ELSEVIER

Computational Geometry 9 (1998) 3–24

Computational Geometry
Theory and Applications

Unit disk graph recognition is NP-hard ☆

Heinz Breu [1], David G. Kirkpatrick *

Department of Computer Science, University of British Columbia, Vancouver, BC, Canada V6T 1Z2

Communicated by G. Di Battista and R. Tamassia; submitted 7 September 1994; revised 29 June 1995

Abstract

Unit disk graphs are the intersection graphs of unit diameter closed disks in the plane. This paper gives a polynomial-time reduction from SATISFIABILITY to the problem of recognizing unit disk graphs. Equivalently, it shows that determining if a graph has sphericity 2 or less, even if the graph is planar or is known to have sphericity at most 3, is NP-hard. We show how this reduction can be extended to 3 dimensions, thereby showing that unit sphere graph recognition, or determining if a graph has sphericity 3 or less, is also NP-hard. We conjecture that K-sphericity is NP-hard for all fixed K greater than 1. © 1998 Elsevier Science B.V.

Keywords: Disk intersection graphs; Disk touching graphs; NP-hard; Sphericity

1. Introduction

A *unit disk graph* is the intersection graph of a set of unit diameter closed disks in the plane. That is, each vertex corresponds to a disk in the plane, and two vertices are adjacent in the graph if the corresponding disks intersect. The set of disks is said to *realize* the graph. Of course, the unit of distance is not critical, since the disks realize the same graph even if the coordinate system is scaled by any convenient amount. Notice that two disks intersect if and only if the distance between their centers is at most the disk diameter. Unit disk graphs can therefore be realized just as well as a set of points in the plane; two vertices are adjacent in the graph exactly when the Euclidean distance between them is at most 1. A *realization* of a unit disk graph is therefore a mapping of the vertices to points which realize the graph in this way. Again, the *unit* of distance is not critical. This paper addresses the recognition problem for unit disk graphs: given a graph, determine if it has a realization.

Unit disk graphs have been used to model several physical problems, for example radio frequency assignment [8] and ship-to-ship communications (attributed to Marc Lipman by [16]). They have also

☆ Research supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

* Corresponding author. E-mail: kirk@cs.ubc.ca.

[1] On leave from Hewlett-Packard Laboratories. E-mail: breu@cs.ubc.ca.

PII S0925-7721(97)00014-X

been used as test cases for heuristic algorithms designed for arbitrary graphs [12]. More applications are described in [3,14].

There are several motivations for recognizing unit disk graphs. First, unit disk graphs are a natural generalization of unit interval graphs, also known as indifference graphs [15]. These graphs are interval graphs that do not contain $K_{1,3}$ as an induced subgraph [15]. They can therefore be recognized in polynomial time [1,6]. Secondly, unit disk graphs can admit more efficient algorithms than arbitrary graphs. For example, the NP-complete problem, maximum CLIQUE, can be solved in polynomial time for unit disk graphs [3]. There are also several familiar NP-complete problems whose corresponding optimization problems can be efficiently approximated for unit disk graphs [14]. Although these latter problems remain NP-complete for unit disk graphs, their approximating algorithms achieve better guarantees than any known for arbitrary graphs. Some of the algorithms for these problems require a realization of the graph. A third motivation is simply to determine if a graph is a unit disk graph, for example, to test the physical feasibility of a graph modeling molecules [9].

We show in this paper that the problem of recognizing unit disk graphs is NP-hard. This answers an open question mentioned in [3,14].

Our result can be interpreted in several related contexts. The first of these concerns what is known as the *sphericity* of a graph G – the smallest dimension d for which G is the intersection graph of a set of d-dimensional unit spheres [5,9]. Unit disk graphs are precisely the graphs of sphericity at most 2, and our NP-hardness result implies that determining the sphericity of a graph is also NP-hard. (Note that graphs of sphericity at most 1 are the unit interval graphs, and hence graphs of sphericity 1 can be recognized in polynomial time.) In the conclusion to this paper, we argue that our proof can be easily modified to show that the recognition of graphs with sphericity at most 3 is also NP-hard. We also conjecture that the same is true for graphs with sphericity at most K, for any $K > 1$.

A natural restriction of the class of unit disk intersection graphs studied here arises when we add the constraint that all disks have disjoint interiors. The resulting *disk touching graphs* (also called *coin graphs* [17]) are just the (necessarily planar) graphs induced by packings of disks in the plane. The results of this paper extend without difficulty to unit disk touching graphs. Indeed several of our constructions that exploit properties of disk packings become simpler when interior disjointness can be assumed.

It is interesting to ask how important the unit constraint is with respect to our recognition results for unit disk intersection graphs and unit disk touching graphs. If there is no constraint on the ratio of the largest to smallest disk, then the resulting class of (unconstrained) disk touching graphs has a polynomial time recognition algorithm by virtue of the fact[2] that this class coincides with the class of planar graphs. On the other hand, the recognition of (unconstrained) disk intersection graphs remains NP-hard [13]. We have recently generalized the reduction of the present paper to show that the recognition problem for intersection (or touching) graphs of disks whose radii are confined to any fixed range, is NP-hard [2].

2. 2-SPHERICITY is NP-hard

A *K-dimensional realization* of a graph $G = (V, E)$ is a function $f : V \to \mathbb{R}^K$ such that $(v_i, v_j) \in E$ if and only if $d(f(v_i), f(v_j)) \leqslant 1$ where d is the Euclidean distance between two points.

[2] This result is frequently attributed to W.P. Thurston, but was evidently first discovered by P. Koebe (cf. [17]).

K-SPHERICITY
Instance: Graph $G = (V, E)$, positive integer K.
Question: Does G have sphericity at most K, that is, does it have a K-dimensional realization?

This paper reduces SATISFIABILITY to 2-SPHERICITY. SATISFIABILITY is a common basis for NP-completeness proofs [7] and is the first problem to have been proved NP-complete [4]. Let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of Boolean *variables*. A clause $c = \{l_1, l_2, \ldots, l_k\}$ is a set of literals, which may be negated, e.g., $\overline{u}_i$, or unnegated, e.g., u_i, variables. A *truth assignment* is a function $t : U \to \{\text{TRUE}, \text{FALSE}\}$. In terms of literals, u_i is TRUE if and only if $t(u_i) = \text{TRUE}$, and $\overline{u}_i$ is TRUE if and only if $t(u_i) = \text{FALSE}$. A clause is *satisfied* by t if at least one $l_i \in c$ is TRUE. Finally, a *satisfying truth assignment* is one which simultaneously satisfies all the clauses.

SATISFIABILITY
Instance: A set $U = \{u_1, u_2, \ldots, u_m\}$ of Boolean variables and a set $C = \{c_1, c_2, \ldots, c_n\}$ of clauses over U.
Question: Is there a satisfying truth assignment for C?

Theorem 1. 2-SPHERICITY *is NP-hard.*

Proof. Given an instance C of SATISFIABILITY, we will construct a graph $G_C = (V_C, E_C)$ such that G_C has a realization[3] if and only if C is satisfiable. We will assume without loss of generality (see comments for SATISFIABILITY in [7]) that each clause in C contains *at most* three literals ($|c_i| \leqslant 3$) and that each variable appears in *at most* 3 clauses. Note that this is not the same as 3SAT.

This paper builds G_C in several stages. First, Section 3 constructs a graph G_C^{SAT} that corresponds closely to the instance C of SATISFIABILITY. We define a notion of orientability for this graph, and prove that it is orientable if and only if C is satisfiable (Lemma 2). Section 4 considers a canonical drawing of this graph on the grid. We define a notion of orientability for this drawing, and prove that it is orientable if and only if the underlying graph is orientable (Lemma 3). Finally, Section 5 forms G_C by simulating components of the grid drawing. Lemma 8 shows that G_C has a realization if and only if the underlying grid drawing is orientable. We finish by showing that the entire reduction can be executed in polynomial time. □

3. A graph that simulates SATISFIABILITY

Begin by constructing a graph G_C^{SAT} from the instance C of SATISFIABILITY. The vertices of the graph correspond to the clauses, variables, and negated variables of the SATISFIABILITY instance C. There is an edge between a literal vertex and a clause vertex if the literal appears in the clause. More formally, $G_C^{\text{SAT}} = (V_C^{\text{SAT}}, E_C^{\text{SAT}})$, where

$$V_C^{\text{SAT}} = \{\widetilde{c}\colon\ c \in C\} \cup \{u^+\colon\ u \in U\} \cup \{u^-\colon\ u \in U\},$$
$$E_C^{\text{SAT}} = \{(\widetilde{c}, u^+)\colon\ c \in C,\ u \in c\} \cup \{(\widetilde{c}, u^-)\colon\ c \in C,\ \overline{u} \in c\}.$$

[3] For brevity, and unless stated otherwise, the remainder of this paper abbreviates "two-dimensional realization" as "realization".

