" with all edge weights given in unary has size polynomial in the size of a representation of G, and can be computed from a representation of G in polynomial time.

This completes the proof. □

It is interesting to note that this construction can not be used as a basis for the results of Sections 5 and 6. This underscores our earlier remark that the proof of Theorem 5.1 relies not simply on the strong NP-completeness of 1-Embeddability of integer-weighted graphs, but on the particular construction used in the proof of Theorem 4.1.

Appendix II: Embeddability of Complete Graphs

In this appendix, we exhibit a class of polynomial-time algorithms due to Shamos [1978] for testing the k-embeddability of complete weighted graphs. For purposes of exposition, we assume a model in which real numbers are primitive data objects on which exact arithmetic operations (including comparisons and extraction of square roots) can be performed in constant time. Within this model, we have the following result.

Theorem II.1:
Let k be any positive integer. Then there exists an algorithm for testing the k-Embeddability of complete weighted graphs which runs in time linear in the number of edges (or, equivalently, quadratic in the number of vertices) of the graph being tested.

Proof:
Let G = ⟨V,E,W⟩ be a complete weighted graph with N vertices, $X_1,\ldots,X_N$. To test the embeddability of G, we will attempt to position successively the vertices of G in a (k+1)-dimensional coordinate space. Without loss of generality, we may send X_1 to the origin and X_2 to $(W(\{X_1,X_2\}),0,\ldots,0)$. For each M, $1 \leq M \leq N$, we define

$D(M) = \min \{j \mid$ the complete weighted graph on $\{X_i | 1 \leq i \leq M\}$ induced from G is j-embeddable$\}$.

If the induced weighted graph on $\{X_1,...,X_M\}$ is not j-embeddable for any j, then D(M) is undefined. For each j, $0 \leq j \leq k$, we define

$P(j) = \min \{M \mid D(M) = j\}$.

If there is no M such that D(M) = j, then P(j) is undefined. Note that if P(j) is well defined, then P(0),...,P(j) are all defined and distinct. As we locate each vertex, we enforce the restriction that at most the first D(M) coordinates of X_M may be non-zero. By following this rule, we guarantee that after the X_M has been located (if this is possible), we will know the value of D(M) and of P(0),...,P(D(M)). The procedure for locating the X_{M+1} (for $1 \leq M < N$) is as follows:

1. Note that there is at most one possible location for X_{M+1} which will satisfy the following criteria:

   - The correct weights are induced for the D(M)+1 edges $\{X_{P(j)},X_{M+1}\}$, $0 \leq j \leq D(M)$.

   - At most the first D(M)+1 coordinates of the location are non-zero.

   - The (D(M)+1)-st coordinate of the location is non-negative.

   This location, if it exists, may be discovered in constant time, since we will always have $D(M) \leq k$.

2. If there are no such locations, or if the (k+1)-st coordinate of the unique location satisfying the criteria is non-zero, halt asserting that G is not k-embeddable. Otherwise, without loss of generality, assign X_{M+1} to the unique location satisfying the criteria.

3. Check that the weights induced for the remaining $\{X_i,X_{M+1}\}$ (where $1 \leq i \leq M$ and $i \neq P(j)$ for any j) are correct. If any are not, then halt asserting that G is not k-embeddable. Note that the time for this step is O(N), since we always have $M < N$.

If we manage to place all the vertices without discovering that G is not k-embeddable, then we will have found a k-embedding for G (and this embedding is unique up to congruence). In any case, the time required is linear

in the number of edges and the space will be linear in the number of vertices.
□

It may be noted that the algorithm given here will not only work for complete weighted graphs, but may be generalized to apply to a large class of incomplete weighted graphs as well; it is only necessary that it be possible to order the vertices such that, when we attempt to locate the vertices in order, each vertex is connected to sufficiently many previously-located vertices that the new vertex can be assigned a unique location without loss of generality. To see the limits of this generalization, however, we need only consider the graph shown in Figure II.1. If the vertices of this graph are assigned to points in the plane in such a way that no three are colinear, then a set of edge weights will be induced which make the graph uniquely 2-embeddable. But if any vertex is removed, the weighted subgraph induced on the remaining five vertices will have infinitely many non-congruent 2-embeddings.

