JOURNAL OF COMPUTATIONAL PHYSICS **137,** 346–365 (1997)
ARTICLE NO. CP975809

An Algorithm for Two-Dimensional Rigidity Percolation: The Pebble Game

Donald J. Jacobs* and Bruce Hendrickson†

**Department of Physics and Astronomy, Michigan State University, East Lansing, Michigan 48824; and †Applied and Numerical Math, Sandia National Laboratories, Albuquerque, New Mexico 87185-1110*
E-mail: *jacobs@pa.msu.edu and †bah@cs.sandia.gov

Received July 19, 1996; revised June 9, 1997

Many important macroscopic properties of materials depend upon the number of microscopic degrees of freedom. The task of counting the number of such degrees of freedom can be computationally very expensive. We describe a new approach for this calculation which is appropriate for two-dimensional, glass-like networks, building upon recent work in graph rigidity. This purely combinatorial algorithm is formulated in terms of a simple *pebble game.* It has allowed for the first studies of the rigidity transition in *generic* networks, which are models of amorphous materials and glasses. In the context of generic rigidity percolation, we show how to calculate the number of internal degrees of freedom, identify all rigid clusters, and locate the overconstrained regions. For a network of n sites the pebble game has a worst case performance of $O(n^2)$. In our applications its performance scaled as $n^{1.15}$ at the rigidity transition, while away from the transition region it grew linearly. © 1997 Academic Press

Key Words: rigidity percolation; graph rigidity; graph algorithm; vector percolation; floppy modes; network glass.

1. INTRODUCTION

Many macroscopic properties of a material can be understood from underlying microscopic structure. For example, microscopic resistance to structural deformation governs material strength. Microscopic rigidity has been widely studied in covalent glass networks by modeling bond lengths and angles as hard constraints. Angular constraints are modeled as next nearest neighbor bond constraints, and therefore no special distinction needs to be made. Using this model the rigidity of network glasses is well characterized by the number of *floppy modes* [24] which can be estimated by constraint counting. Mean field approximations for the properties of linear elasticity in network glasses [5] have also been successful in characterizing material strength. However, current understanding of the mechanical stability in such materials is limited and further theoretical development is needed. Fundamental questions remain unresolved like the nature of the *rigidity transition* from a floppy to a rigid network.

The study of network rigidity and the rigidity transition is known as *rigidity percolation* [4, 8] due to its close connections to connectivity percolation theory [21]. The essential property common to all percolation type problems is that of a connected pathway. Some property of interest, such as fluid flow, can be traced along paths within a network. In rigidity percolation the paths consist of sites that are mutually rigid. There are two important differences between rigidity and connectivity (e.g., fluid flow) percolation. Rigidity percolation is a vector (not a scalar) problem, and unlike connectivity, rigidity is inherently nonlocal. That is, whether a region is floppy or rigid generally depends upon structural details that are far away. These properties significantly complicate the study of rigidity percolation. Although brute force numerical techniques have been applied with great vigor [1, 2, 9, 20], their $O(n^3)$ run time for a network with n sites has severely constrained system sizes.

In this paper we describe a new algorithm for exploring rigidity percolation which combines insights from *graph rigidity* and material science [12]. Graph rigidity is concerned with the rigidity properties of structures composed of rigid bars joined via revolute joints, and has been well studied by mathematicians and engineers.

The results in graph rigidity are most complete for networks without any geometric singularities or symmetries. Conveniently, amorphous solids and polymeric glasses generally fit this description. For this class of problems, rigidity properties can be reduced to questions of connectivity, without worrying about geometric details. Thus, unlike many other aspects of physics, rigidity is simplified in networks which lack any regularity.

Algorithms for graph rigidity are most highly developed for two-dimensional systems, so that is the focus of this paper. The basic understanding of the rigidity transition for two-dimensional networks has been eluding researchers for many years due to the prohibitive cost of simulating large networks. This paper describes the techniques that have enabled the first detailed studies of rigidity percolation [12, 13]. Moukarzel [17] and Moukarzel and Duxbury [18] have used an approach closely related to ours in simultaneous and independent work with a different emphasis.

In Section 2, we review some fundamentals of rigidity theory and we recast the rigidity algorithm proposed by Hendrickson [10] in terms of a simple *pebble game.* Discussion of our implementation is given in Section 3 where the powerful method of triangularization is included which significantly improves the performance. We also discuss how to count the number of floppy modes, decompose the network into rigid clusters, and locate the overconstrained regions. In Section 4 we demonstrate the pebble game as a tool for studying rigidity percolation in two-dimensional generic networks. In Section 5 a brief summary and future outlook is given.

2. GRAPH RIGIDITY AND THE PEBBLE GAME

The structure of a network glass can be described as a *graph* G consisting of a set of *vertices* V, numbered from 1 to n, and a set of m *edges* E each of which connects a pair of vertices. The vertices correspond to sites and edges to bonds. We will say an edge is *incident* to vertex i if the edge connects i to some other vertex j, and we will denote such an edge by (i, j). The subgraph *induced* by a set of vertices is the graph consisting of those vertices and all edges that are only incident to those vertices. A *framework* $p(G)$ is a graph G along with a mapping $p\colon V \to \mathbb{R}^2$ which assigns each vertex to a point in the plane. We are interested in whether or not a framework can be flexed while keeping all the edge lengths unchanged.

To formalize our notion of a glass-like structure, we will assume that the locations of the vertices are *generic*; that is, the vertex coordinates are algebraically independent over the rationals. This requirement is actually stronger than we need; we just have to avoid several specific algebraic dependencies. The set of generic realizations is dense in the space of all realizations, and almost all realizations are generic. Therefore the key advantage in considering network glasses, unlike regular lattices, is that their rigidity properties are easier to analyze because all problematic configurations are eliminated.

2.1. *Basic Concepts*

A mathematical analysis of rigidity requires a formal definition of the intuitive notion of a flexible framework. We will limit our development to two dimensions, although the basic definitions are easily generalized.

A *finite flexing* of a framework $p(G)$ is a family of realizations of G, parameterized by t so that the location, r_i, of each vertex i, is a differentiable function of t and $|r_i(t) - r_j(t)|^2 = \textit{constant}$ for every $(i, j) \in E$. Thinking of t as time, and differentiating the edge length constraints we find that

$$(u_i - u_j) \cdot (r_i - r_j) = 0 \quad \text{for every } (i, j) \in E, \tag{1}$$

where u_i is the instantaneous velocity of vertex i. An assignment of velocities that satisfies Eq. (1) for a particular framework is an *infinitesimal motion* of that framework. Clearly the existence of a finite flexing implies an infinitesimal motion, but the converse is not always true. However, each infinitesimal motion corresponds to a finite flexing [19] in a generic realization, and this simplification is actualized in network glasses.

The infinitesimal motions of a framework constitute a vector space. Note that a motion of the Euclidean space itself, a rotation or translation, satisfies the definition of a finite flexing. Such finite flexings are said to be *trivial.* In two dimensions there are three such macroscopic trivial flexings, two translations and a rotation. If a framework has a nontrivial infinitesimal motion it is *infinitesimally flexible.* Otherwise it is *infinitesimally rigid.* Since we are considering only generic realizations we will drop the prefix and refer to frameworks as either rigid or flexible.