This graph models the testing of a truth assignment for SATISFIABILITY. Say that the graph is *orientable* if its edges can be directed such that, for each clause vertex, outdegree($\widetilde{c}$) $\geqslant 1$ and, for each pair of literal vertices, indegree(u^+) $= 0$ or indegree(u^-) $= 0$.

Intuitively, an edge directed from $\widetilde{c}$ to u^+ (respectively u^-) means that clause c has selected literal u (respectively $\overline{u}$) to satisfy it. That is, it requests that $t(u) =$ TRUE (respectively $t(u) =$ FALSE). If C is satisfiable, it must be possible to orient G_C^{SAT} since every clause must be satisfied (outdegree($\widetilde{c}$) $\geqslant 1$), and no truth assignment can set both a literal and its complement TRUE (indegree(u^+) $= 0$ or indegree(u^-) $= 0$). Conversely, if G_C^{SAT} is orientable, there must be a satisfying truth assignment for C since, for every pair of literal vertices indegree(u^+) $= 0$ or indegree(u^-) $= 0$ (each variable can be set either TRUE or FALSE) and, for every clause vertex $\widetilde{c}$, outdegree($\widetilde{c}$) $\geqslant 1$ (every clause is satisfied). This proves the following lemma.

Lemma 2. *C is satisfiable if and only if G_C^{SAT} is orientable.*

4. Drawing the graph on the grid

We draw the graph G_C^{SAT} on the grid as shown in Fig. 1. Each of the $(6|U|+1) \times (3|C|+2)$ grid vertices in this drawing is either unused, or is associated with a unique *component* of the drawing. Each component is enclosed by a unit square centered on its grid vertex.

The drawing is made up of three groups of components: communication components, literals and clauses. There are, in turn, three groups of communication components: wires, corners and cross-overs. A *wire* is a unit length line segment passing through a grid vertex, a *corner* is two half-length line segments meeting at right angles at a grid vertex, and a *cross-over* is two unit length line segments crossing at right angles on a grid vertex. There are therefore two types of wire components (horizontal and vertical), four types of corners, and one type of cross-over.

Each component in the drawing has one to four *terminals*. Each terminal terminates a line segment and is centered on a side of the unit square enclosing the component. The terminal on the top (or north) side of the unit square is called the T terminal. Similarly, the terminals on the bottom, left and right are called the B, L and R terminals, respectively. Wires and corners have two terminals each. Cross-overs have four terminals, and literals and clauses have up to three terminals each, equal to their degree in G_C^{SAT}. Two components in the drawing are *adjacent* if they have coincident terminals. We consider a complementary pair of literals to be a single truth setting component, and so do not show terminals between them. Fig. 1 depicts the set of all terminals as small circles.

An *orientation* of a terminal is a *direction*, N, S, E or W. A terminal T (respectively B, L, R) is *directed away* from its component if it is oriented N (respectively S, W, E) and is *directed towards* its component otherwise. Say that a grid drawing is *orientable* if all terminals can be oriented subject to the condition that:

draw1: only terminals adjacent to vertical line segments are directed N or S,

draw2: only terminals adjacent to horizontal line segments are directed E or W,

draw3: every wire, corner, cross-over line segment and clause has at least one terminal directed away from it,

draw4: every truth setting component has a literal component with *all* terminals directed away from it.

Fig. 2 shows an orientation of a portion of Fig. 1.

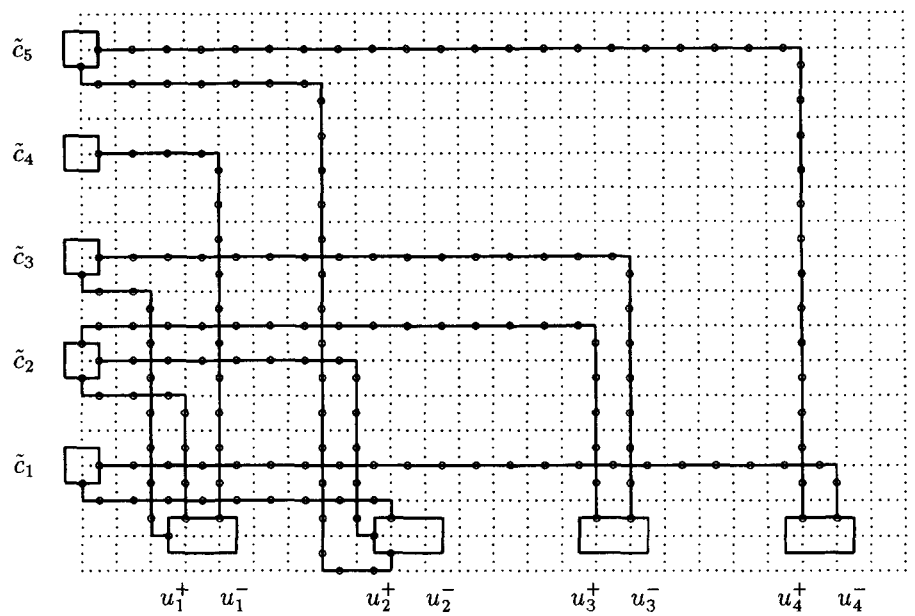


Fig. 1. The graph corresponding to the SATISFIABILITY instance $U = \{u_1, u_2, u_3, u_4\}$, $C = \{\{u_2, \overline{u}_4\}, \{u_1, u_2, u_3\}, \{u_1, \overline{u}_3\}, \{\overline{u}_1\}, \{u_2, u_4\}\}$ drawn on the grid. The clauses are drawn as squares. The literals are embedded as adjacent pairs (by variable) and so are drawn as rectangles. A literal component has up to three terminals: the three on the left of the variable box belong to the unnegated literal and the three on the right to the negated literal. Note that the area of the grid is $(6|U| + 1) \times (3|C| + 2)$.

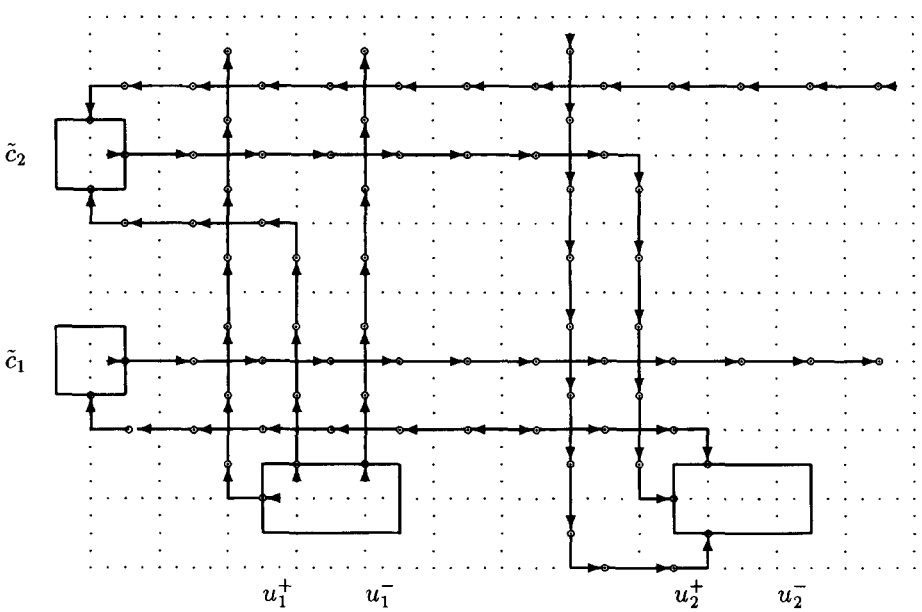


Fig. 2. An oriented grid drawing. The directions are drawn as arrows. Note that the path from u_2^+ to $\tilde{c}_1$ has a wire component that has both terminals directed away from it.