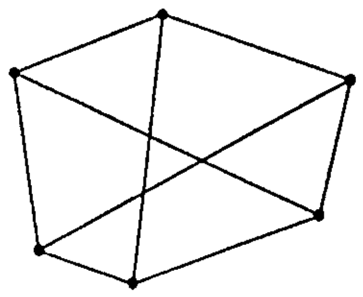


Figure II.1 A uniquely embeddable graph with no triangles.

Further explorations in this direction would take us beyond the scope of this paper. Yemini [1978] exhibits a number of interesting "counterexamples" of the flavor of Figure II.1. We also leave untouched the issues of numerical stability which arise when the preceding algorithm is performed with inexact arithmetic, and possibly on inexact data.

Dynamic-Programming Algorithms for Recognizing Small-Bandwidth Graphs in Polynomial Time

James B. Saxe
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this paper we investigate the problem of testing the bandwidth of a graph: Given a graph, G, can the vertices of G be mapped to distinct positive integers so that no edge of G has its endpoints mapped to integers which differ by more than some fixed constant, k? We exhibit an algorithm to solve this problem in $O(f(k)N^{k+1})$ time, where N is the number of vertices of G and f(k) depends only on k. This result implies that the "Bandwidth $\overset{?}{\leq}$ k" problem is not NP-Complete (unless P = NP) for any fixed k, answering an open question of Garey, Graham, Johnson, and Knuth. We also show how the algorithm can be modified to solve some other problems closely related to the "Bandwidth $\overset{?}{\leq}$ k" problem.

This work was supported by the Office of Naval Research under Contract N00014-76-C-0370.

1 Introduction

The subject of this paper is the computational complexity of a problem on graphs. To speak precisely of the problem, we will need the following notation and definitions.

Notation:

Let u and v be vertices of a graph G. We will say "u—v in G" to denote that {u,v} is an edge of G. Where G is clear from context, we will write simply "u—v".

Definitions:

Let G be a graph with vertex set V, and let $N = |V|$. A layout of G is a one-to-one mapping, f, from V onto $\{1,...,N\}$. The bandwidth of f is defined as the maximum distance between the images under f of any two vertices that are connected by an edge of G. That is,

$$\text{bandwidth}(f) = \max\{f(u)-f(v) \mid u\text{—}v\}.$$

The bandwidth of G is defined as the least possible bandwidth for any layout of G. Thus,

$$\text{Bandwidth}(G) = \min\{\text{bandwidth}(f) \mid f \text{ is a layout of } G\}.$$

Problem (Bandwidth Minimization):

Given an arbitrary graph, G, and a positive integer, k, determine whether $\text{Bandwidth}(G) \leq k$.

Note that the notion of graph bandwidth is equivalent to the more familiar notion of matrix bandwidth in that $\text{Bandwidth}(G) \leq k$ iff there exists a permutation matrix P such that $(PCP^{-1})_{i,j} = 0$ whenever $|i-j| > k$, where C is G's connection matrix. For any particular positive integer k, we can define a restricted version of the bandwidth minimization problem as follows:

Problem (Bandwidth $\overset{?}{\leq}$ k):

Given a graph, G, determine whether $\text{Bandwidth}(G) \leq k$.

Papadimitriou [1976] has shown that the general bandwidth minimization problem, in which k is specified in the input, is NP-Complete. The problem was later studied by Garey, Graham, Johnson, and Knuth [1978], who found a linear-time algorithm for the the problem "Bandwidth $\overset{?}{\leq}$ 2", and also improved on Papidimitriou's result by showing the problem for general k to be NP-Complete even when G is restricted to be a tree with no vertex of degree greater than three. A number of questions are left open by their work, however. One such question is whether there exists a polynomial-time algorithm for the problem "Bandwidth $\overset{?}{\leq}$ 3". In this paper, we will answer this question affirmatively by exhibiting an algorithm[1] which solves the problem "Bandwidth $\overset{?}{\leq}$ k" in polynomial time for any fixed k. Section 2 of this paper introduces the fundamental concepts and assumptions we will use in describing our algorithm. In Section 3 the algorithm is described and its performance is analyzed. In Section 4 we discuss some modifications of the algorithm to solve related problems. Finally, in Section 5, we discuss some remaining open problems.