Our first task is to be able to determine whether a particular framework is rigid or flexible. Conveniently, this now becomes a property of the underlying graph as the following theorem indicates [7].

THEOREM 2.1. *If a graph has a single infinitesimally rigid realization, then all its generic realizations are rigid.*

The frameworks built from a graph are either all infinitesimally flexible or almost all rigid. This allows for the characterization of graphs as either rigid or flexible according to the typical behavior of a framework, without reference to a specific realization. It also allows us to be somewhat cavalier in the distinction between rigid frameworks and graphs that have rigid realizations. Henceforth such graphs will be referred to as *rigid graphs.*

How can a rigid graph be recognized? Clearly, graphs with many edges are more likely to be rigid than those with only a few, since the edges are constraining the possible movements of the vertices. In two dimensions a set of n vertices has $2n$ possible independent motions, but the three rigid body motions cannot be constrained. If each edge adds an independent constraint, then $2n - 3$ edges should be required to eliminate all nonrigid motions of the graph. Clearly, if any induced subgraph with n' vertices has more than $2n' - 3$ edges then these edges cannot be independent. This basic intuition is sound as the following result of Laman indicates [15].

THEOREM 2.2 (Laman). *The edges of a graph $G = (V, E)$ are independent in two dimensions if and only if no subgraph $G' = (V', E')$ has more than $2n' - 3$ edges.*

COROLLARY 2.3. *A graph with $2n - 3$ edges is rigid in two dimensions if and only if no subgraph G' has more than $2n' - 3$ edges.*

Laman's result was the first graph theoretic characterization of rigid graphs in two dimensions. Several equivalent characterizations have since been discovered [3, 11, 16, 22, 23]. We will refer to a subgraph with n' vertices and $2n' - 3$ independent edges as a *Laman subgraph.*

Unfortunately, for all its intuitive appeal the obvious generalization of Laman's theorem is not sufficient in higher dimensions (although it is a necessary condition for rigidity). The characterization of generically rigid graphs in three dimensions remains an open problem.

2.2. *The Pebble Game*

Although Laman's theorem characterizes rigidity, as stated it gives a poor algorithm. It requires counting the edges in every subgraph, of which there are an exponential number. Sugihara discovered the first polynomial time algorithm for determining the independence of a set of edges in two dimensions [22]. Imai presented an $O(n^2)$ algorithm for rigidity testing using a network flow approach [11]. This complexity was matched by Gabow and Westermann using matroid sums [6] and again by Hendrickson using bipartite matching [10]. In this section we reinterpret and simplify Hendrickson's algorithm to make it practical.

We will make extensive use of the following formulation of Laman's theorem.

THEOREM 2.4. *For a graph $G = (V, E)$ having m edges and n vertices, the following are equivalent.*

A. *The edges of G are independent in two dimensions.*

B. *For each edge (a, b) in G, the graph formed by adding three additional edges (a, b) has no induced subgraph G' in which $m' > 2n'$.*

The basic idea behind our algorithm is to grow a maximal set of independent edges one at a time. Denote these *basis* edges by $\hat{E}$. A new edge is added to the basis if it is discovered to be independent of the existing set. If $2n - 3$ independent edges are found then the graph is rigid. The key will be the efficient determination of whether or not a new edge is independent of the current basis.

Assume that we have a (possibly empty) set of independent edges $\hat{E}$. Combined with the vertices of G these form a graph $\hat{G}$. Note that the number of edges in $\hat{G}$ is $O(n)$. We wish to determine if another edge, e, is independent of $\hat{E}$. Adding e to $\hat{G}$ produces $\overline{G}$. By Theorem 2.4, e is independent of $\hat{E}$ if and only if there is no subgraph with too many edges ($m' > 2n'$) after any edge in $\overline{G}$ is quadrupled (i.e., adding three additional edges between the same pair of vertices). Actually, only e needs to be quadrupled as the following lemma demonstrates.

LEMMA 2.5. *A new edge e is independent of $\hat{E}$ if and only if the graph G_{4e} formed by quadrupling e has no induced subgraph G' in which $m' > 2n'$.*

Proof. The "only if" portion of the proof follows immediately from Theorem 2.4. For the "if" portion assume that no subgraph of G_{4e} has too many edges when e is quadrupled, but e is not independent of $\hat{E}$. Then by Theorem 2.4 there must exist some edge in $\hat{E}$ whose quadrupling causes G', a subgraph of $\overline{G}$, to have $m' > 2n'$. Since the edges of $\hat{E}$ are independent this bad subgraph must include e. But this bad subgraph has the same number of edges it had within G_{4e}, which leads to a contradiction. ∎

Lemma 2.5 reduces the complexity of independence testing to that of counting edges in subgraphs once the new edge is quadrupled. To do this we will make use of the following *pebble game.* Each vertex is given two pebbles. A vertex can use its pebbles to cover any two edges which are incident to that vertex. We would like to assign the pebbles in such a way that all the edges in the graph are covered. We will call such an assignment a *pebble covering.* We will show below that the existence of a pebble covering is equivalent to there being no induced subgraph on n' vertices with more than $2n'$ edges, which is the independence condition in Lemma 2.5. Before we can prove this, we need to describe an approach for moving pebbles around to cover a new edge.

Assume that we have a set of edges that are covered with pebbles. We now want to add a new edge, rearranging pebbles as necessary to cover it. First look at the vertices incident to the new edge. If either vertex has a free pebble, then use it to cover the new edge and exit. Otherwise, their pebbles are being used to cover existing edges. If a vertex at the other end of one of these edges has a free pebble then that pebble can be used to cover the existing edge, freeing up a pebble which can cover the new edge. More formally, we are searching for free pebbles in a *directed* graph, in which each edge has a preferred direction. Specifically, if edge

e_{ab} is covered by a pebble from vertex a, then the edge is directed from a to b. From each vertex v, we search along the (at most two) edges directed away from v. This search for free pebbles continues until either a pebble is found and a sequence of swaps allows a new edge to be covered, or else no more vertices can be reached and no free pebbles have been discovered. The algorithm is sketched in Fig. 1.