Lemma 3. *A grid drawing is orientable if and only if the underlying graph is orientable.*

Proof. Suppose we have an orientation for G_C^{SAT}. Then orient the terminals along each path in the grid drawing so that they are consistent with the orientation of the corresponding edge in E_C^{SAT}. This ensures that every wire, corner and cross-over line segment has *precisely* one terminal directed away from it. It also ensures that each clause has one terminal directed away from it, since the clause vertices have outdegree at least one. Finally, every truth setting component has a literal component with all terminals directed away from it since one of the literal vertices has zero indegree in G_C^{SAT}.

Now suppose that grid drawing has been oriented. Condition **draw3** ensures that any path in the drawing, corresponding to an edge in E_C^{SAT}, contains at most one wire, corner or cross-over segment with both terminals directed away from the component. One terminal of such a component can be redirected by reversing the direction of all terminals in the path from it to the literal. Doing so keeps all conditions satisfied since this operation does not change the orientation of any clause terminals, and it directs any literal terminals away from the literal, in keeping with condition **draw4**. Condition **draw3** then ensures that all terminals along the path are oriented consistently with some orientation for the corresponding edge. Under this orientation, all clause vertices have outdegree at least one, since the corresponding terminal is directed away from the clause. Furthermore, condition **draw4** ensures that $\mathrm{indegree}(u^+) = 0$ or $\mathrm{indegree}(u^-) = 0$. □

Corollary 4. *A grid drawing is orientable if and only if the underlying instance of* SATISFIABILITY *is satisfiable.*

5. Reduction from grid drawing orientability

We are now ready to construct G_C. To do so, we create a graph component for each grid drawing component: wires, corners, cross-overs, truth setters and clauses. Sections 5.2–5.6 provide the details. Each of these graph components has two or more *terminals* – labeled T, B, L or R – that correspond to the grid drawing terminals. Each terminal is an induced subgraph on four vertices, labeled a, b, c and d. To construct G_C, connect every pair of adjacent[4] components together by identifying the appropriate terminals, by label. Terminal T (respectively B, L, R) should be identified with an adjacent B (respectively T, R, L) terminal. The reader will find that the example presented in Section 5.7 clarifies this process.

5.1. Properties of cages

Before giving detailed descriptions of the various graph components, we must describe the building blocks from which they are constructed. The main building blocks are cycles, which we call "cages". Two cages are joined together at a shared edge to create larger components. The remaining building blocks are independent sets ("clusters") of one, two or three vertices ("beads"). Bead clusters are associated with the edge shared by two cages. In any realization, all the beads in a cluster are forced to lie in one of the two incident cages. To ensure this consistent embedding, beads are held in close

[4] Two graph components are *adjacent* if the associated grid drawing components are adjacent.

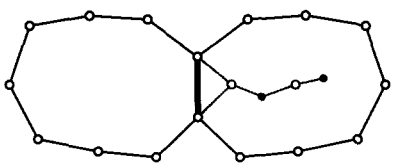


Fig. 3. Two adjacent cages with a shared 2-bead cluster.

proximity by additional "link" vertices. The latter also serve to attach the bead cluster to the endpoints of its associated cage edge (which we refer to as a "hinge" to reinforce the fact that its associated bead cluster can be embedded in the cage on either side). The link vertices serve no purpose other than to hold the bead cluster together, so we would like to keep them to a minimum. In fact, most cages in the following construction have only one bead and one link vertex, though some cages have as many as three beads. Since the beads themselves are independent, they must be realized as disjoint disks. This allows us to use cage capacity arguments to rule out certain embeddings. Fig. 3 illustrates two cages sharing an edge (bold) to which a cluster of two beads (dark circles) is attached by two link vertices (light circles). The cluster is shown embedded in the right cage, but, by reflection across the hinge, could also be embedded in the left cage.

The *capacity* of a cage is the maximum number of independent beads that can be embedded inside a cage in any realization. The capacity depends only on the number of cage vertices. A bead embedded inside a cage diminishes its remaining capacity. It thereby displaces other beads, which may otherwise have been embedded inside the cage. The following construction ensures that a bead can only be embedded in one of two adjacent cages. A displaced bead therefore displaces other beads from whatever cage it is embedded in. This is the basic mechanism for propagating information in a realization of G_C.

The corollary to the following lemma shows that the notion of a vertex being embedded inside the realization of a cycle is well defined.

Lemma 5. *Let f be a realization of a unit disk graph $G = (V, E)$. Let (v_a, v_b) and (v_c, v_d) be edges in E with distinct endpoints, and denote $f(v_i) = i$ for $i \in \{a, b, c, d\}$. If the line segments (a, b) and (c, d) cross, then the subgraph induced by $\{v_a, v_b, v_c, v_d\}$ contains a triangle.*

Proof. Suppose that (a, b) and (c, d) cross. Consider the quadrangle $acbd$. One of its interior angles is at least $\pi/2$. The lengths of the corresponding sides are thus smaller than the diagonal, and the lemma follows. □

Corollary 6. *A realization of a vertex induced cycle is a plane graph.*

Define an *n-cage* as a cage with a capacity for exactly n beads. G_C requires 0-cages, 1-cages, 2-cages and 3-cages. These are cycles with 3 to 6 vertices, 7 or 8 vertices, 9 vertices and 10 vertices, respectively. The capacities of these cages can be verified by considering the optimal disk packings shown in Fig. 4. In these packings, all unit disks are adjacent, and the interior beads are themselves optimally packed. These cases show that a 6-vertex cage *cannot* contain a bead, an 8-vertex cage cannot contain two beads, a 9-vertex cage cannot contain three beads and a 10-vertex cage cannot contain four beads. Recall that all containments must be strict.

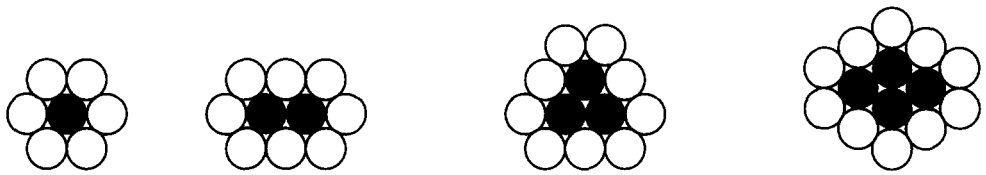


Fig. 4. Optimal packings limiting a 0-cage, 1-cage, 2-cage and 3-cage.

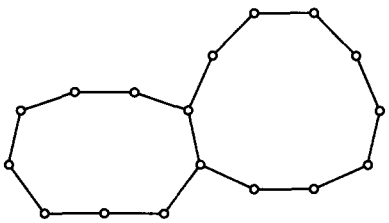


Fig. 5. How cages are hooked together.

The skeleton of the graph under construction will be composed of many cages "hooked together". The construction hooks two cages together by identifying two adjacent vertices from one cage with two adjacent vertices from the other cage; these adjacent vertices form a hinge. See Fig. 5. Lemma 5 guarantees that cages do not cross or overlap each other in any realization. In addition, cages are kept from containing each other by their capacities. This connection strategy therefore ensures that, in any realization, the clockwise order of the edges about every cage is determined by the order of the edges about *any* cage in the connected graph. It is therefore useful to think of the embedding of the skeleton of the graph, that is, the cages without link or bead vertices, to be invariant under all realizations. Different realizations simply allow beads (and links) to "flip" from one cage into another. The component definitions which follow will clarify this construction.

The properties described above allow realizations of unit disk graphs to emulate SATISFIABILITY decisions, depending on which cage encloses which cluster of beads. In addition, we must ensure that the skeleton of the graph, that is, the graph induced on the cages, is always realizable. In particular, the components must "fit together". For example, they must stretch between two grid vertices (from Fig. 1). To achieve this requirement, each component joins together several cages. For example, the forthcoming wire component consists of five cages. The number five comes from the realization scheme described in Section 5.7. This requirement also accounts for the number of vertices in a cage, where there is a choice. For example, a 0-cage can have 3 to 6 vertices; the number used by a particular component ensures realizability.

5.2. *Wires*

The graph in Fig. 6 implements a wire as a string of five 1-cages (with eight vertices each) connected as described in the previous subsection. The vertices in the wire's two terminals are labeled a_L, b_L, c_L and d_L, and a_R, b_R, c_R and d_R in Fig. 6. The vertical wire can be obtained from Fig. 6 by relabeling terminal R as terminal T, and terminal L as terminal B.