2 Fundamental Concepts and Assumptions

Throughout the following we will assume that G denotes a graph with vertex set V and edge set E, that k denotes a particular positive integer,[2] and that we wish to determine whether G has any layout of bandwidth $\leq$ k. We let N denote the cardinality of V. Note that if G is not connected then G has a layout of bandwidth $\leq$ k iff each of its components has such a layout. Also, it is clearly impossible for G to have such a layout if G has any vertex of degree 2k or greater. We therefore assume, without loss of generality, that G is a connected graph having no vertex of degree greater than or equal to 2k. Note that an arbitrary graph can be partitioned into its connected components by depth-first search in O(max(n,e)) time, where n is the number of vertices and e is the number of edges,[3] and that this is O(n) if a

[1]More correctly, a class of algorithms, one for each value of k.

[2]When using the "big-oh" notation, we will regard k as fixed and therefore omit factors that depend only on k.

[3]See, for example, Aho, Hopcroft, and Ullman [1974, Chapter 5].

fixed bound is given on the degree of any vertex. Moreover, an obvious modification to the depth-first search algorithm allows it to detect the presence of a vertex with degree greater than a fixed bound in time which is proportional only to the number of vertices and not to the number of edges.

We now introduce the key notion of a partial layout.

Definitions:

A partial layout of G is a one-to-one function, f, from some subset of V onto $\{1,\ldots,M\}$, for some M such that $0 \leq M \leq N$. We say that f is feasible if it can be extended to a (total) layout, g, such that bandwidth(g) $\leq$ k. The bandwidth of f is the maximum distance between the images of any two edge-connected vertices of G which are in the domain of f. If u—v and u is in the domain of f and g is not, then the edge $\{u,v\}$ is said to be dangling from f.

Consider a partial layout, f, of size M. Clearly, f cannot possibly be feasible unless

1. bandwidth(f) $\leq$ k, and
2. whenever u and v are vertices of G such that $f(u) < M-k$ and u—v, v is also in the domain of f.

If f satisfies both these conditions, then f is said to be a plausible partial layout. The sequence $(f^{-1}(\max(M-k+1,1)),\ldots,f^{-1}(M))$, taken together with the set of dangling edges of f, is called the active region of f. We now come to the theorem on which our principal algorithm depends.

Theorem 2.1:

Let f and g be two plausible partial layouts of G having identical active regions. Then,

1. f and g have identical domains, and
2. f is feasible iff g is feasible.

Proof:

Since G is connected, the domains of f and g must each consist precisely of those vertices which are path-connected to vertices in the active region by paths not including any dangling edges. Thus, (1) holds. To see that (2) holds, we need only note that any assignment of the remaining vertices which extends either f or g to a total layout of bandwidth $\leq$ k must also extend the other to such a layout. □

Finally, we define the notion of a successor of a plausible partial layout (or active region), which will be necessary to explain our algorithms.

Definition:

Let f be a plausible partial layout of G. Then a successor of f is a plausible partial layout, g, which extends f by precisely one element. In this case, the active region of g is also said to be a successor of the active region of f. We also say that (the active region of) f is a predecessor of (the active region of) g.

3 The Algorithm

Theorem 2.1 allows us to say that two plausible partial layouts are equivalent if they have identical active regions. The algorithm we present is essentially a breadth-first search over the space of all the induced equivalence classes of plausible partial layouts, where each such equivalence class is uniquely characterized by active region of its representatives. Alternatively, we may think of the algorithm as a dynamic-programming search over the plausible partial layouts. Each active region consists of at most k vertices and each vertex has no more than 2k edges, each of which may or may not be dangling. Thus the number of equivalence classes is bounded above by[4]

$$\sum_{0 \leq i \leq k} \binom{N}{i} (2^{2k})^i = O(N^k).$$

Our algorithm uses the following two data structures:

1. A (fifo) queue, Q, whose elements are active regions.

[4]As we will mention in Section 5, the coefficient on this bound is quite loose.

2. An array, A, which contains one element for each possible active region. Each element, A[r], of A consists of a Boolean flag, A[r].examined, telling whether the active region r has already been considered in the search and a list, A[r].unplaced, of vertices which is intended to list all vertices NOT in the domain of each plausible partial layout with active region r.

