ELSEVIER
Theoretical Computer Science 169 (1996) 23–37
Theoretical Computer Science

The logic engine and the realization problem for nearest neighbor graphs [☆]

Peter Eades [a,*], Sue Whitesides [b]

[a] *Department of Computer Science, University of Newcastle, University Drive, Callaghan, NSW 2308, Australia*
[b] *School of Computer Science, McGill University, 3480 University Street #318, Montreal, Quebec, Canada H3A 2A7*

Abstract

Roughly speaking, a "nearest neighbor graph" is formed from a set of points in the plane by joining two points if one is the nearest neighbor of the other. There are several ways in which this intuitive concept can be made precise.

This paper investigates the complexity of determining whether, for a given graph G, there is a set of points P in the plane such that G is isomorphic to a nearest neighbor graph on P. We show that this problem is NP-hard for several definitions of nearest neighbor graph.

Our proof technique uses an interesting simulation of a mechanical device called a "logic engine".

1. Introduction

This paper investigates the problem of realizing a given graph G as a "nearest neighbor graph" of a set P of points in the plane. Roughly speaking, a "nearest neighbor graph" is formed from a set of points in the plane by joining two points if one is a nearest neighbor of the other. Fig. 1 gives examples of several kinds of nearest neighbor graphs.

A nearest neighbor graph is an example of a *proximity* graph; intuitively, this is a graph which captures notions about the proximity relations between points in space. The Euclidean minimum spanning tree of a set of points provides another example of a proximity graph. Others include Delaunay triangulations, relative neighborhood graphs, Gabriel graphs, and sphere of influence graphs; for a survey, see [15]. Proximity graphs are much studied in the pattern recognition literature, essentially because a proximity graph can be used to capture the "shape" of a set of data points.

[☆] Research supported in part by grants from the Australian Research Council (Eades) and NSERC of Canada and FCAR of Quebec (Whitesides). This work was done in part while Whitesides was visiting University of Newcastle.

[*] Corresponding author. E-mail: eades@cs.newcastle.edu.au.

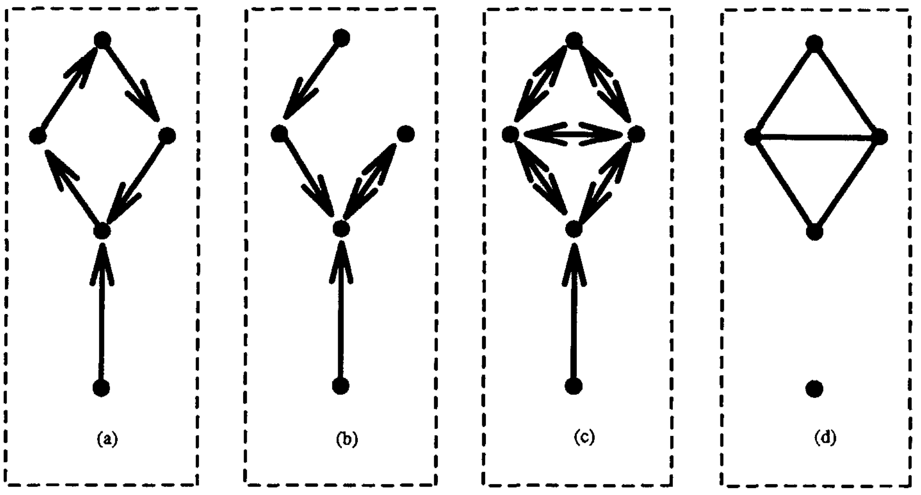

Fig. 1. Several kinds of nearest neighbor graphs, on the same set of points.

Also, the emerging field of graph drawing has stimulated interest in proximity graphs. Graph drawing research seeks to find good geometric representations of graphs, where the notion of good varies with application area. Graph drawing algorithms are used for visualizing relational information, especially in software and information engineering (for example, in CASE tools); see [4] for a bibliographic survey. Intuitively, a geometric representation of a graph as a proximity graph is good because nodes which are related by an edge are close to each other. This intuition has led to a number of investigations of the problem of drawing graphs as proximity graphs:

- Lubiw and Sleumer [13] present characterizations of relative neighborhood graphs.
- Bose et al. [2] show that for several classes of proximity graphs (for example, relative neighborhood graphs, relatively closest graphs, and Gabriel graphs), the problem of determining whether a *tree* can be realized as a proximity graph can be solved in polynomial time.
- Liotta [12] provides a taxonomy of proximity representations of graphs and gives a variety of algorithms for producing the representations.
- Dillencourt [6] investigates and partially solves the problem of drawing a graph as a Delaunay triangulation.
- Eades and Whitesides [7] prove that the realization problem for Euclidean minimum spanning trees is NP-hard.

A brief survey of this approach appears in [5]. The current paper is motivated by graph drawing applications.