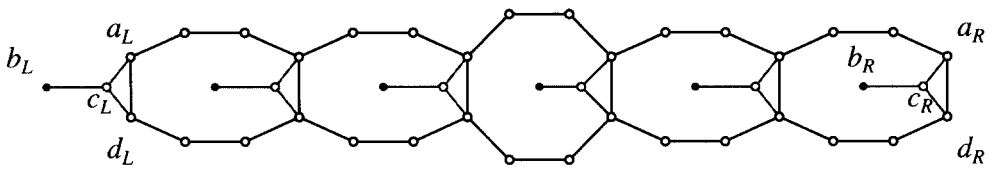


Fig. 6. The horizontal wire component: drawn with both terminals oriented W.

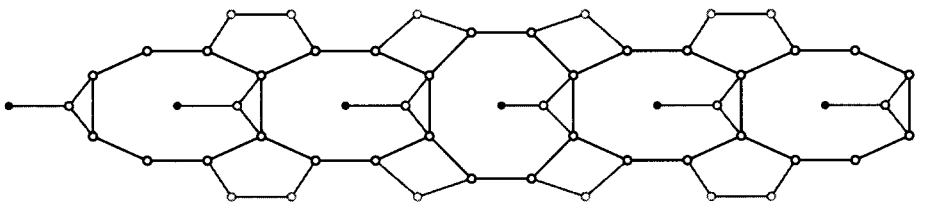


Fig. 7. Wire with mortar.

How do we know that the beads in the wire will be embedded in the cages of the wire in any realization? The answer is, that as things stand, we do not know. The solution adopted in this paper is to add mortar – extra vertices – around the hinge vertices as shown in Fig. 7. "Mortar" is a way of constraining a bead to certain cages for all realizations. The effect of the mortar is to surround each hinge vertex with small cages. The edges in the associated bead cluster cannot cross any of the added cage edges by Lemma 5 since the beads are independent of all cage vertices and the link vertices are independent of all but the two hinge vertices. Also, neither the bead nor the link vertices can be embedded inside any of the cages created by the mortar since all such cages are 0-cages. It is hard to see the underlying structure of the components if mortar vertices are shown. The remaining component diagrams, therefore, do not show any mortar vertices, but they should be understood to be present. Mortar is also required when two components are connected.

We say that the T (respectively B, L, R) terminal is *oriented* S (respectively N, E, W) if its bead (labeled b) is embedded inside its cage, and that it is oriented N (respectively S, W, E) otherwise. The properties of the cages and the mortar ensure that, if the T (respectively B, L, R) terminal is oriented S (respectively N, E, W), then its B (respectively T, R, L) terminal also is oriented S (respectively N, E, W). Note that it is also possible for the T (respectively L) terminal to be oriented N (respectively W) and the B (respectively R) terminal to be oriented S (respectively E) simultaneously. This ensures that at least one terminal is directed away from the wire.

5.3. Corners

As can be seen from Fig. 8, the graph for corners is also a string of five 1-cages. The corner's two terminals are labeled L and B in Fig. 8. The other three corners can be obtained from Fig. 8 by relabeling B as T (and exchanging label a_L with d_L), or L as R (and exchanging label a_B with d_B), or both. Again, the properties of cages ensure that at least one terminal is directed away from the corner.

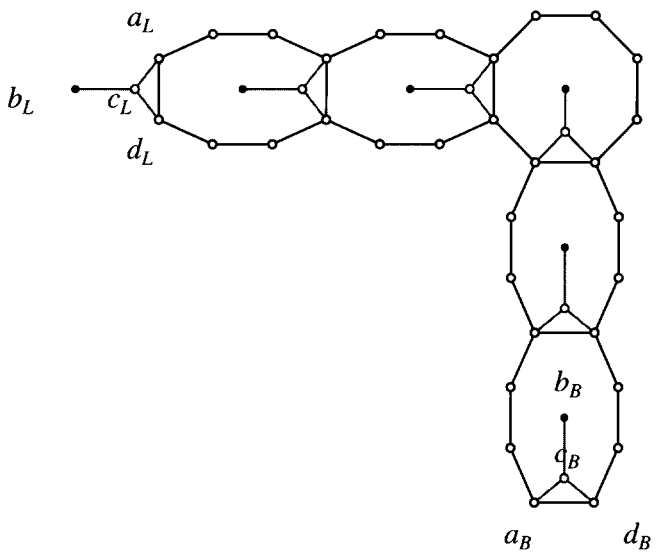


Fig. 8. The corner component: drawn with the L terminal oriented W and the B terminal oriented N.

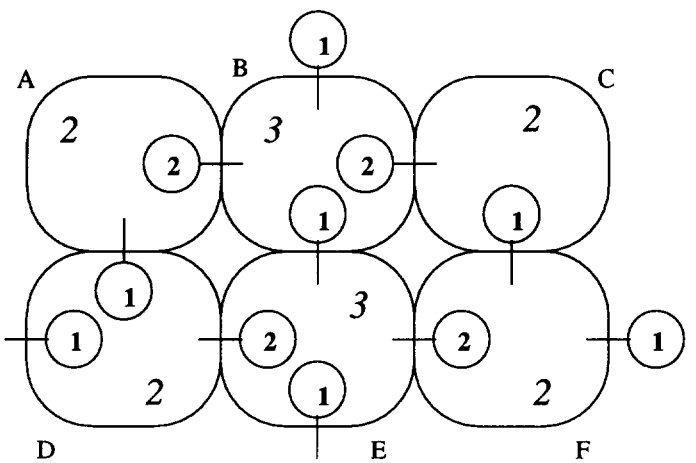


Fig. 9. Cross-over schematic.

5.4. Cross-overs

Of all components used in this reduction, the cross-over component is the most difficult to understand. The reader will find it easier to consider the cross-over schematic in Fig. 9 before looking at the complete component. In this schematic, cages are depicted as rings joined at hinges. Each cage is labeled with its capacity of independent beads. Bead clusters are depicted as circles, labeled by the cluster size, attached to a line crossing the associated hinge. Think of the bead clusters as flipping from one cage to another across their hinge.

The heart of the cross-over is the cycle of 3-cages and 2-cages. The following lemma is crucial to the operation of the cross-over component.

Lemma 7. *Each 3-cage in the cross-over component contains precisely one 2-bead cluster in any realization.*

Proof. The 3-cages cannot contain two 2-bead clusters since neither cage can accommodate four independent beads. Assume for the sake of contradiction that the 3-cage E does not contain either 2-bead cluster in some realization. Then the 2-cages D and F must each contain the 2-bead cluster that they share with cage E. Since D and F cannot contain any more than two independent beads, this forces cages A and C to contain the single beads that they share with cages D and F, respectively. But this means that A and C cannot contain the 2-bead clusters that they share with cage B. This forces cage B to contain them both. This is the contradiction that proves the lemma since cage B cannot contain 4 independent beads in any realization. A symmetric argument applies if we assume that cage B does not contain either 2-bead cluster. □

Let us now examine the cross-over in action. Assume the 1-bead on the bottom hinge (on cage E) is embedded in cage E. This 1-bead, together with whichever 2-bead cluster is in the cage by Lemma 7, fills cage E to capacity. The 1-bead shared between cage E and B must therefore lie in cage B. This 1-bead, together with whichever 2-bead cluster is in the cage, fills cage B to capacity. This forces the 1-bead at the top hinge (on cage B) out of the cage. A symmetric argument applies if the bead on the top hinge is embedded in cage B.

Assume now that the 1-bead on the left hinge (on cage D) is embedded inside cage D. It cannot occupy cage D together with the 2-bead cluster shared with cage E. Hence this 2-bead cluster must lie in cage E. The 2-bead cluster shared with cage F must therefore lie in cage F, since cage E cannot accommodate two 2-bead clusters. This 2-bead cluster forces the 1-bead on the right hinge (on cage F) out of the cage, and out of the cross-over. Incidentally, it also completes the feed-back cycle exploited in Lemma 7 as follows. It forces a 1-bead into cage C, which forces a 2-bead into cage B, which forces a 2-bead into cage A, which forces a 1-bead back into cage D. A symmetric argument applies if the 1-bead on the right hinge is embedded inside cage F.

Complete the cross-over component by adding a 1-cage "lead" to the left, top and right hinges. Also add a lead, consisting of a string of two 1-cages, to the bottom hinge. In the schematic, the 2-cages are shown directly connected to one another. In the actual component, each pair is connected with a 1-cage. The completed cross-over component is shown in Fig. 10. Although no mortar *vertices* are added, some mortaring by edges between cages is evident.