At the start of our algorithm, Q is initialized to contain the single element representing the active region (henceforward denoted Φ) of the empty partial layout, ϕ. The flag A[Φ].examined is set to TRUE and A[Φ].unplaced is initialized to list all the elements of V. The remaining A[r].examined are initially FALSE, and the remaining A[r].unplaced are uninitialized. The algorithm now proceeds as follows:

<u>Algorithm</u> <u>B</u> (Bandwidth testing):

```
1. Extract an active region, r, from the head of Q.
2. From A[r].unplaced, determine the successors of r.
3. For each successor, s, of r such that A[s].examined is FALSE, perform
   the following steps:
      a. Set A[s].examined to TRUE.
      b. Compute A[s].unplaced by deleting the last vertex of s from
         A[r].unplaced.
      c. If A[r].unplaced is the empty set, then halt asserting that
         Bandwidth(G) ≤ k.
      d. Insert s at the end of Q.
4. If Q is empty, then halt asserting that Bandwidth(G) > k. Otherwise, go
   to Step 1.
```

The space required by this algorithm is clearly $O(N^{k+1})$. To determine the running time, we note first that since there are $O(N^k)$ possible active regions, each of Steps 1 through 4 will be executed $O(N^k)$ times. The individual executions of Steps 1 and 4 each take only constant time, so the contribution of these steps to the total running time of the algorithm is $O(N^k)$. Since any active region, r, has at most N successors (zero or one for each element of A[r].unplaced), each execution of Step 2 takes O(N) time. The contribution of Step 2 to the total execution time is therefore $O(N^{k+1})$. Determining the contribution of Step 3 is (a little) trickier. During a single execution of Step 3, Steps 3.a through 3.d may be executed as many as N times, and the amount of computation in Step 3.b may be $\theta(N)$. Thus it appears possible that Step 3 may contribute $\theta(N^{k+2})$ to the total execution time. If we look more carefully, however, we see that 3.a through 3.d are executed at most once for each active region. Thus the total contribution of Step 3 is $O(N^{k+1})$. Adding the contributions of all the steps gives us the following result.

Theorem 3.1:

Let k be any positive integer. Then there is an algorithm which solves the problem "Bandwidth $\overset{?}{\leq}$ k" using $O(N^{k+1})$ time and $O(N^{k+1})$ space.

Proof:

To test the bandwidth of G, we first perform an O(N)-time depth first search which either

(1) determines that G has some vertex of degree greater than 2k, or

(2) partitions G into connected components none of which have any vertex of degree greater than 2k.

In case (1), we know immediately that Bandwidth(G) $\geq$ k. In case (2), we apply Algorithm B to the connected components of G. □

While Algorithm B will tell us whether G has a layout of bandwidth $\leq$ k, it does not actually produce such a layout. In order to allow such a layout to be recovered, we may associate with each active region, s, an additional field, A[s].predecessor. When s is appended to Q in Step 3.f., we make A[s].predecessor point to a predecessor of s (namely the r we chose in Step 1).[5] If the algorithm finds an active region, t, such that A[t].unplaced is empty, it is a simple matter to recover a layout by tracing back through the predecessor fields.

[5] Note that this pointer need only name the single vertex (if any) which is contained in r but not in s.

4 Modifications for Related Problems

Another question left open by Garey, Graham, Johnson, and Knuth [1978] is whether there exists a polynomial-time algorithm to *count* the layouts of a graph having bandwidth $\leq$ k, even for k = 2. We now give an affirmative answer to a closely related question by exhibiting a class of polynomial-time algorithms (one for each positive integer k) for determining the number of bandwidth $\leq$ k layouts of any *connected* graph.[6]

Our algorithm for enumerating layouts of bandwidth $\leq$ k is a slightly modified form of Algorithm B. The data structures are the same as those for Algorithm B, with the following additions:

1. Each entry, A[r], of A has a third field, A[r].count, which will hold the number of (so far discovered) plausible partial layouts whose active region is r.
2. There is a variable, Total, which will hold the number of (so far discovered) layouts of bandwidth $\leq$ k.