Nearest neighbor graphs are perhaps the most primitive kind of proximity graph. For example, for a given set of points, the minimum spanning tree, the Gabiel graph, the relative neighborhood graph and the Delaunay triangulation all contain a nearest neighbor graph (see [15]).

In this paper we show that, given a graph G, it is NP-hard to determine whether a set P of points in the plane can be found such that G is isomorphic to a nearest neighbor graph on P. The result holds for several precise definitions of the intuitive concept of nearest neighbor graph.

To prove this result we introduce a mechanical device called a "logic engine", illustrated in Fig. 2. This device mechanically simulates the well-known NP-complete problem *NOT-ALL-EQUAL-3SATISFIABILITY*. Our results are proved by simulating the logic engine with nearest neighbor graphs. The logic engine is designed from a proof paradigm first used by Bhatt and Cosmodakis [1]; the paradigm has proved very useful in obtaining complexity results for geometric problems (see [7, 11, 10, 3] for examples), and we believe that it can be applied to several classes of proximity graph problems.

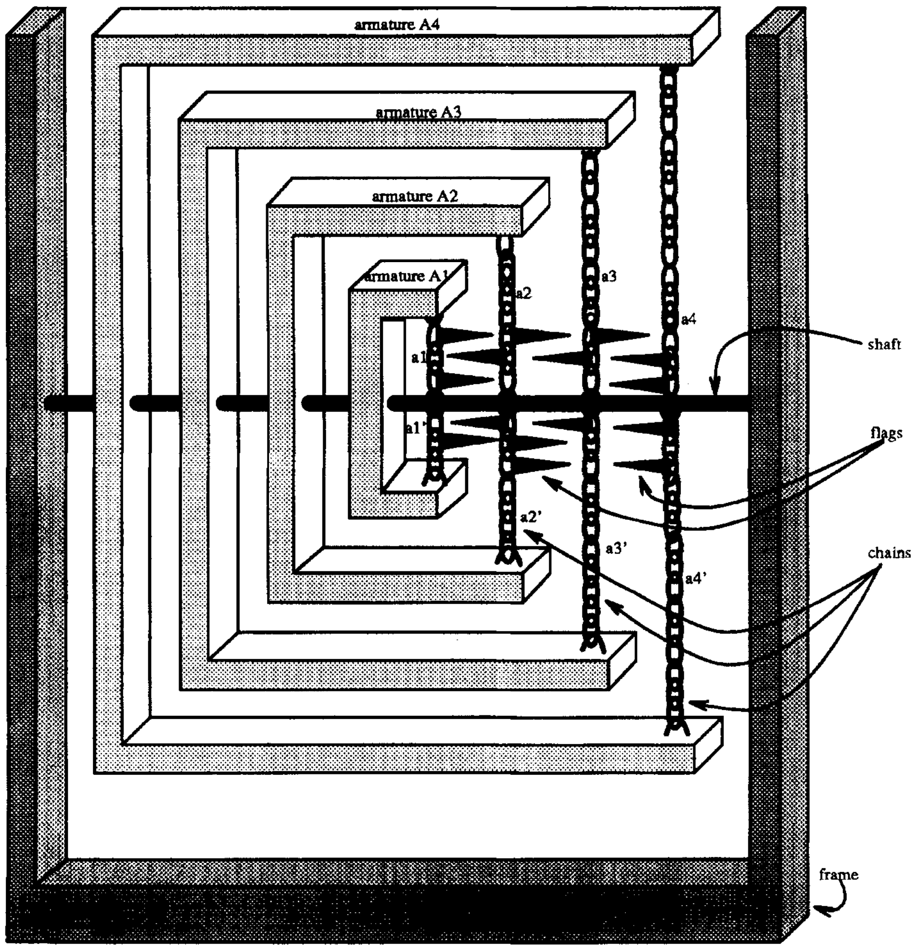

Fig. 2. The essential features of a logic engine.

The following definitions are needed because a point may have more than one nearest neighbor.

Definition 1. Suppose that P is a set of points in the plane. A *weak nearest neighbor graph* G on P is a directed graph with a vertex v_i for each point $p_i \in P$, and an arc set that satisfies two properties:

- If (v_i, v_j) is an arc of G then p_j is a nearest neighbor of p_i. In other words, if there is an arc from v_i to v_j then the open circle centered at p_i, of radius equal to the distance $d(p_i, p_j)$ between p_i and p_j, contains only point p_i from P.
- For each vertex v_i of G, there is exactly one arc (v_i, v_j). In other words, every vertex is joined to exactly one of its nearest neighbors.

Definition 2. Suppose that P is a set of points in the plane. A *strong nearest neighbor graph* G on P is a directed graph with a vertex v_i for each point $p_i \in P$. There is an arc (v_i, v_j) in G if and only if p_j is a nearest neighbor of p_i. That is, each vertex is joined to every one of its nearest neighbors.