The cross-over's four terminals are labeled T, B, L and R in Fig. 10. The cross-over construction ensures that, if the T (respectively B, L, R) terminal is oriented S (respectively N, E, W), then the B (respectively T, R, L) terminal also is oriented S (respectively N, E, W). However, it is possible for the T and B terminals simultaneously to be oriented N and S, respectively. Similarly, it is possible for the L and R terminals simultaneously to be oriented W and E, respectively.

5.5. *Truth setters*

The truth setting component is shown in Fig. 11. Its heart is a pair of connected 3-cages. The cage on the left corresponds to the unnegated literal and the cage on the right to the negated literal. A cluster of three independent beads is connected by two link vertices to the common hinge of the 3-cages.

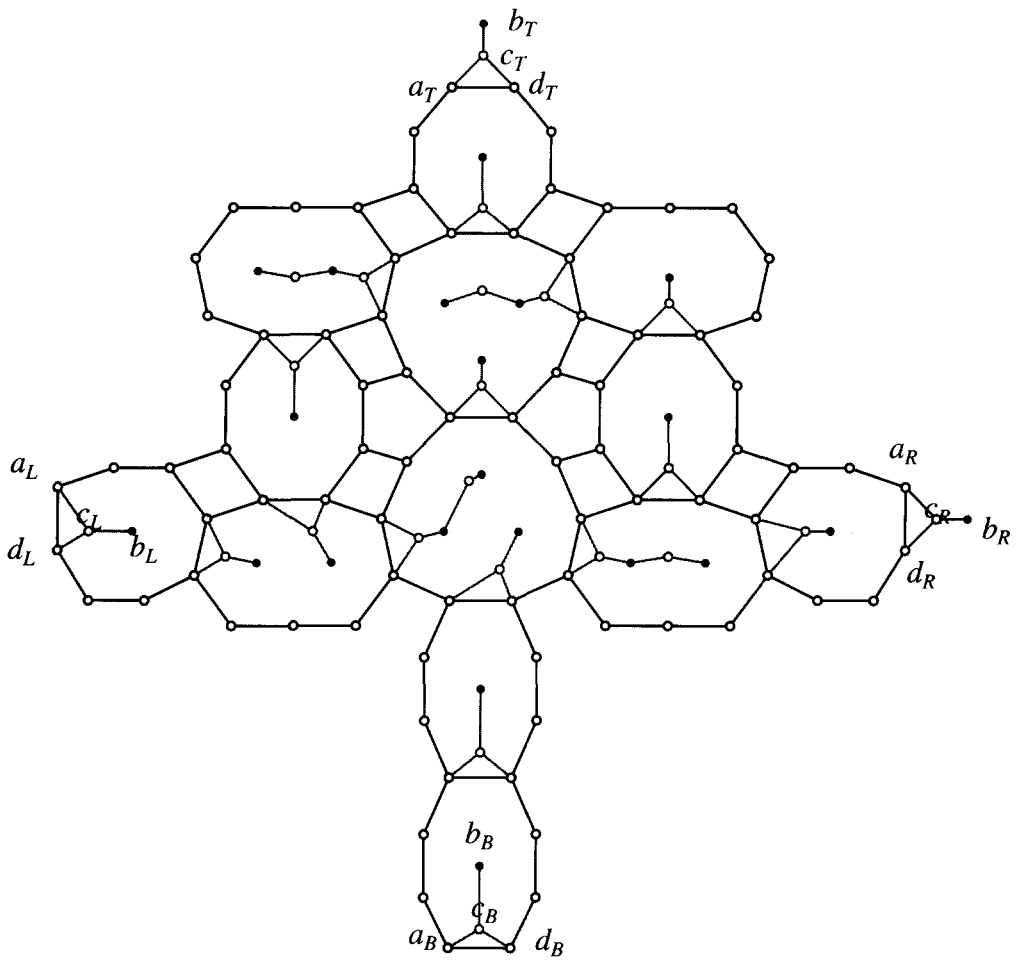


Fig. 10. The cross-over component: drawn with the T and B terminals oriented N, and the L and R terminals oriented E.

Since this cluster is in one of the two 3-cages in any realization, it displaces all incident single beads, ensuring that all associated terminals are oriented away from the 3-cage.

5.6. *Clauses*

The clause testing component is shown in Fig. 12. Its heart is a single 2-cage. Since this cage can contain at most two independent beads, it must be that at least one terminal is oriented away from it.

If a terminal is not used in the grid drawing, the corresponding graph component terminal is "capped" by appending a 0-cage to it. Fig. 13 shows a clause that has one terminal with cap and mortar (could not resist). This ensures that the terminal is oriented *towards* the clause, thereby forcing one of the remaining terminals to be oriented away by reducing the capacity of the 2-cage by one.

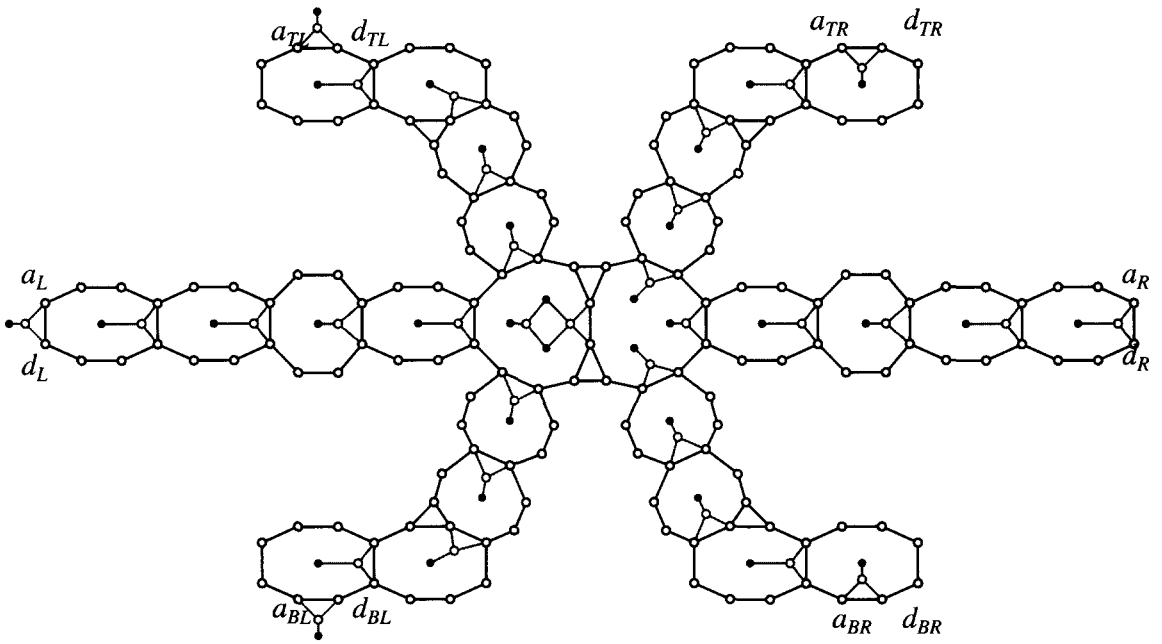


Fig. 11. The truth setting component: drawn with the 3-cluster in the 3-cage corresponding to the unnegated literal.

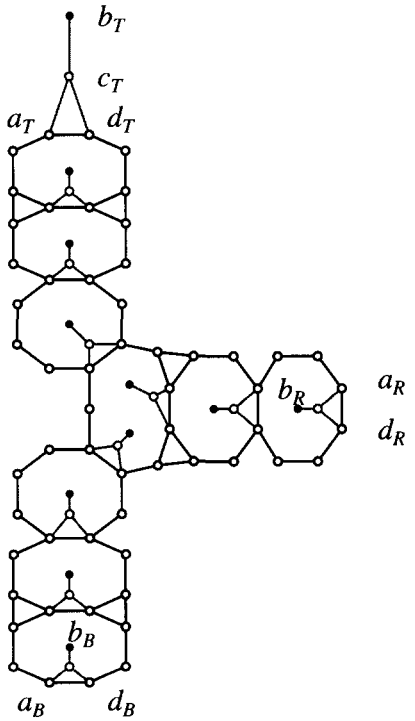


Fig. 12. Clause component: drawn with T and B terminal oriented N, and R terminal oriented W.