Several observations about this algorithm are in order. First, the time to perform an enlargement is linear in the number of vertices. All the initialization is linear, and the recursive calls to Find_Pebble perform a depth first search in the graph

```
Algorithm Enlarge_Cover($G$, $e_{ab}$)
    For Each vertex $v$
        seen($v$) = False, path($v$) = -1
    found = Find_Pebble($G$, $a$, seen, path)
    If (found) Then
        Rearrange_Pebbles($G$, $a$, seen)
        Return (Success)
    If (not seen($b$)) Then
        found = Find_Pebble($G$, $b$, seen, path)
        If (found) Then
            Rearrange_Pebbles($G$, $b$, path)
            Return (Success)
    Return (Failure)

Function Find_Pebble($G$, $v$, seen, path)
    seen($v$) = True, path($v$) = -1
    If ($v$ has free pebble) Then
        Return (True)
    Else
        $x$ = neighbor along edge my pebble covers
        If (not seen($x$)) Then
            path(v) = x
            found = Find_Pebble($G$, $x$, seen, path)
            if (found) Then Return (True)
        $y$ = neighbor along edge my other pebble covers
        If (not seen($y$)) Then
            path(v) = y
            found = Find_Pebble($G$, $y$, seen, path)
            if (found) Then Return (True)
    Return (False)

Subroutine Rearrange_Pebbles($G$, $v$, path)
    While (path($v$) $\neq -1$)
        $w$ = path($v$)
        if (path($w$) = $-1$) Then
            Cover edge ($v$,$w$) with free pebble from $w$
        Else
            Cover edge ($v$,$w$) with pebble from edge ($w$,path($w$)
        $v = w$
```

FIG. 1. Algorithm for enlarging a pebble covering.

of covered edges, which has at most $2n$ edges. Second, since the search implicit in `Find_Pebble` is just a search for free pebbles within a directed graph, the order of the search does not affect whether or not a pebble is found. In our implementation we use a breadth first search instead of the depth first search implicit in the recursion. Third, by applying `Enlarge_Cover` four times, once for each copy of the quadrupled edge, we can determine whether a pebble covering exists for G_{4e}.

Fourth, if `Enlarge_Cover` fails then all the pebbles owned by the vertices encountered in the search are already covering bonds. If n' vertices were encountered, then they must have at least $2n'$ induced edges to consume all their pebbles. This observation allows us to prove that pebble coverings are equivalent to edge independence, but we first need the following two lemmas.

LEMMA 2.6. *If the new edge, e, is tripled instead of quadrupled to form G_{3e}, then G_{3e} has a pebble covering.*

Proof. Assume the contrary. Then there is some induced subgraph G' of G_{3e} with $m' > 2n'$. Remove the three copies of e from this subgraph and quadruple one of the other edges already present. This altered subgraph still has $m' > 2n'$, but it is the graph generated by quadrupling an edge in $\hat{G}$. But since the edges of $\hat{G}$ are assumed to be independent this is a contradiction. ∎

LEMMA 2.7. *If G_{4e}, the graph constructed by quadrupling e, does not have a pebble covering, then the set of vertices n' encountered in* `Find_Pebble` *already have $2n' - 3$ induced edges.*

Proof. By Lemma 2.6 when e is quadrupled the first three copies of it can be covered. Since all the pebbles among the n' vertices are now in use, $2n' \leq m' + 3$. Since the m' edges are independent, we must have equality. ∎

In other words, when our algorithm determines that a new edge is not independent, it identifies a subgraph on n' vertices which already contains $2n' - 3$ edges. Otherwise, the algorithm will successfully enlarge the covering. This leads to the following theorem.

THEOREM 2.8. *A new edge e is independent of $\hat{E}$ if and only if there exists a pebble covering when e is quadrupled.*

Proof. Assume that a pebble covering exists. No subgraph on n' vertices can have more than $2n'$ induced edges since this is the total number of pebbles available to those vertices. When combined with Lemma 2.5, this completes the "if" portion of the proof.

For the "only if" proof, assume that e is independent, but no pebble covering exists. Algorithm `Enlarge_Cover` will either enlarge a covering or identify a subgraph on n' vertices with more than $2n'$ edges. If no such subgraph exists, the algorithm will find a pebble covering. ∎

COROLLARY 2.9. *If $\hat{E}$ is independent and a corresponding pebble covering $\hat{G}$ is known, then determining whether a new edge is independent requires $O(n)$ time.*

Proof. By Lemma 2.5, testing for independence of e requires just enlarging the covering in $\hat{G}$ to include the four copies of e. This requires four invocations of the `Enlarge_Cover` algorithm. ▌

This gives a two-dimensional rigidity testing algorithm that runs in worst case time $O(nm)$. Build a maximal set of independent edges one at a time by testing each edge for independence. Each test involves the enlargement of a pebble covering requiring $O(n)$ time. If the covering succeeds the edge is independent and is added to the basis. Otherwise it is discarded.

3. IMPLEMENTATION

A nice feature about the pebble game is that it directly corresponds to a physical picture. A free pebble at a site represents one independent motion. A site with two free pebbles has two translational motions. If two additional free pebbles are found at another site, then the distance between this pair of sites is not fixed. A bond placed between these sites is an independent constraint since the distance of separation is set to the bond length. Covering this bond with one of the four free pebbles marks it as an independent constraint. Once a bond is covered it must remain covered as pebbles are rearranged. The constraints in the movement of pebbles gives the property that a rigid cluster of two or more sites cannot have more than three free pebbles. The three free pebbles physically correspond to the three degrees of freedom required to specify the position of a rigid body in two dimensions.

We are interested in using the pebble game as a tool to study generic percolation and more generally the mechanical stability of network glasses. We wish to decompose a network into its rigid clusters. A site shared by more than one rigid cluster acts as a revolute joint or *pivot.* Many rigid clusters can be connected together in a network through pivots forming floppy regions that are governed by collective motions. The number of floppy modes [12, 24] is given by $F = 2n - (m - m_r)$, where n is the number of sites, m is the number of bonds, and m_r is the number of redundant bonds. The number of redundant bonds within a network is unique but their location is not unique because this depends on the choice of bonds used as an independent basis. Each redundant bond creates an overconstrained region. These regions may overlap depending on the distribution of redundant bonds.

There are three basic questions about the rigidity of a network that one can address. (1) How many independent degrees of freedom exist in the system? (2) Which set of sites form rigid clusters? (3) Where are the overconstrained regions? In Fig. 2 we show an example of a two-dimensional generic bar-joint network that has been analyzed using the pebble game. The rigid clusters, pivots, and overconstrained regions can be verified by inspection. We will work with this example to explain the implementation of the pebble game and how the above questions are answered.

3.1. *The Pebble Search*

The network connectivity is stored in a standard neighbor table which is used for building the network and in part for identifying rigid clusters as discussed below.

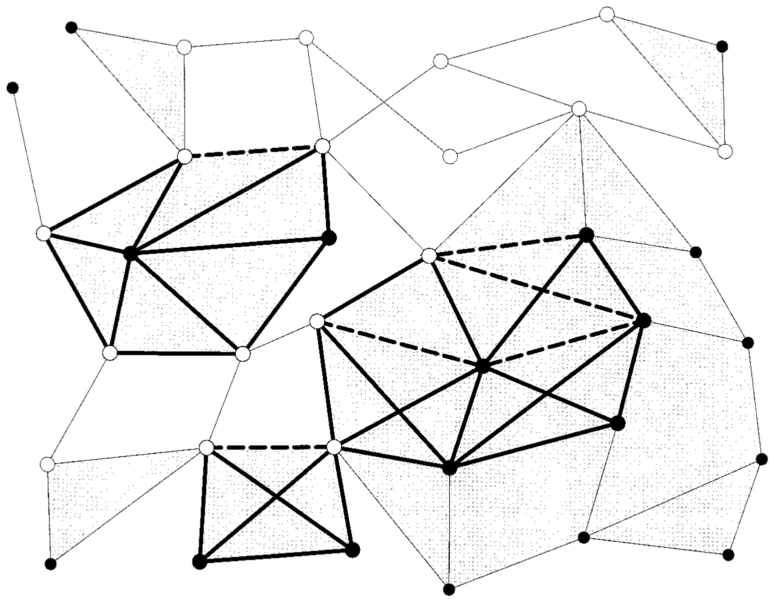

FIG. 2. A genetic network. Shaded regions highlight rigid clusters with more than one bond. The filled circles denote sites that are a member of only one cluster. Open circles denote pivots. Thick solid and dashed lines denote overconstrained bonds. A dashed line indicates a redundant bond.