At the start of the algorithm, Total and all the A[r].count are initialized to zero, except for A[Φ], which is initialized to 1. The remaining variables are initialized as for Algorithm B. We then proceed as follows:

Algorithm E (Enumerate layouts):

```
1. Extract an active region, r, from the head of Q.
2. From A[r].unplaced, determine the successors of r.
3. For each successor, s, of r, perform the following steps:
      a. If A[s].examined is TRUE, go to f.
      b. Set A[s].examined to TRUE.
      c. Compute A[s].unplaced by deleting the last vertex of s from
```

[6]Note that the number of bandwidth < k layouts of an arbitrary graph is not uniquely determined by the numbers of bandwidth < k layouts of its connected components because the topologies of the components impose constraints on how the various layouts may overlap. The algorithms cannot be applied directly to non-connected graphs because they depend on Theorem 2.1.

```
        A[r].unplaced.
    d. If A[r].unplaced is the empty set, then increase Total by
       A[r].count.
    e. Insert s at the end of Q.
    f. Increase A[s].count by A[r].count.
4. If Q is empty, then halt. Otherwise, go to Step 1.
```

Study of this algorithm gives us the following result:

Theorem 4.1:

Let k be any positive integer. Then there exists an $O(N^{k+1})$-time, $O(N^{k+1})$-space algorithm which, given any *connected* graph, G, computes the number of layouts of G having bandwidth $\leq k$.

Proof:

We claim that Algorithm E (preceded by a depth-first search to ensure that no vertex of G has degree greater than 2k) has the desired properties. By an analysis similar to that for Algorithm B, Algorithm E will run in $O(N^{k+1})$ time. We must now show that it correctly counts the layouts of bandwidth $\leq k$. To do this, it suffices to show that by the time that any plausible partial layout, r, is selected in Step 1, A[r].count contains the total number of plausible partial layouts whose active region is r. This in turn may be shown inductively if we can only show that no active region, r, is chosen in Step 1 until every predecessor of r has been chosen. This last follows at once from the fact (which may be established by induction) that the active regions proceed through the queue in non-decreasing order of their lengths, where the length of an active region, r, is defined to be the number of vertices in the domain of any plausible partial layout whose active region is r. □

We may view Bandwidth Minimization as the problem of finding a layout with minimax edge length. We will now look at the corresponding minisum problem.

Definition:

Let G be a graph with edge set E, and let f be a layout of G. Then the total edge length of f is given by the sum

$$\sum_{\{u,v\} \in E} |f(u)-f(v)|$$

where each edge, {u,v}, contributes precisely once to the sum (rather than once as u—v and once as v—u).

Problem: (Optimal Linear Arrangement)

Given a graph, G, and an integer, t, determine whether there is a layout of G having total edge less than or equal to t.

The Optimal Linear Arrangement (O.L.A.) problem was found to be NP-Complete by Garey, Johnson, and Stockmeyer [1976]. However, Shamos [1979] has pointed out that the methods of the present work can be used to provide polynomial-time algorithms for a class of restricted versions of O.L.A. For every positive integer, k, we define a restriction of O.L.A. as follows:

Problem: (O.L.A. for bandwidth $\leq$ k)

Given a graph, G, determine the minimal total edge length of any layout of G having bandwidth $\leq$ k or determine that no such layout exists.

Applying the methods used above, we obtain the following result.

Theorem 4.2:

Let k be any positive integer. Then there exists an algorithm which solves O.L.A. for bandwidth $\leq$ k in $O(N^{k+1})$ time and $O(N^{k+1})$ space.

Proof:

An algorithm having the desired properties when applied to connected graphs with no vertex having degree greater than 2k may be constructed by a slight modification of Algorithm E: instead of maintaining with each active region a count of the partial layouts having that active region, we maintain an indication of the minimum sum of the lengths of all edges whose endpoints have are in the domains of all plausible partial layouts having that active region. The details are left to the reader. For arbitrary graphs we first perform a depth-first search which either detects the presence of a vertex with degree greater than 2k (implying that Bandwidth(G) $\geq$ k) or partitions G into its connected components, taking linear time in either case. We then compute the minimal total edge length for G by finding and summing the minimal total edge lengths for the connected components. □

We note that the previous result remains valid if we consider edge weighted graphs and the "total edge length" is taken as a weighted sum. For connected graphs, we can also use the method of Algorithm E to obtain a count of the layouts with minimal total edge length for bandwidth $\leq k$.