Definition 3. Suppose that P is a set of points in the plane. The *mutual nearest neighbor graph* on P is an undirected graph with a vertex v_i for each point $p_i \in P$, and an edge (v_i, v_j) if and only if the points p_i and p_j corresponding to v_i and v_j are each nearest neighbors of the other.

Weak nearest neighbor graphs are in Figs. 1(a) and (b), and a strong nearest neighbor graph is in Fig. 1(c). Fig. 1(d) is a mutual nearest neighbor graph.

For a given set P of points, the strong and mutual nearest neighbor graphs are unique. Note, however, that there may be more than one weak nearest neighbor graph on a set of points. Both the weak nearest neighbor graph and the mutual nearest neighbor graph (with each undirected edge replaced by a pair of oppositely directed arcs) are subgraphs of the strong nearest neighbor graph.

A directed graph G is *realizable as a weak nearest neighbor graph* if there is a set P of points in the plane such that a weak nearest neighbor graph on P is isomorphic to G. Realizability for strong nearest neighbor graphs is defined in the same way. An undirected graph G is *realizable as a mutual nearest neighbor graph* if there is a set P of points in the plane such that the mutual nearest neighbor graph is isomorphic to G. The remainder of this paper is mostly devoted to showing that the following problem is NP-hard.

Mutual Nearest Neighbor Graph Realization (MNNGR)
Instance: An undirected graph G.
Question: Is G realizable as a mutual nearest neighbor graph?

Theorem 1. MNNGR *is* NP-*hard.*

For future reference, we give the definition of the NP-complete problem (see [9]) that we use in the proof of Theorem 1.

Not-All-Equal-3-Sat (NAE3SAT)

Instance: A set C of m clauses $c_1, c_2, \ldots, c_m$, each containing 3 literals from a set X of n boolean variables $X_1, X_2, \ldots, X_n$ and their complements.

Question: Can consistent truth values be assigned to the literals so that each clause contains at least one true literal and at least one false literal?

Without loss of generality, we assume that no clause in C contains both a variable X_j and its complement X_j'. Such clauses are automatically satisfied by every consistent assignment of truth values, and they can be pruned quickly from any NAE3SAT instance.

We also prove the NP-hardness of the realizability problems for strong nearest neighbor graphs and weak nearest neighbor graphs; these results follow easily from Theorem 1 and the results of [7].

The rest of this paper is organized as follows. Section 2 describes the proof paradigm in terms of the *logic engine*, which provides an easy-to-understand introduction to our NP-completeness reduction. Section 3 shows how to simulate the essential properties of a logic engine with a graph to be realized as a mutual nearest neighbor graph; this constitutes a proof of Theorem 1. Section 4 briefly discusses the complexity of the realizability problem for other types of nearest neighbor graphs. The last section gives some concluding remarks about proximity graphs and the logic engine approach.

2. Logic engines for the NAE3SAT problem

This section describes a buildable mechanical device we call a "logic engine".

2.1. Logic engine design

The (m,n) logic engine is designed to encode instances of NAE3SAT having m clauses and n variables. The basic engine is described below; in the following subsection we show how to modify the basic engine to encode a particular instance of NAE3SAT.

The essential features of an (m,n) logic engine (see Fig. 2) are as follows.

- The engine has a rigid *frame*, which supports a non-rotating *shaft*.
- To the shaft is mounted a nested sequence of n *armatures* A_j, $1 \leqslant j \leqslant n$. Each armature can rotate about the shaft, but its position on the shaft is fixed; it cannot slide back and forth along the shaft. The spacing between armatures is designed to ensure that the armatures can rotate independently of one another.
- Each armature A_j holds two tautly stretched chains a_j and a_j' of equal-length *links*. One stretches from one end of the armature to the shaft, the other stretches from the other end of the armature to the shaft. Each of the chains on the innermost armature A_1 holds m links. The chains of the remaining armatures are proportionately longer.
- The sides of the frame extend on either side of the shaft at least as far as the chains do.

In each chain held by an armature, the m links closest to the shaft are numbered $1, 2, \ldots, m$, where the link adjacent to the shaft is numbered 1.

Looking ahead to the connection between NAE3SAT and the logic engine, each armature A_j corresponds to a variable X_j in an instance of NAE3SAT. The chain a_j corresponds to the variable X_j, and the chain a'_j corresponds to its complement X'_j. Planar layouts of the logic engine (that is, configurations where all the armatures and chains lie in the same plane) correspond to truth assignments for NAE3SAT as follows. Each armature A_j can be in one of two positions: either a_j can be above the shaft with a'_j below the shaft (corresponding to $X_j = 1$ and $X'_j = 0$); or a'_j can be above the shaft with a_j below the shaft (corresponding to $X'_j = 1$ and $X_j = 0$).