Information about the network rigidity is efficiently stored in a *pebble index* which is a neighbor table with two columns defining the state of the two pebbles associated with each site. Either the pebble is free or it covers a bond connecting to the site that the pebble index points to. Uncovered bonds play no role. The pebble index is well suited to facilitate the frequently required pebble searches and rearrangements made by `Find_Pebble` and `Rearrange_Pebbles`, respectively.

Every site is initially assigned two free pebbles, and then bonds are added one at a time. The particular order of bond placement is arbitrary. `Enlarge_Cover` is used to test if a bond is independent or redundant. A bond is independent if two free pebbles can be collected at both incident sites simultaneously. The directed graph is automatically updated to cover an independent bond. We show in Fig. 3 our example network after placing the bottom row of bonds.

The pebble covering can be illustrated by assigning a direction to each covered bond. The bond (edge) is *directed* away from the site whose pebble covers it. A portion of such a directed graph is shown in Fig. 4 which corresponds to part of our example network in a late stage of analysis. Here we are interested in freeing up the two pebbles assigned to the bottom-left corner site. The outward search for a free pebble is easily accomplished using `Find_Pebble` by branching outward into unchecked territory guided by the directed graph. As soon as a free pebble is found, it is successively swapped using `Rearrange_Pebbles` along the backward trail. As the pebbles are rearranged, the new directed graph is automatically recorded within the pebble index. In general, `Find_Pebble` and `Rearrange_`

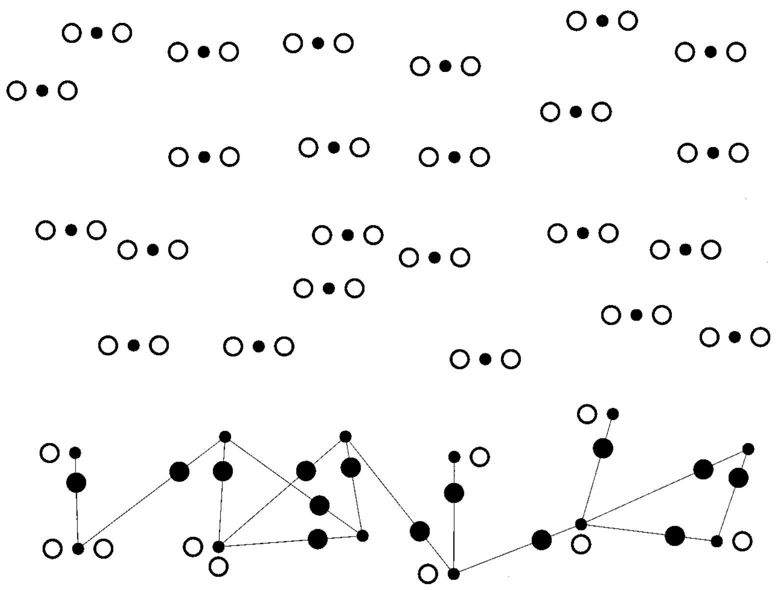

FIG. 3. Small filled circles denote sites. Large (filled, open) circles denote (covering, free) pebbles.

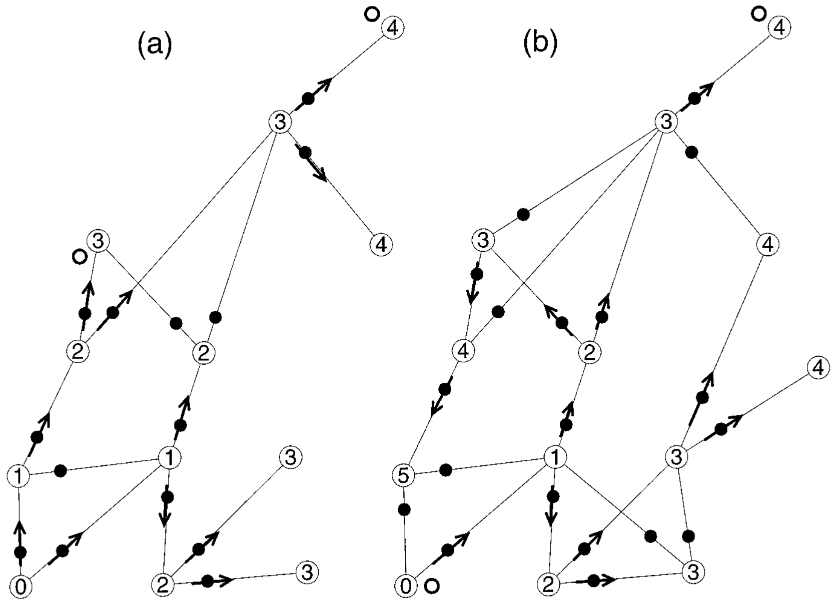

FIG. 4. A portion of a directed graph as determined by the pebble covering. The (filled, open) circles denote (covering, free) pebbles. A pebble search follows along the arrows. Each site is labeled by the *minimum* search length from the starting site. For example, the site with number 0 is the starting site and the number 3 on a site indicates that it can be reached in three steps. (a) A breath first search for a free pebble. (b) After a pebble rearrangement, a second pebble search is made.

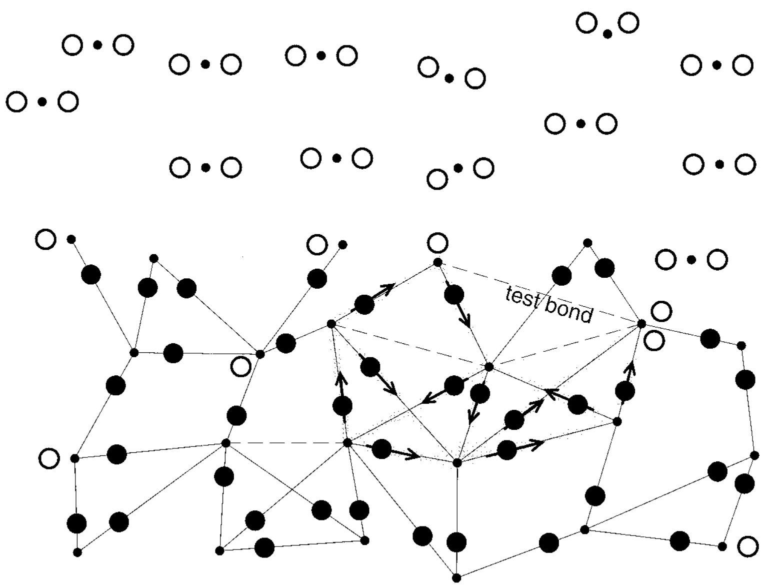

FIG. 5. A failed search (highlighted) across a test bond leads to a set of closed paths as indicated by the arrows. Dashed lines denote uncovered redundant bonds.