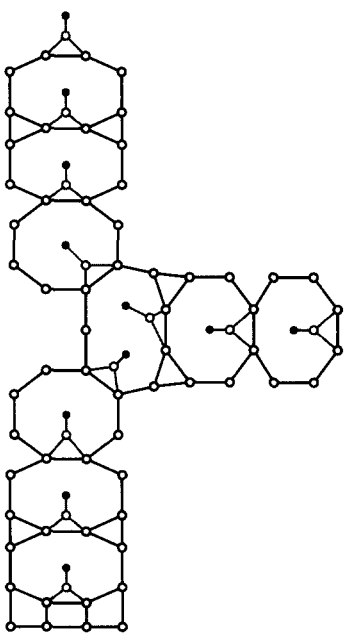


Fig. 13. Clause testing component with bottom terminal capped.

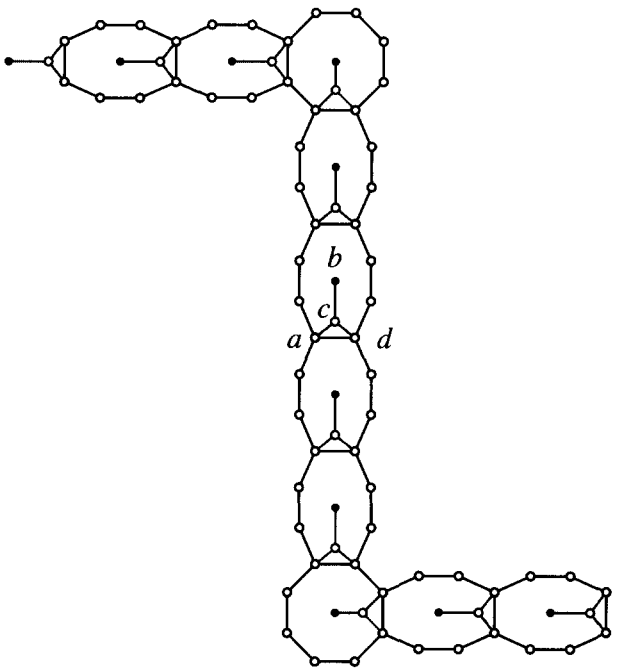


Fig. 14. Two connected corners. The T terminal of the lower corner has been identified with the B terminal of the upper corner.

5.7. *Building and realizing G_C*

Now that all components have been described, we can present an example showing how components are connected. Note that terminals always have the same structure: a bead vertex connected to two hinge vertices by a link vertex. Fig. 14 shows how two components are connected, two corners in this case.

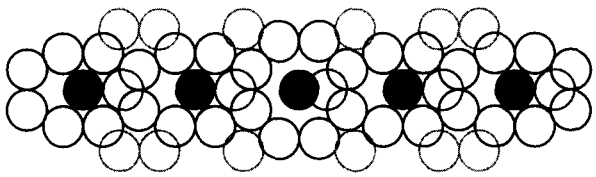
Recall that a literal component in the grid drawing has *at most* 3 terminals, whereas each literal of the truth setting component above has *exactly* 3 terminals. The unused truth setting terminals are not connected to the cages of any other component in G_C. In these cases, the incident terminal bead may simply be "absorbed by the ether", if the adjacent literal 3-cage is occupied in a realization.

Lemma 8. *The graph G_C has a realization if and only if the underlying grid drawing is orientable.*

Proof. Given a realization, orient the grid drawing terminals exactly as the realization terminals. The definition of orientation for realization terminals ensures that conditions **draw1** and **draw2** are met. The nature of the wire, corner, cross-over and clause components ensures that condition **draw3** is met. Finally, the nature of the truth setting component ensures condition **draw4** is met. That is, the grid drawing is orientable if G_C has a realization.

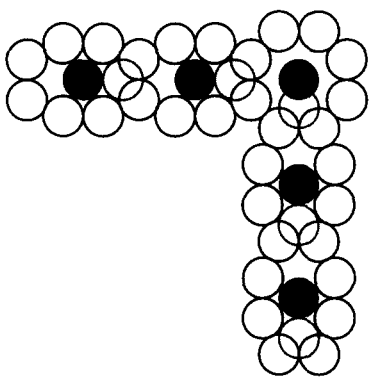
Now assume that the terminals of the grid drawing have been oriented. From this we will construct a realization for G_C. This realization has a unit of distance equal to 10. Equivalently, divide all coordinates by 10 for a unit of distance equal to 1. Let the lower left grid corner have x–y coordinates $(0,0)$.

Table 1
Wire (with mortar) coordinates. This wire is horizontal and has both terminals oriented W. By appropriate rotations we get representations also for other orientations



−68	−5		−68	5		−59	−9		−59	9		−54	0	bead
−49	−9		−49	9		−45	−15	mortar	−45	15	mortar	−44	0	link
−40	−5		−40	5		−35	−15	mortar	−35	15	mortar	−31	−9	
−31	9		−26	0	bead	−21	−9		−21	9		−16	0	link
−14	−15	mortar	−14	15	mortar	−12	−5		−12	5		−5	−12	
−5	12		0	0	bead	5	−12		5	12		7	0	link
12	−5		12	5		14	−15	mortar	14	15	mortar	21	−9	
21	9		26	0	bead	31	−9		31	9		35	−15	mortar
35	15	mortar	36	0	link	40	−5		40	5		45	−15	mortar
45	15	mortar	49	−9		49	9		54	0	bead	59	−9	
59	9		64	0	link	68	−5		68	5				

Table 2
Corner coordinates. This corner has the (0, 0) grid vertex in the northeast corner. The L terminal is oriented W and the B terminal is oriented N. The other orientations and three corners can be obtained by rotating these coordinates by 90 degrees and by reflecting about the coordinate axes



−68	−5		−68	5		−59	−9		−59	9		−54	0	bead
−49	−9		−49	9		−44	0	link	−40	−5		−40	5	
−31	−9		−31	9		−26	0	bead	−21	−9		−21	9	
−16	0	link	−12	−5		−12	5		−9	−59		−9	−49	
−9	−31		−9	−21		−5	−68		−5	−40		−5	−12	
−5	12		0	−64	link	0	−54	bead	0	−36	link	0	−26	bead
0	−7	link	0	0	bead	5	−68		5	−40		5	−12	
5	12		9	−59		9	−49		9	−31		9	−21	
12	−5		12	5										

The coordinates for each component realization are given in Tables 1–5, allowing the reader to verify all claims of adjacency and non-interference. To make this task somewhat easier, the points for each component are plotted as circles of radius 5. Note that two circles intersect exactly when the Euclidean distance between the corresponding points is less than or equal to 10. For each component in the grid drawing, construct the corresponding component realization, depending on its orientation.

Refer back to Fig. 1. Recall that every component in that figure is centered on a grid vertex and is enclosed by a (grid unit) square. The realization magnifies the grid unit by 136. The reader can verify this by noting that each component realization is contained in the square $[-68, 68] \times [-68, 68]$. The one exception is the truth setting component, Table 4, which is really two literal components, and therefore takes up two squares that constitute the rectangle $[-136, 136] \times [-68, 68]$. The "magic" number 136 is not crucial; it arose in our attempt to create a realization with small integer coordinates. Note, however, that even "small integer coordinates" are not necessary for an NP-hardness proof; we use them here to clarify the exposition. Therefore, to center a component on its logical grid coordinates, simply translate its realization to $(136x, 136y)$, where (x, y) are the grid coordinates of the component. The grid coordinates of a component are the coordinates of the corresponding grid vertex in all cases but one. Again, the one exception is the truth setting component, which is really two literal components. Referring back to Fig. 1, we can see that literal u_i^+ has coordinates $(3 + 6i, 1)$, and literal u_i^- has coordinates $(4 + 6i, 1)$, so that variable u_i has coordinates $(3.5 + 6i, 1)$. Therefore, the truth setting component for variable u_i needs to be translated to $(136x, 136y) = (136(3.5 + 6i), 136) = (476 + 816i, 136)$ again with small integer coordinates.