Finally, all the previous results extend to "directed" versions of Bandwidth Minimization and O.L.A., in which G is a directed graph and a layout, f, is acceptable only if $f(u) < f(v)$ whenever (u,v) is an edge of G.[7]

5 Open Problems

The most obvious problem left open by this work is that of improving the performance of Algorithm B. Although the expense of this algorithm is "only polynomial" in the size of the examined graph it is still sufficiently expensive (particularly in terms of space) to render it impractical for all but the smallest cases (consider, for example, determining whether Bandwidth(G) ≤ 5, where G is a graph of forty vertices). The fact that Garey, Graham, Johnson, and Knuth [1978] have a linear-time algorithm for "Bandwidth(G) $\overset{?}{\leq}$ 2", while Algorithm B takes cubic time for the same problem offers some hope that the degree of the polynomial can be reduced for higher values of k as well. Indeed, it is conceivable (even if $P \neq NP$) that there are linear algorithms for all values of k, with coefficients growing exponentially in k.

One approach to improving the performance is to attempt to reduce the number of active regions examined, and this can indeed be done to some extent. For example, we may prune the search by noting that, while a plausible partial layout may have $\theta(k^2)$ dangling edges, such a partial layout cannot actually be feasible if those edges lead to more than k distinct vertices. Unfortunately, graphs of the form

[7] A good starting point for the reader who is interested in learning more about Bandwidth, O.L.A., and their variations is Appendix A1 of Garey and Johnson [1979].

$$v_1 — v_2 — ... — v_{N-1} — v_N$$

supply an existence proof that the number of equivalence classes of plausible partial layouts of bandwidth $\leq$ k can in fact be $\theta(N^k)$.

In Algorithm B, we reduce the search space from the set of all plausible partial layouts to the much smaller set of equivalence classes of partial layouts. To look at it another way, given two partial layouts, f and g, if we recognize (by equality of active regions) that f is feasible iff g is feasible, then we feel free to search for completions of only one of the partial layouts. The algorithm of Garey, Graham, Johnson, and Knuth cuts down the search space by methods which are similar but more sophisticated. In particular, they can avoid searching for completions of a partial layout,[8] f, by choosing to search for completions of a layout, g, such that g is feasible whenever f is feasible, but not necessarily only when f is feasible.

It is interesting to note that "worst-case" numbers of feasible active regions seem to arise precisely in circumstances where large pieces of the graph can be laid out in bandwidth much less than k. We define a <u>maximal</u> graph of size N and bandwidth k as a graph whose edge set is $\{\{v_i,v_j\} \mid |i-j| \leq k\}$, where $\{v_i \mid 1 \leq i \leq N\}$ is the edge set.[9] The algorithm of Garey, Graham, Johnson, and Knuth relies heavily on the fact that if all the even numbered vertices or all the odd numbered vertices are deleted from a maximal graph of bandwidth 2, the induced graph on the remaining vertices is a maximal graph of bandwidth 1. For testing higher bandwidths it is possible that similar use may be made of the fact that deleting every k-th vertex from a maximal graph of bandwidth k leaves a maximal graph of bandwidth k-1.

Another potentially fruitful course of investigation would be to look for efficient algorithms for approximate bandwidth minimization. For example, given a graph, G, we may wish to produce a layout for G whose bandwidth is no more than, say, twice

[8] In their terminology, a partial layout of G is a map from a subset of the vertices of G to an *arbitrary* set of integers.

[9] Note that a graph of N vertices has bandwidth < k iff it is isomorphic to a subgraph of a maximal graph of size N and bandwidth k.

the minimum possible. To the author's knowldge it has not yet been determined whether this problem (when phrased as a language recognition problem) is NP-Complete.

Acknowledgements

The author gratefully acknowledges the helpful comments of Jon L. Bentley, Michael R. Garey, Christos H. Papadimitriou, and Michael I. Shamos.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley. Reading, Massachusetts (1974).

Garey, M. R., R. L. Graham, D. S. Johnson, and D. E. Knuth. "Complexity Results for Bandwidth Minimization." *SIAM Journal on Applied Mathematics* 34 (1978). pp. 477-495.

Garey, M. R. and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company. San Francisco (1979).

Garey, M. R., D. S. Johnson, and L. Stockmeyer. "Some Simplified NP-Complete Graph Problems." *Theor. Comput. Sci.* 1 (1976). pp. 237-267.

Papadimitriou, C. H. "The NP-Completeness of the Bandwidth Minimization Problem." *Computing* 16 (1976). pp. 263-270.

Shamos, M. I. Private communication (1979).