`Pebbles` become slower as the pebble searches lengthen, which happens when rigid regions become larger and the number of floppy modes diminishes. The run time can be significantly improved by reducing the average search length as described in the next section.

3.2. *Redundant Bonds and Laman Subgraphs*

A bond placed between a pair of sites is redundant when two free pebbles cannot be obtained at each incident site. A failed search for the forth pebble leads to a set of closed loops back to the two incident sites. The set of sites visited in the failed search constitutes a Laman subgraph. An example of finding a redundant bond and its associated Laman subgraph is shown in Fig. 5. All bonds which connect pairs of sites belonging to the Laman subgraph are overconstrained. This information must be *explicitly* recorded when it is of interest because it is completely lost once the next pebble search is made.

Without any cost in memory, a substantial efficiency improvement is made by *implicitly* recording Laman subgraphs within the directed graph. The idea is to change the connectivity of the pebble index immediately after a failed pebble search in order to reduce the average length of future searches. A Laman subgraph consists of n' mutually rigid sites with $2n' - 3$ independent bonds. Generally there are many ways to distribute these bonds to maintain rigidity. Since only independent bonds are relevant to the pebble index, the topology of a Laman subgraph can be transformed to a set of edge-sharing triangles. This effectively reduces an *arbitrarily large* Laman subgraph to the size of two edge-sharing triangles, dramatically shortening subsequent pebble searches.

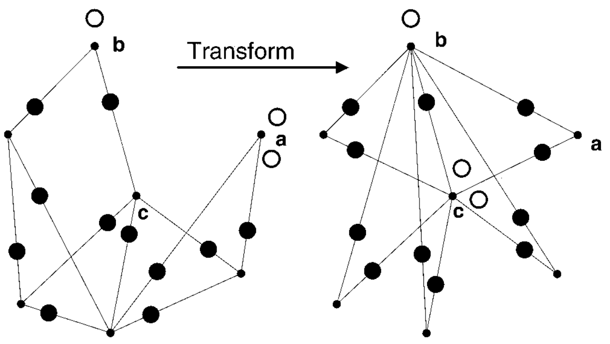

FIG. 6. (a) A Laman subgraph as a self-contained directed graph. (b) Triangularization transforms its topology into a set of triangles sharing a common base.

To implement this transformation notice that a failed pebble search identifies a self-contained directed graph that can be lifted from the network. In Fig. 6 we show how the Laman subgraph of Fig. 5 is transformed using a *triangularization* procedure.

1. Start at the redundant bond having three free pebbles. Label the incident site with two free pebbles as **a** and the incident site with one free pebble as **b** and define site **c** as the first site searched from **b**.

2. Place two free pebbles on site **c**. The covered bond directed from site **b** to site **c** defines a *common base*.

3. All remaining sites in the Laman subgraph, including site **a**, are triangulated to the common base, $\overline{\mathbf{bc}}$, as Fig. 6 demonstrates. Note that bond $\overline{\mathbf{ab}}$ could have been used as a common base; however, the pebble currently covering bond $\overline{\mathbf{bc}}$ must instead be used to cover bond $\overline{\mathbf{ba}}$.

Moukarzel [17] describes an alternative method in exploiting the discovery of Laman subgraphs. Although these techniques improve the practical behavior of algorithms, they do not reduce the worst case run time below $O(n^2)$.

3.3. *Identifying Rigid Clusters*

After placing the last bond in the network, it is decomposed into rigid clusters. Since a bond can only belong to one rigid cluster, each bond is assigned a unique cluster label. A set of bonds having the same label defines a rigid cluster. A site is identified as a pivot when it belongs to two or more rigid clusters. A streamlined method for identifying the rigid clusters uses `Find_Pebble` and `Rearrange_Pebbles` as basic tools.

Isolated sites are identified first as rigid clusters. To identify the remaining clusters:

1. Introduce a new cluster label for an unlabeled bond.

2. Gather three pebbles at its two incident sites.

3. Temporarily pin the three free pebbles down and mark the two incident sites as rigid.

4. Using the network neighbor table, check the unmarked nearest neighbors to the set of rigid sites. For each new unmarked site, perform a pebble search and attempt to free a pebble.

5. If a free pebble is found, the site is not mutually rigid with respect to the initial bond nor is any other site that was encountered during a pebble rearrangement. Mark all these sites floppy.

6. If a free pebble is not found, the site is mutually rigid with respect to the initial bond as well as all other sites that make up the failed search. Mark these sites as rigid and include them in the set of rigid sites.

7. Return to Step 4 until all nearest neighbors to the set of rigid sites have been marked floppy.

8. All bonds between pairs of sites marked rigid are given the same cluster label.

9. Remove floppy and rigid marks from all sites. Return to Step 1 until no unlabeled bonds remain.

The identification of rigid clusters using the above method is very efficient when previously marked sites, either as floppy or rigid, are used within `Find_Pebble`. The branching that occurs during a search for a free pebble is terminated at sites marked rigid. If a site is encountered that has been marked floppy, then it is immediately known that a free pebble can be extracted. Therefore, new sites can be identified as being mutually rigid or floppy with respect to the initial bond without reanalyzing any part of the network.

In general a rigid cluster will contain two types of bonds. *Isostatic* bonds are those essential to maintaining the rigidity of the structure—no redundant bond can replace them. Overconstrained regions are composed of a set of bonds, such that any one can be removed without changing the overall rigidity, because a redundant bond will take its place. Recall that these overconstrained regions are identified from failed pebble searches, and only these regions can carry stress [18]. However, both the overconstrained and the isostatic regions are found when identifying rigid clusters because the neighbor table of the network is used to facilitate a complete search.

3.4. *Boundary Conditions and Spanning Clusters*

One question we would like to answer about a network is whether or not there is a spanning rigid cluster. If so, the network will make a transition from being rigid to floppy as constraints are randomly removed. An example of this type of application is given in Section 4 for a generic triangular network. The particular boundary conditions will affect the likelihood of finding a spanning cluster. It is worth mentioning three common types of boundary conditions.

The simplest implementation is free boundary conditions where the network connectivity is truncated at the edges. The abrupt discontinuity tends to make the edges more floppy than within the bulk of the network. To avoid these surface effects, periodic boundary conditions are frequently used. By wrapping the network back on to itself the connectivity looks similar everywhere. (Actually, this so-called "periodic" boundary condition corresponds to a network having long-range bonds connecting its opposite edges.)