Note that when the engine and its armatures lie flat, the links in the chains line up to form rows. A clause c_k corresponds to the kth rows out from the shaft on either side. Clearly the set of links associated with c_k does not depend on how the armatures are rotated about the shaft.

The basic logic engine may be modified by attaching *flags* to the links $1, 2, \ldots, m$ in chains a_j and a'_j for $1 \leqslant j \leqslant m$. The next section shows how to attach flags to specific links to obtain encodings of instances of NAE3SAT.

Each flag can rotate freely about the chain. The flag thus has two possible positions when the logic engine is placed in the plane: it can point toward the front, or it can be "flipped" to point toward the rear. However, the flags are designed so that when the logic engine is placed in the plane, collisions involving flags occur under the following conditions.

- Two flags that lie in the same row and that are attached to chains of adjacent armatures collide with each other if and only if they are flipped so that they point toward each other.
- Any flag attached to the chain of the outermost armature A_n collides with the frame if it points toward the front edge of the frame.
- Any flag attached to the chain of the innermost armature A_1 collides with that armature if it points toward it.

The process of encoding an instance of NAE3SAT involves attaching flags to links according to the clause-literal incidence relation. Then an attempt is made to flip the armatures and the flags to produce a collision-free planar configuration of the logic engine and its remaining flags. In the next subsection, we show that an instance of NAE3SAT is a "yes" instance if and only if one or more such planar collision-free configurations exist.

Firstly, however, we note a purely geometric property. Consider a planar configuration of an (m, n) logic engine and its armatures. Each armature A_j is configured independently of the others, with the chain a_j corresponding to X_j positioned either above or below the shaft. Hold this configuration of armatures fixed for the moment, and consider any of the first m rows of links above or below the shaft. We allow the flags in this row to be turned in either direction, independently of one another. (In particular, they need not all point in the same direction.) If all n links in the row are flagged, then it is clearly impossible to avoid a collision in this row. On the other hand, if one or more of the links in the row is unflagged, then collisions in that row may be avoided simply by directing all the flags toward any one of the unflagged links. This leads to the following observation.

Observation 1. *A given planar configuration of a logic engine and its armatures has a collision avoiding placement of its flags if and only if each row contains at least one unflagged link.*

2.2. Customizing logic engines

This subsection explains how to customize an (m,n) logic engine to encode a particular instance of NAE3SAT.

The instance of NAE3SAT is encoded as follows. For $1 \leqslant i \leqslant m$ we attach a flag to the ith link of every chain a_j and a'_j $(1 \leqslant j \leqslant n)$, except that

1. if variable X_j appears in clause c_i then the ith link of a_j is unflagged; and
2. if variable X'_j appears in clause c_i then the ith link of a'_j is unflagged.

Fig. 3 illustrates part of a logic engine with flags, customized to encode the following instance of NAE3SAT:

$$c_1 = \{X'_1, X_2, X'_3\}, \qquad c_2 = \{X_1, X_3, X_4\}, \qquad c_3 = \{X'_1, X_3, X'_4\}.$$

We claim that an instance of NAE3SAT is a "yes" instance if and only if the customized (m,n) logic engine has a planar configuration without collisions.

Lemma 1. *An instance of* NAE3SAT *is a "yes" instance if and only if the corresponding customized (m,n) logic engine has a collision-free planar layout.*

Proof. Suppose that we have a "yes" instance of NAE3SAT, and the truth assignment t gives at least one true and at least one false literal for each clause. The armatures may be rotated to simulate the truth assignment t as follows: if $t(X_j) = 1$, then place a_j at the top and a'_j at the bottom; if $t(X_j) = 0$, then place a'_j at the top and a_j at the bottom. With this layout of armatures, since each clause c_i contains at least one literal Y with $t(Y) = 1$ and at least one literal Z with $t(Z) = 0$, there is at least one unflagged link in each horizontal row of links; thus a collision-free layout is possible.

On the other hand, suppose that we have a collision-free planar layout of the customized logic engine; then there is at least one unflagged link in each row. Thus there is at least one true and at least one false literal in each clause. □

Fig. 3 shows a collision-free planar layout of an encoded logic engine. Note that the armature A_1 and its chains have been rotated around the shaft, and the flags have been flipped so that they do not collide. This layout corresponds to a truth assignment that has at least one true literal and at least one false literal in each clause.

Suppose that we regard two collision-free planar configurations as *equivalent* provided that their armatures are oriented in the same way. Then there is a clear

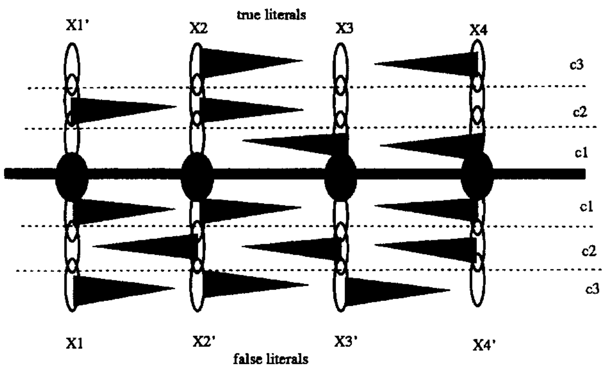

Fig. 3. Part of a (3,4) logic engine, customized with flags.

one–one correspondence between not-all-equal truth assignments and equivalence classes of collision-free planar configurations of the customized logic engine.