Table 3
Cross-over coordinates. The T and B terminals are oriented N, and the L and R terminals are oriented E. The other orientations can be obtained by keeping all cage and bead locations the same, but rearranging the link vertices as needed by symmetry

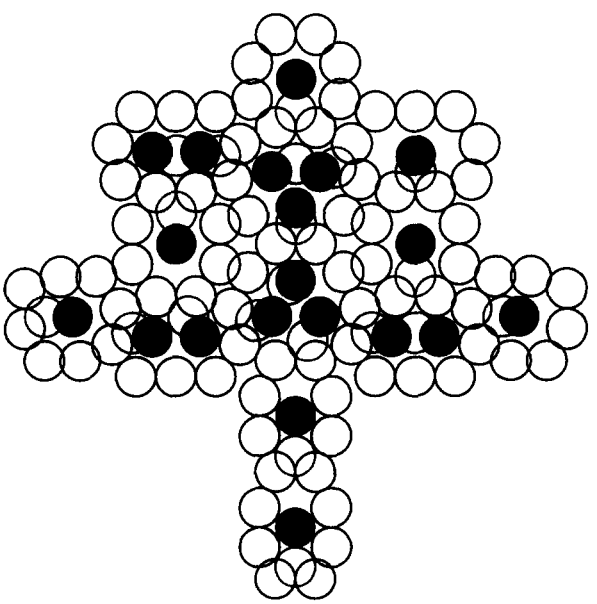


Table 3
(Continued)

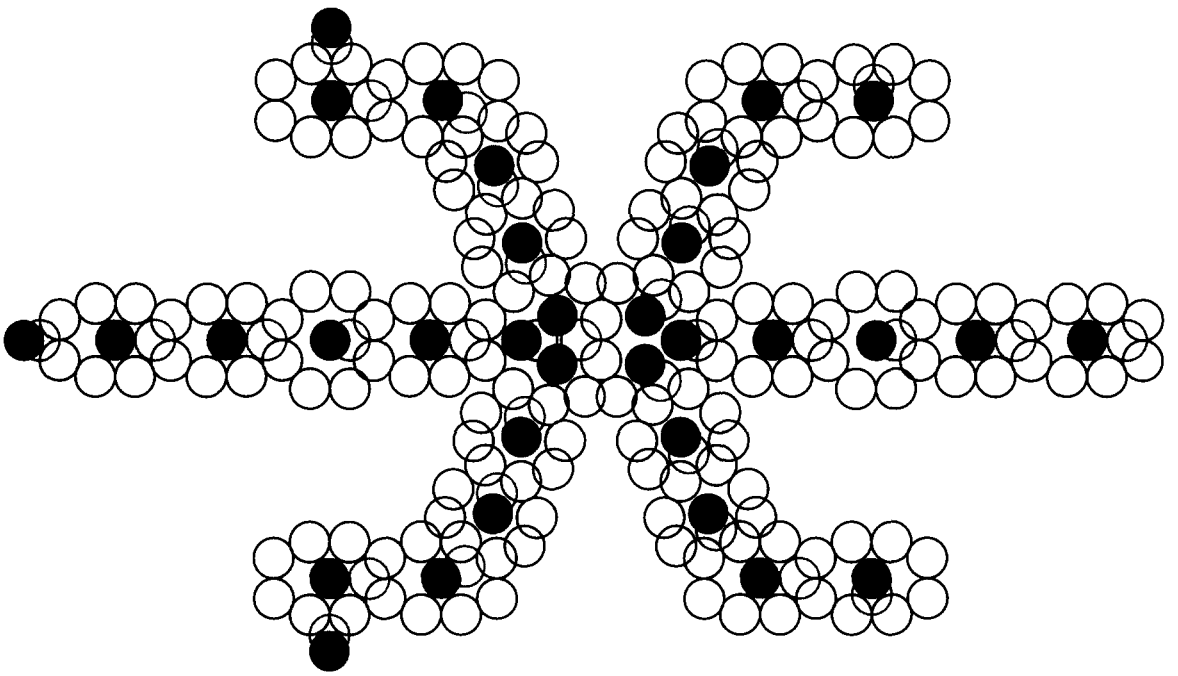
−68	−5		−68	5		−63	−13		−63	−2	link	−59	8	
−56	−2	bead	−54	−13		−50	8		−46	−9		−46	41	
−44	0		−44	32		−41	−6	link	−41	11		−41	21	
−40	−17		−40	49		−36	−7	bead	−36	39	bead	−35	3	
−35	29		−30	−17		−30	16	bead	−30	24	link	−30	38	link
−30	49		−27	−2	link	−25	3		−25	29		−24	−7	bead
−24	39	bead	−20	−17		−20	49		−19	11		−19	21	
−19	38	link	−16	0		−16	32		−14	−9		−14	41	
−12	9		−12	23		−11	52		−11	61		−10	−3	link
−9	−60		−9	−50		−9	−32		−9	−22		−6	−2	bead
−6	34	bead	−5	−68		−5	−41		−5	−13		−5	16	
−5	45		−5	68		−2	6	link	0	−65	link	0	−55	bead
0	−37	link	0	−27	bead	0	7	bead	0	21	link	0	25	bead
0	36	link	0	49	link	0	57	bead	3	−8	link	5	−68	
5	−41		5	−13		5	16		5	45		5	68	
6	−2	bead	6	34	bead	9	−60		9	−50		9	−32	
9	−22		10	35	link	11	52		11	61		12	9	
12	23		14	−9		14	41		16	0		16	32	
19	−6	link	19	11		19	21		20	−17		20	49	
24	−7	bead	25	3		25	29		30	−17		30	−6	link
30	8	link	30	16	bead	30	34	link	30	38	bead	30	49	
35	3		35	29		36	−7	bead	40	−17		40	49	
41	11		41	21		44	0		44	32		46	−9	
46	41		50	8		52	−2	link	54	−13		56	−2	bead
59	8		63	−13		68	−5		68	5				

This procedure results in duplicated points corresponding to terminal hinge vertices. Remove one set; they were only duplicated for clarity of exposition of each component. Note this procedure does *not* duplicate beads and links on terminals.

5.8. *The reduction can be implemented in polynomial time*

The grid drawing has area $(6|U|+1)\times(3|C|+2)$, and therefore at most this many components since a unit square encloses each component. Each component of G_C has a constant number of vertices and edges belonging to cages, links, beads and mortar. Finally, each connection of components requires

Table 4
Truth setter coordinates. This truth setting component realization has the 3-bead cluster inside the left 3-cage. Therefore, the three leftmost T, L and B terminals are oriented N, W and S, respectively. The other possible configurations can be obtained by symmetry: retain the coordinates of the cages and beads, but alter the link coordinates accordingly. This realization shows the leftmost T, L and B terminal beads being absorbed by the ether. In a complete realization, delete those external chain and bead vertices on terminals connected to wires or corners

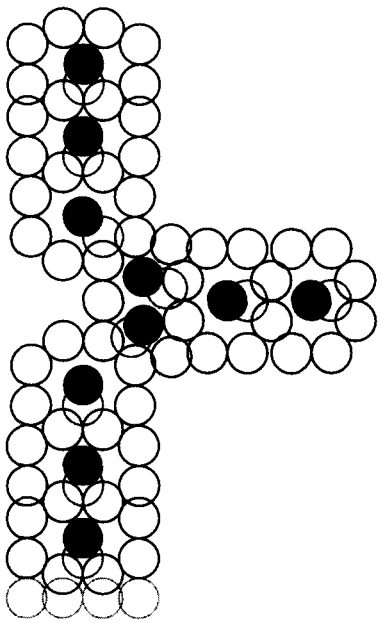


−145	0	bead	−141	0	link	−136	−5		−136	5		−127	−9	
−127	9		−122	0	bead	−117	−9		−117	9		−112	0	link
−108	−5		−108	5		−99	−9		−99	9		−94	0	bead
−89	−9		−89	9		−84	0	link	−82	−64		−82	−54	
−82	54		−82	64		−80	−5		−80	5		−73	−68	
−73	−50		−73	−12		−73	12		−73	50		−73	68	
−68	−77	bead	−68	−73	link	−68	−59	bead	−68	0	bead	−68	59	bead
−68	73	link	−68	77	bead	−63	−68		−63	−50		−63	−12	
−63	0	link	−63	12		−63	50		−63	68		−58	−59	link
−58	59	link	−57	−5		−57	5		−54	−64		−54	−54	
−54	54		−54	64		−48	−9		−48	9		−45	−68	
−45	−50		−45	50		−45	68		−43	0	bead	−40	−59	bead
−40	59	bead	−39	−44		−39	44		−38	−9		−38	9	
−37	−37		−37	37		−35	−68		−35	−50		−35	50	

Table 4
(Continued)