Another useful boundary condition is to apply *rigid bus bars* on two opposite sides of a network. The sites along an edge are made to be mutually rigid by triangulating each site to a common base. Either free or periodic boundary conditions can be applied to the other two edges sandwiched between the two bus bars. A useful application is to find the stressed backbone when the bus bars are loaded [17, 18]. This is implemented by applying a test bond between the top and bottom bus bar. When the test bond is found to be redundant the network can support the loading.

Testing for spanning rigid clusters is relatively simple. A spanning cluster is a collection of sites that are mutually rigid that form a connected path from one side of the network to the other side. It can be defined along certain directions, such as only along the horizontal or vertical direction, or one may wish to require a spanning rigid cluster to span both horizontal and vertical directions. With free boundaries, a cluster is spanning provided that it contains sites on opposite edges of the network. With horizontal rigid bus bars, a vertical spanning cluster forms when the two bus bars become mutually rigid. With periodic boundary conditions, a rigid path through the network must be found while making sure that the connecting path totally wraps around in the direction of interest. The vertical and horizontal directions must be checked separately even if spanning is required in both directions.

4. APPLICATION TO RIGIDITY PERCOLATION

The utility of the pebble game is nicely demonstrated in the context of generic rigidity percolation. We will focus on the pebble game rather than the results of rigidity percolation which are published elsewhere [12, 13, 18]. As a typical example, we consider an $L \times L$ generic triangular network. It has the connectivity of a triangular lattice but the site locations are distorted as to define a generic network. The degree of rigidity within the network is controlled by randomly removing either bonds or sites. The pebble game is then used as a tool to study the mechanical stability in these networks. All our simulations were performed on a DEC Alpha workstation.

For bond dilution all $n = L^2$ sites are present in the network and $m = p3L^2$ bonds are randomly placed throughout, where p is the *bond concentration* which ranges from $0 \leq p \leq 1$. For site dilution $(1 - q)L^2$ sites are randomly removed from the network leaving $n = qL^2$ sites present, where the *site concentration* q varies from $0 \leq q \leq 1$. Bonds are placed between all neighboring sites. The number of bonds in the network will vary, but the expected number is $\langle m \rangle = q^2 3L^2$.

At full concentration (when all sites/bonds are present) the generic triangular network is rigid and overconstrained. The network becomes floppy when it is sufficiently diluted. A transition from a rigid to floppy network defines the rigidity percolation threshold. This threshold can be found for a given network by monitoring when a spanning rigid cluster first appears as the bond or site concentration is increased. Operationally this is done by adding one bond to the network at a time. Using the approach detailed in this paper, we have found the percolation thresholds

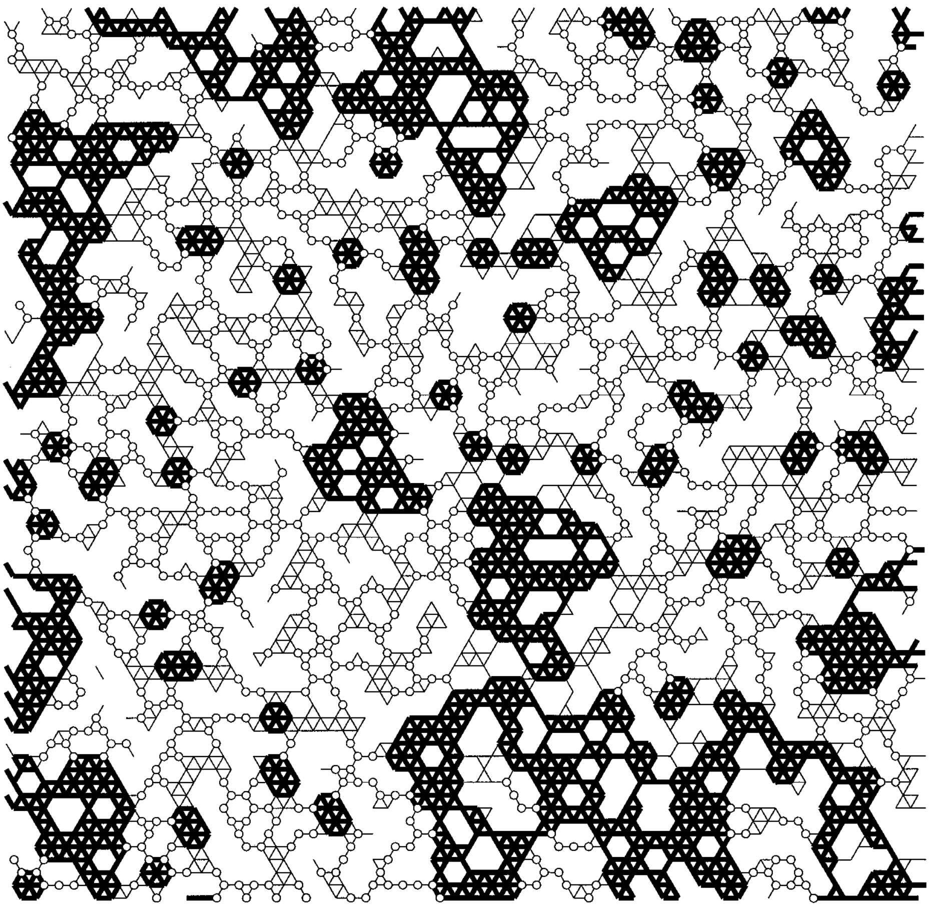

FIG. 7. The *topology* of a section of a site-diluted generic triangular network below the percolation threshold ($q = 0.6825$). Any particular realization will have distortions (not shown). The thin (thick) lines denote isostatic (overconstrained) bonds. The open circles denote pivot sites.

(extrapolated to infinite systems) for bond and site dilution to be $p_{\text{cen}} = 0.6602 \pm 0.0003$ and $q_{\text{cen}} = 0.6976 \pm 0.0003$, respectively [13].

We show two typical sections of a site-diluted network below and above the rigidity percolation threshold in Figs. 7 and 8, respectively. We have decomposed these networks into their rigid clusters. From the cluster statistics one can study rigidity percolation in a similar way as is done for connectivity percolation [21]. The pebble game serves as a tool for identifying rigid clusters, pivot sites, and overconstrained regions as demonstrated in Figs. 7 and 8. Although the overconstrained regions stand out the most in this diagram, they often make up only part of a rigid cluster. Many rigid clusters or regions within them are isostatic. As the site concentration is increased, various clusters merge and the overconstrained regions expand. Note that stress resides only within these overconstrained regions. The spanning overconstrained region of the network is called the *stress carrying backbone.* Locating this backbone allows one to substantially reduce the calculational time using relaxation type methods to find elastic constants [17, 18].