3. The simulation of a logic engine by a graph

For this section we limit our attention to *mutual* nearest neighbor graphs and abbreviate our terminology accordingly: we say that a graph is *realized* by a set P of points if G is isomorphic to the mutual nearest neighbor graph on P, and G is *realizable* if it is realizable as a mutual nearest neighbor graph.

This section presents a proof of Theorem 1. For motivation we preview how the transformation of NAE3SAT to MNNGR will proceed. Given an instance of NAE3SAT with m clauses and n variables, we define an (m,n) logic *graph* G, which is an instance of MNNGR. We design the (m,n) logic graph to simulate an (m,n) logic engine before the addition of flags. After customization by attachment of subgraphs that simulate flags, the customized graph will have a realization if and only if the NAE3SAT instance is a “yes” instance.

The first subsection shows how to construct a basic (m,n) logic graph. Furthermore, it describes the ways in which the logic graph can be realized as a mutual nearest neighbor graph; these realizations correspond to the planar configurations of the logic engine.

The second subsection shows how the logic graph can be customized to simulate the logic engine and thus to simulate NAE3SAT.

3.1. Logic graph design

We begin with an elementary lemma which is easy to prove and is used throughout.

Lemma 2. *Suppose that G is a mutual nearest neighbor graph of some point set P and that G is connected. Then all edge segments of G have the same length.*

In view of Lemma 2, we can assume that the edge segments of a realization of a connected mutual nearest neighbor graph all have *unit* length.

We say that a graph G is *uniquely realizable* if all the sets that realize G are *equivalent*, that is, one can be obtained from the other by rotations, reflections, translations and changes of scale. The proof of NP-hardness depends on the unique realizability of certain graphs.

We will build a simulation of the logic engine with a large uniquely realizable graph; but we begin with a simple lemma which ensures the correctness of the technique of building larger uniquely realizable graphs from smaller uniquely realizable graphs.

Lemma 3. *Suppose that H is a graph isomorphic to a subgraph of graph G, and that H is uniquely realizable. Then in any realization of G, the drawings of the subgraph corresponding to H must be equivalent to the unique realization of H.*

We can now describe the essential building blocks of the logic graph.

Definition 4. (a) Fig. 4(a) gives a realization of a *link graph* as a mutual nearest neighbor graph of its vertex-points.

(b) A link graph may be extended to a *flagged link graph* by the addition of three new vertices f, g, and h as shown in Fig. 4(b).

(c) A *chain* graph of length k is a sequence of k link graphs joined together as shown in Fig. 4(c).

From Lemma 2 we can deduce the following.

Lemma 4. *Both the link graph and the flagged link graph are uniquely realizable.*

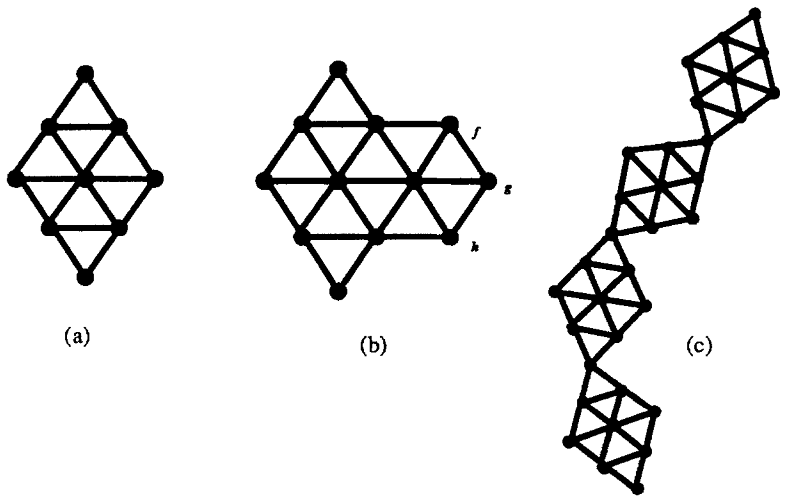

Fig. 4. (a) Link graph, (b) flagged link graph, (c) chain.