−35	68		−34	−56	link	−34	56	link	−33	0	link	−32	−25	
−32	25		−30	−18		−30	18		−29	−31		−29	−5	
−29	5		−29	31		−27	−43	bead	−27	43	bead	−26	−64	
−26	−54		−26	−38	link	−26	38	link	−26	54		−26	64	
−22	−12		−22	12		−20	−35		−20	−24	bead	−20	0	bead
−20	24	bead	−20	35		−19	−51		−19	−19	link	−19	19	link
−19	51		−16	−44		−16	0	link	−16	44		−13	−16	
−13	16		−12	−32		−12	32		−11	−6	bead	−11	6	bead
−9	−25		−9	25		−5	0	link	−4	−14		−4	14	
0	−5		0	5		4	−14		4	14		9	−25	
9	25		11	−6	bead	11	6	bead	12	−32		12	32	
13	−16		13	16		15	−10	link	15	10	link	16	−44	
16	44		19	−51		19	51		20	−35		20	−24	bead
20	0	bead	20	24	bead	20	35		22	−28	link	22	−12	
22	12		22	28	link	24	0	link	26	−64		26	−54	
26	54		26	64		27	−43	bead	27	43	bead	29	−47	link
29	−31		29	−5		29	5		29	31		29	47	link
30	−18		30	18		32	−25		32	25		35	−68	
35	−50		35	50		35	68		37	−37		37	37	
38	−9		38	9		39	−44		39	44		40	−59	bead
40	59	bead	43	0	bead	45	−68		45	−50		45	50	
45	68		48	−9		48	9		50	−59	link	50	59	link
53	0	link	54	−64		54	−54		54	54		54	64	
57	−5		57	5		63	−68		63	−50		63	50	
63	68		64	−12		64	12		68	−63	link	68	−59	bead
68	59	bead	68	63	link	69	0	bead	73	−68		73	−50	
73	50		73	68		74	−12		74	0	link	74	12	
80	−5		80	5		82	−64		82	−54		82	54	
82	64		89	−9		89	9		94	0	bead	99	−9	
99	9		104	0	link	108	−5		108	5		117	−9	
117	9		122	0	bead	127	−9		127	9		132	0	link
136	−5		136	5										

Table 5
Clause (with cap and mortar) coordinates. The Bottom terminal is capped in this realization. Both the Top and Bottom terminals are oriented N. The Right terminal is oriented W. Other caps and orientations can be obtained by symmetry. All cage and bead vertex locations are the same; only the link vertices need be rearranged



−14	−74	mortar	−14	−64		−14	−54		−14	−46		−14	−36	
−14	36		−14	46		−14	54		−14	64		−13	−26	
−13	−16		−13	16		−13	26		−5	−74	cap	−5	−68	
−5	−50		−5	−32		−5	−10		−5	10		−5	32	
−5	50		−5	68		0	−64	link	0	−59	bead	0	−46	link
0	−41	bead	0	−26	link	0	−21	bead	0	21	bead	0	36	link
0	41	bead	0	54	link	0	59	bead	5	−74	cap	5	−68	
5	−50		5	−32		5	−10		5	0		5	10	
5	16	link	5	32		5	50		5	68		12	−9	link
13	−26		13	−16		13	16		13	26		14	−74	mortar
14	−64		14	−54		14	−46		14	−36		14	36	
14	46		14	54		14	64		15	−6	bead	15	6	bead
21	3	link	22	−14		22	14		25	−5		25	5	
31	−13		31	13		36	0	bead	41	−13		41	0	link
41	13		47	−5		47	5		52	−13		52	13	
57	0	bead	62	−13		62	0	link	62	13		68	−5	
68	5													

only a constant number of additional mortar vertices. Since the size of the SATISFIABILITY instance is a polynomial function of $|U|$ and $|C|$, the entire recognition instance can be built in polynomial time.

6. Concluding remarks

This paper shows that unit disk graph recognition, or K-SPHERICITY for $K = 2$, is NP-hard by reducing SATISFIABILITY to it. What can be said about the complexity of K-SPHERICITY for other values of K? We mentioned in the Introduction that graphs with sphericity 1 are called unit interval graphs and can be recognized in polynomial time. That is, K-SPHERICITY is solvable in polynomial time for $K = 1$.

The complexity of 3-SPHERICITY is an open question due to Havel [9–11] arising from studies of molecular conformation. This problem, too, is NP-hard since our reduction for 2-SPHERICITY can be modified for 3-SPHERICITY. The basic building blocks are again beads in cages, but this time the cages are three-dimensional. Their capacities again would be deduced by optimal sphere packing arguments, but in three dimensions. Optimal packing is admittedly more complicated in higher dimensions, but we simply require that some such packings, and therefore constrained cages, exist. Note that the optimal packings need not be unique.

Embed the SATISFIABILITY graph in the three-dimensional grid as follows. Begin by embedding the clauses and variables in a two-dimensional grid, as before. Think of this as a horizontal layer, with $z = 1$. Now, add more horizontal layers to the grid for a total of $3|C|$ layers, so that each occurrence of a literal in a clause has its own layer. Route a literal to a clause by first routing from the literal to the corresponding layer, using the third dimension. Then route over to the clause's (x, y) coordinates on the dedicated grid layer. Conclude by routing to the clause using the third dimension. In this way, no wires interfere, and the construction does not even require cross-over components. We believe that an analogous construction is possible also in higher dimensions.

Conjecture 9. K-SPHERICITY is NP-hard for all fixed K greater than 1.

It is apparent from the nature of cages and beads that the assumption of at most 3 literals per clause, and that each variable appears in at most 3 clauses, is not really necessary to our reduction. Fan out and fan in components can be manufactured from the basic building blocks used in the construction. They are avoided here in order to clarify the exposition.

Note that an instance of 2-SPHERICITY constructed by the reduction in this paper is planar. This is true whether or not it has a realization, since it is always possible to embed all bead and link vertices inside their incident cage without crossing edges. That is, 2-SPHERICITY remains NP-hard even if the graph is planar.

Finally, an instance of 2-SPHERICITY constructed by our reduction has a 3-dimensional realization. To see this, embed the cages in the (horizontal) plane, as usual. Then embed the bead and link vertices directly above their incident hinges, using the third dimension. Since the hinges are independent of one another, so are the bead clusters. This implies that 2-SPHERICITY remains NP-hard even if the graph has sphericity at most 3.

References

[1] K.S. Booth, G.S. Luecker, Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms, J. Comput. System Sci. 13 (1976) 335–379.
[2] H. Breu, D.G. Kirkpatrick, On the complexity of recognizing intersection and touching graphs of disks, in: Proc. Graph Drawing '95 (Passau, 1995), Lecture Notes in Computer Science 1027, Springer, Berlin, 1996, pp. 88–98.
[3] B.N. Clark, C.J. Colbourn, D.S. Johnson, Unit disk graphs, Discrete Math. 86 (1/3) (1990) 165–177.
[4] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.
[5] P.C. Fishburn, On the sphericity and cubicity of graphs, J. Combin. Theory Ser. B 35 (1983) 309–318.
[6] D.R. Fulkerson, O.A. Gross, Incidence matrices and interval graphs, Pacific J. Math. 15 (3) (1965) 835–855.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[8] W.K. Hale, Frequency assignment: theory and applications, Proc. IEEE 68 (12) (1980) 1497–1514.
[9] T.F. Havel, The combinatorial distance geometry approach to the calculation of molecular conformation, Ph.D. Thesis, University of California, Berkeley, 1982.
[10] T.F. Havel, The combinatorial distance geometry approach to the calculation of molecular conformation, Congr. Numer. 35 (4) (1982) 361–371.
[11] T.F. Havel, I.D. Kuntz, G.M. Crippen, The combinatorial distance geometry method for the calculation of molecular conformation, I. A new approach to an old problem, J. Theor. Biol. 35 (4) (1983) 359–381.
[12] D.S. Johnson, C.A. Aragon, L.A. McGeogh, C. Schevon, Optimization by simulated annealing: an experimental evaluation. Part II. Graph coloring and number partioning, Oper. Res. 39 (3) (1991) 378–406.
[13] J. Kratochvil, Personal communication, February 1995.
[14] M.V. Marathe, H.B. Hunt III, S.S. Ravi, Geometry based approximations for intersection graphs, in: Fourth Canadian Conference on Computational Geometry, 10–14 August 1992, pp. 244–249.
[15] F.S. Roberts, Indifference graphs, in: Proof Techniques in Graph Theory (Proc. of the Second Ann Arbor Graph Theory Conference), Academic Press, New York, 1968, pp. 139–146.
[16] F.S. Roberts, Quo vadis, graph theory, Technical Report 91-33, DIMACS, May 1991.
[17] H. Sachs, Coin graphs, polyhedra, and conformal mapping, Discrete Math. 134 (1994) 133–138.