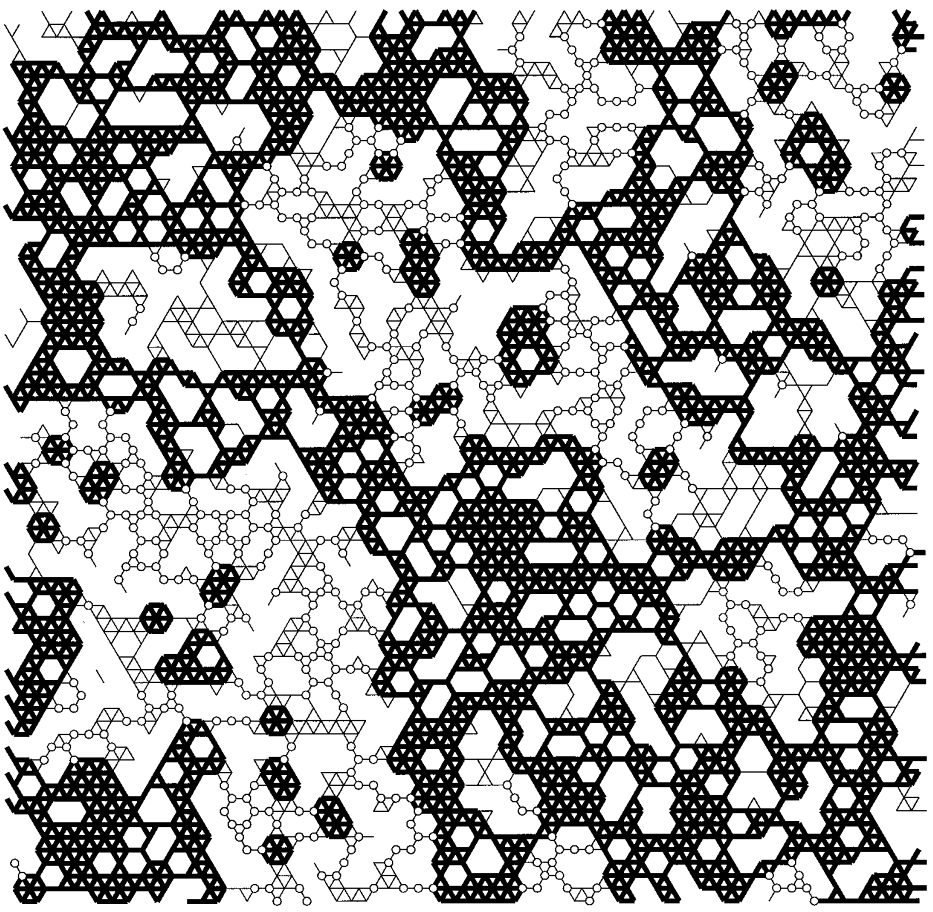

FIG. 8. Same as Fig. 7 except $q = 0.7125$ above the percolation threshold.

The number of floppy modes is a global quantity that tracks the degree of rigidity within the network. The number of floppy modes F can be expressed as

$$F = 2n - \sum_{s=2}^{n} C_s(2s - 3), \tag{2}$$

where C_s is the number of rigid clusters having s sites. This sum rule simply follows from counting $(2s - 3)$ independent bonds per cluster containing s sites. The first and second derivatives of F with respect to bond or site concentration show interesting behavior at the percolation threshold [12, 13, 24]. The first derivative is related to the change in F when one bond or site is added to the network. The second derivative is related to the change in F when a *pair* of bonds or sites is added to the network. The first and second derivatives can be calculated exactly for any given network by considering all possible ways to add either one bond (or site) or a pair of bonds (or sites), respectively. However, it is sufficient to sample only a small fraction of these possibilities to obtain accurate results. Note that triangularization cannot be implemented while performing this sampling because the pebble index must be restored after each test is completed.

In bond-diluted networks the first derivative of the fraction of floppy modes, given by $f = F/2n$, can be calculated exactly by knowing the total number of overconstrained bonds, m_0, as

$$\frac{d}{dp} f(p) = -\frac{z}{4}\left(1 - \frac{m_0}{m}\right), \tag{3}$$

where z is the full coordination number of the network. This result follows by considering all possible outcomes in the change of F with the *removal of one bond.* The removal of an overconstrained bond does not change F, while the removal of an isostatic bond increases F by one mode. This result also makes calculating the second derivative easier because one need only consider the change in the number of overconstrained bonds due to the addition of a single bond. There is no similar shortcut for site-diluted networks. Removing a single site generally corresponds to removing more than one bond locally, which influences the network in a correlated way.

The performance of the pebble game applied to bond- and site-diluted networks is shown in Fig. 9 over an entire concentration range. In both the bond- and the site-diluted networks we count the number of floppy modes, calculate its first derivative, locate all overconstrained regions, decompose the network into rigid clusters, and check for a spanning rigid cluster and a stress carrying backbone. In bond-diluted networks the second derivative of F is also calculated which requires about the same fraction of time to calculate the first derivative of F for site dilution.

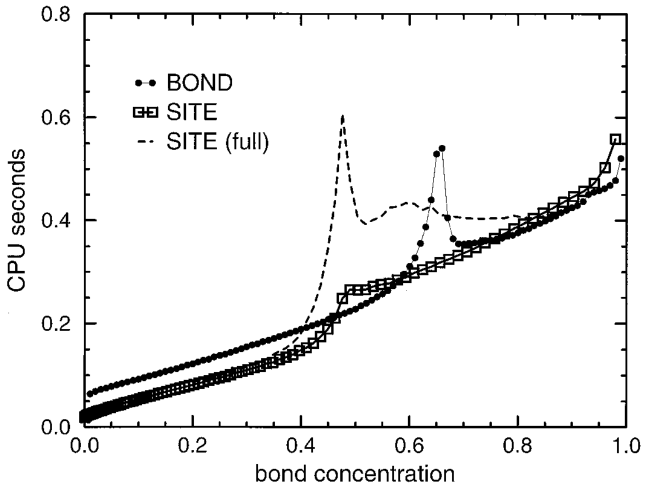

FIG. 9. The performance of the pebble game when analyzing rigidity percolation on a 128 × 128 generic triangular network. The dashed line includes a full calculation for the second derivative of F for site dilution.

This is about 5% of the total CPU time plotted in Fig. 9. The time to calculate the second derivative of F is large for site-diluted networks; therefore it is obtained by numerically differentiating the first derivative. However, in Fig. 9 we show the performance when the second derivative of F is also calculated by the method of sampling the change in F with a random placement of pairs of sites.

It is seen that the total CPU time increases near the rigidity percolation threshold. This occurs because the average length of a pebble search is becoming very long. This is a reflection that there are many large nonspanning interconnected isostatic regions. Below the threshold, pebble searches are shorter because there are more floppy modes and the isostatic regions are smaller. Above the threshold many large overconstrained regions appear, so that by triangularization the typical length of a pebble search shortens. These general effects are the same for both bond and site dilution; however, in the latter case the peak near the threshold is not as pronounced because the effective size of the network is increasing as the site concentration is increased. Also the cost in calculating the second derivative of F in site-diluted networks is too expensive to be practical.

The overall time dependence on network size is shown in Fig. 10 where we compare bond- and site-diluted networks at their respective critical thresholds and also near full concentration. Near the rigidity transition we find a time dependence that scales as $O(n^{1.15})$, with site-diluted networks performing better because their effective size is smaller. As expected, near full concentration we find virtually no difference between bond and site dilution, and surprisingly the time dependence on network size is nearly linear. Without triangularization this time dependence scales as $O(n^2)$. Although not shown, a linear dependence is also found below the threshold. The general shape of the curves in Fig. 9 will be the same for all network sizes, except the peaks near the percolation threshold will become more pronounced since in this concentration range the CPU time grows somewhat faster than linearly.