Note that a chain graph of length greater than one is *not* uniquely realizable as a mutual nearest neighbor graph. While all edges of a realization have the same length by Lemma 2, the angle between edges belonging to consecutive link graphs of the chain but sharing a common endpoint can vary from just over $\pi/3$ to just under π, assuming that the edges are adjacent in the cyclic ordering of edges about the shared endpoint. In fact, this phenomenon explains the reason for the nested armatures in the logic engine/graph design. Their purpose is to hold chains taut while still permitting them to be on either side of the shaft. If the chains were not taut, then the customized logic graph defined below might be realizable for "no" instances of NAE3SAT.

From the basic building blocks above, we can build a *logic graph.*

Definition 5. An (m,n) *logic graph* consists of a frame to which armatures with chains are attached as in Fig. 5.

As Fig. 5 shows, the (m,n) logic graph is realizable.

Lemma 5. *The* (m,n) *logic graph is uniquely realizable.*

Proof. One can show that the frame is uniquely realizable using Lemmas 2 and 3. To prove uniqueness of the realization of the shaft, we need to show that the shaft is taut. The Euclidean distance between the endpoints of an edge on the shaft is precisely 1 by Lemma 2. Thus the maximum Euclidean distance between the extremal endpoints of the shaft is equal to the number of edges in the shaft, and this can be achieved only if the vertices are stretched out along a line. Thus the unique realizability of the frame forces the shaft to be drawn as a straight line as shown.

It is not difficult to extend this argument to show uniqueness of the armatures and the attached chains, and then to the whole logic graph. □

The set of points in the plane occupied by the vertices in any realization is unique up to rotations, translations, reflections and changes of scale. However, the *labeled* graph has many realizations. These are described as follows.

- Each armature can be turned about the shaft so that either side of the armature can be placed in the closed region determined by the shaft and the frame.
- On each chain, each link except the center link can be turned, independently of the other links, so that either one of the two degree 3 vertices of the link lies in the closed region defined by that particular armature and its chain.

Thus the realizations of the logic graph simulate the planar layouts of the logic engine.

This completes the definition of basic logic graphs.

3.2. Customizing logic graphs

The customization of logic graphs to simulate NAE3SAT follows the customization of logic engines presented in Section 2.2. A logic graph customized with flags for the same example as in Section 2.2 is in Fig. 6. Note that a "collision" occurs between

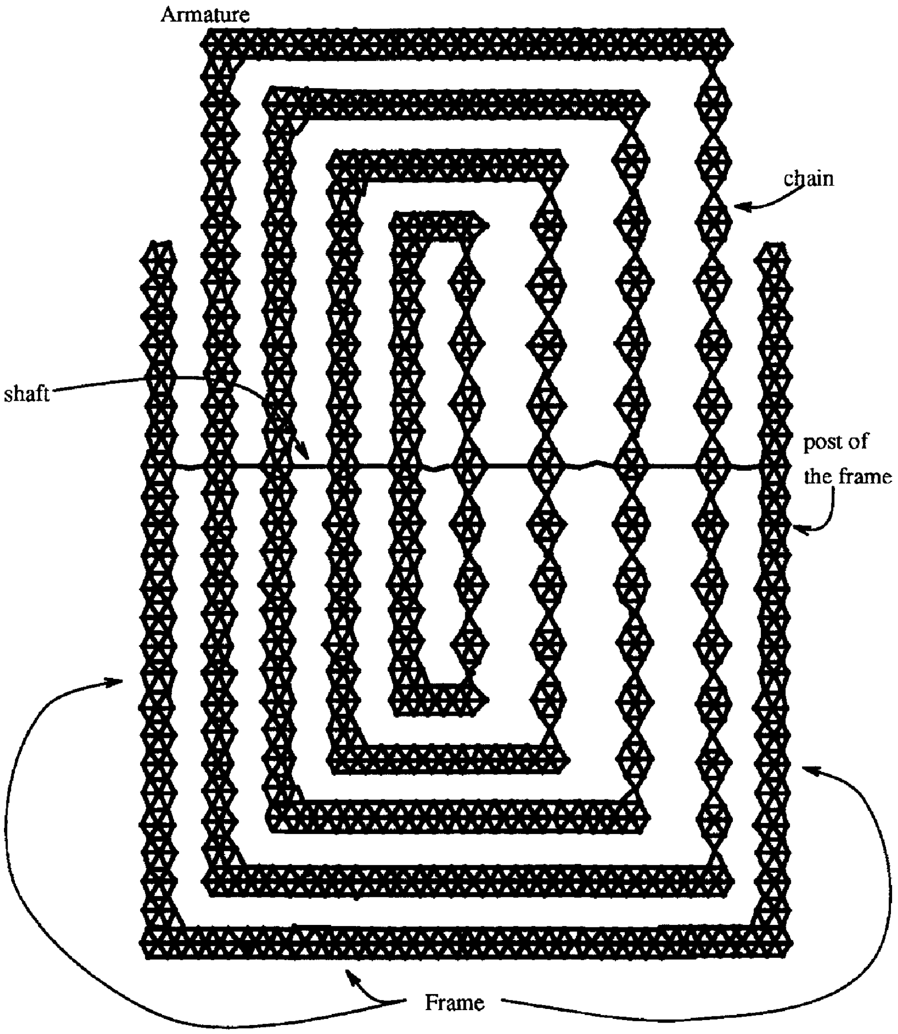

Fig. 5. An (m, n) logic graph.

two flags whenever a vertex of one flag is placed within a distance of one unit from a vertex of the other flag.

Lemma 6. *There is a polynomial time transformation from* NAE3SAT *to* MNNGR.

Proof. It is clear that the construction of the customized logic graph as described in the previous subsection can be done in time proportional to the size of the NAE3SAT instance.

The proof that the customized logic graph is realizable if and only if the NAE3SAT instance is a "yes" instance follows the same argument as in Section 2.2. □

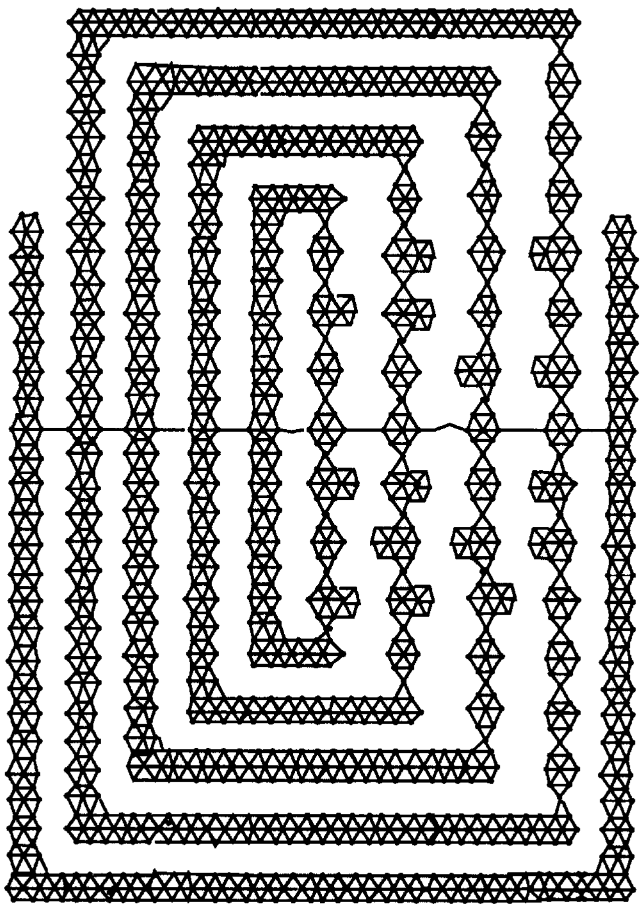

Fig. 6. An (m,n) logic graph customized for an instance of NAE3SAT.

Theorem 1 follows immediately from Lemma 6 and the NP-completeness of NAE3SAT [9].

4. Other types of nearest neighbor graphs

In this section we briefly indicate techniques for proving that the realization problems for the strong and weak nearest neighbor graphs are also NP-hard.

Strong Nearest Neighbor Graph Realization (SNNGR)
Instance: A directed graph G.
Question: Is G realizable as a strong nearest neighbor graph? That is, is there a set P of points in the plane such that the strong nearest neighbor graph Γ on P is isomorphic to G?

Weak Nearest Neighbor Graph Realization (WNNGR)
Instance: A directed graph G.
Question: Is G realizable as a weak nearest neighbor graph? That is, is there a set P of points in the plane and a weak nearest neighbor graph Γ on P such that G is isomorphic to Γ?

Theorem 2. SNNGR *and* WNNGR *are* NP-*hard.*

Proof. First consider SNNGR. We consider the undirected logic graph constructed in Section 3 as a directed graph in which each arc is oriented in both directions. This is realizable as a strong nearest neighbor graph if and only if it is realizable as a mutual nearest neighbor graph. Thus this transformation proves that SNNGR is NP-hard.

Next we consider WNNGR. The NP-hardness of realizing Euclidean minimum spanning trees is established in [7] using a simulation of a logic engine by a tree called a *logipede*. This graph simulates a logic engine in a way similar to the logic graph introduced in Section 3. An example of a logipede is shown in Fig. 7. Here l_1, l_2, l_3, l_4 represent literals and c_1, c_2, c_3 represent clauses.

To show that WNNGR is NP-hard, one can use essentially the same transformation. From an instance of NAE3SAT we can construct a logipede. Then we must give a direction to each edge, since weak nearest neighbor graphs are directed. We choose a leaf of the logipede, and orient the edge incident with that leaf in both directions.