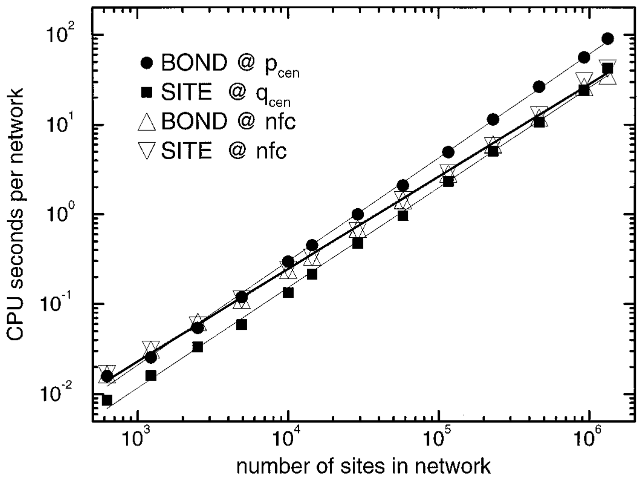

FIG. 10. CPU time for the pebble game as a function of network size. The number of sites is taken as $n = L^2$, although in site dilution only the fraction q of the sites are present. A comparison is made at their respective thresholds and near full concentration.

5. CONCLUSIONS

A major advance in our fundamental understanding of the mechanical stability of glass-like materials has been made by bridging the concepts of graph rigidity with material science. Namely, we see that numerous glass-like materials are well modeled as a simple bar-joint structure which is generic in nature. Only the topology of the network becomes important in this setting. The surprise is that networks lacking any underlying symmetry are easier to deal with mathematically.

We have presented an algorithm, manifested as a pebble game, that is sufficiently powerful to determine useful information about the rigidity of very large, generic networks in two dimensions. We have shown how the pebble game can be implemented to count the number of floppy modes, locate overconstrained regions, and identify all rigid clusters. For bond- and site-diluted generic triangular networks, we find that the pebble game has an $O(n^{1.15})$ time dependence near the rigidity transition, where n is the number of sites in the network. This algorithm can be used as a precursor to relaxation type methods in calculating quantities such as elastic constants or internal strain energies.

The success of the pebble game for two-dimensional generic networks is already stimulating research in rigidity percolation as well as other related problems dealing with mechanical stability in two dimensions. Hopefully, it will also provide new impetus toward a combinatorial characterization of rigidity in three-dimensional generic networks. Substantial progress has already been made in this direction [14] by generalizing the pebble game algorithm described here to three-dimensional bond-bending networks.

ACKNOWLEDGMENTS

The authors are indebted to Bob Connelly for helpful discussions about graph rigidity, and to M. F. Thorpe and P. M. Duxbury about rigidity percolation. Jacobs was partially supported under NSF Grant DMR-96 32182. Hendrickson was funded by a Hertz Foundation Fellowship and by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research and works at Sandia National Laboratories, operated for the U.S. DOE under Contract DE-AC04-76DP00789.

REFERENCES

1. S. Arbabi and M. Sahimi, Mechanics of disordered solids. I. Percolation on elastic networks with central forces, *Phys. Rev. B* **47,** 695 (1993).
2. Y. Cai and M. F. Thorpe, Floppy modes in network glasses, *Phys. Rev. B* **40,** 10535 (1989).
3. H. Crapo, *On the Generic Rigidity of Plane Frameworks,* Technical Report, Bat 10, Institut National de Recherche en Informatique et en Automatique, Cedex, France, 1988.
4. S. Feng and P. N. Sen, Percolation on elastic networks: New exponent and threshold, *Phys. Rev. Lett.* **52,** 216 (1984).
5. S. Feng and M. F. Thorpe, Effective-medium theory of percolation on central-force elastic networks, *Phys. Rev. B* **31,** 276 (1985).
6. H. N. Gabow and H. H. Westermann, Forests, frames and games: Algorithms for matroid sums and applications, in *Proceedings, 20th Annual Symposium on the Theory of Computing, Chicago, 1988,* pp. 407–421.

7. H. Gluck, Almost all simply connected closed surfaces are rigid, in *Geometric Topology,* Lecture Notes in Mathematics, No. 438 (Springer-Verlag, Berlin, 1975), pp. 225–239.
8. E. Guyon, S. Roux, A. Hansen, D. Bibeau, J. P. Troadec, and H. Crapo, Non-local and non-linear problems in the mechanics of disordered systems: Application to granular media and rigidity problems, *Rep. Prog. Phys.* **53,** 373 (1990).
9. A. Hansen and S. Roux, Universality class of central-force percolation, *Phys. Rev. B* **40,** 749 (1989).
10. B. Hendrickson, Conditions for unique graph realizations, *SIAM J. Comput.* **21,** No. 1, 65 (Feb. 1992).
11. H. Imai, On combinatorial structures of line drawings of polyhedra. *Discrete Appl. Math.* **10,** 79 (1985).
12. D. J. Jacobs and M. F. Thorpe, Generic rigidity percolation: The pebble game, *Phys. Rev. Lett.* **75,** 4051 (1995).
13. D. J. Jacobs and M. F. Thorpe, Generic rigidity percolation in two dimensions, *Phys. Rev. E* **53,** 3682 (1996).
14. D. J. Jacobs, Generic rigidity in three dimensional bond-bending networks, preprint, 1997.
15. G. Laman, On graphs and rigidity of plane skeletal structures, *J. Eng. Math.* **4,** 331 (1970).
16. L. Lovasz and Y. Yemini, On generic rigidity in the plane, *SIAM J. Algebraic Discrete Math.* **3,** 91 (1982).
17. C. Moukarzel, An efficient algorithm for testing the rigidity of planar graphs, *J. Phys. A: Math. Gen.* **29,** 8097 (1996).
18. C. Moukarzel and P. M. Duxbury, Stressed backbone and elasticity of random central-force systems, *Phys. Rev. Lett.* **75,** 4055 (1995).
19. B. Roth, Rigid and flexible frameworks, *Amer. Math. Monthly* **88,** 6 (1981).
20. S. Roux and A. Hansen, Transfer matrix study of the elastic properties of central-force percolation, *Europhys. Lett.* **6,** 301 (1988).
21. D. Stauffer, *Introduction to Percolation Theory* (Taylor & Francis, London/Philadelphia, 1985).
22. K. Sugihara, On redundant bracing in plane skeletal structures, *Bull. Electrotech. Lab.* **44,** 376 (1980).
23. T.-S. Tay and W. Whiteley, Generating isostatic frameworks, *Topol. Struct.* **11,** 21 (1985).
24. M. F. Thorpe, Continuous deformations in random networks, *J. Non-Cryst. Solids* **57,** 355 (1983).