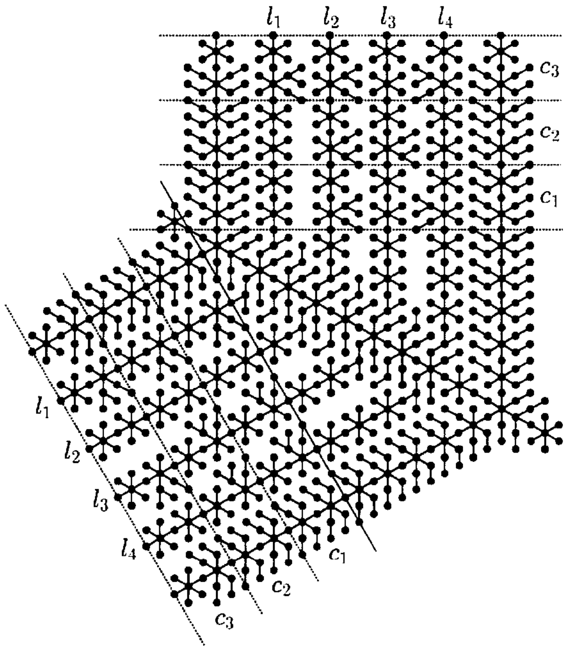

Fig. 7. The logipede.

Then we orient every other edge in the logipede toward the chosen leaf. This ensures that the outdegree of every vertex is precisely one. Further, every realization of the logipede as a minimum spanning tree is also a realization as a weak nearest neighbor graph. It follows that the transformation given in [7], followed by the orientation of edges described above, can be used to show that WNNGR is NP-hard. □

5. Concluding remarks

The logic engine approach provides a powerful method for investigating the complexity of layout problems; see [7, 11, 10, 3] for examples. As a final example, consider the following problem, investigated in [8].

Unit Planar Drawing (UPD)
Instance: A planar graph G.
Question: Is there a planar drawing of G in which each edge has length one?

Eades and Wormald show that UPD is NP-hard, using a complex reduction to a flow problem. Using the logic engine approach, a much simpler proof is possible: in fact, one can follow almost the same construction as in Section 3.

Polynomial time realization algorithms are available for some kinds of proximity graphs as long as the input is restricted; for example, linear time algorithms for drawing *trees* as relative neighborhood graphs are available (see [12] for a survey of such algorithms). To the best of the authors' knowledge, none of the common kinds of proximity graphs have polynomial time algorithms for general inputs. It would be interesting to know if all these problems are NP-hard.

References

[1] S. Bhatt and S. Cosmodakis, The complexity of minimizing wire lengths in VLSI layouts, *Inform. Process. Lett.* **25** (1987) 263–267.

[2] P. Bose, G. Di Battista, W. Lenhart and G. Liotta, Proximity constraints and representable trees, in: *Graph Drawing*, Lecture Notes in Computer Science, Vol. 894 (Springer, Berlin) 328–339.

[3] F.J. Brandenburg, Nice drawings of graphs and trees are computationally hard, TR MIP-8820, Univ. of Passau, 1988.

[4] G. Di Battista, P. Eades, R. Tamassia and I. Tollis, Algorithms for drawing graphs: an annotated bibliography, in: *Computational Geometry: Theory and Applications* **4** (1994) 235–282. Currently available from wilma.cs.brown.edu by ftp.

[5] G. Di Battista, W. Lenhart and G. Liotta, Proximity drawability: a survey, in: *Graph Drawing*, Lecture Notes in Computer Science, Vol. 894 (Springer, Berlin) 340–351.

[6] M. Dillencourt, Realizability of Delaunay triangulations, *Inform. Process. Lett.* **38** (1990) 283–287.

[7] P. Eades and S. Whitesides, The realization problem for Euclidean minimum spanning trees is NP-hard, *Algorithmica* **16** (1996) 60–82.

[8] P. Eades and N. Wormald, Fixed edge length graph drawing is NP-hard, *Discrete Appl. Math.* **28** (1990) 111–134.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*completeness* (W.H. Freeman, San Francisco, 1979).

[10] A. Gregori, Unit length embedding of binary trees on a square grid, *Inform. Process. Lett.* **31** (1989) 167–172.
[11] P.J. Idicula, Drawing trees in grids, M.Sc. Thesis, Computer Science, Univ. Auckland, 1990.
[12] G. Liotta, Computing proximity drawings of graphs, Ph.D. Thesis, Univ. of Rome "La Sapienza", 1995.
[13] A. Lubiw and N. Sleumer, Maximal outerplanar graphs are relative neighborhood graphs, in: *Proc. Canadian Conf. on Computational Geometry* (1993) 198–203.
[14] K. Sugiyama, A readability requirement on drawing digraphs: level assignment and edge removal for reducing the total length of lines, TR 45, International Institute for Advanced Study of Social Information Science (March 1984).
[15] G. Toussaint, A graph-theoretical primal sketch, in: G. Toussaint, ed., *Computational Morphology* (Elsevier, Amsterdam, 1988) 229–260.