Electronic Transactions on Numerical Analysis.
Volume 21, pp. 107-124, 2005.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

FINDING NONOVERLAPPING SUBSTRUCTURES OF A SPARSE MATRIX*

ALI PINAR† AND VIRGINIA VASSILEVSKA‡

Dedicated to Alan George on the occasion of his 60th birthday

Abstract. Many applications of scientific computing rely on sparse matrix computations, thus efficient implementations of sparse matrix kernels are crucial for the overall efficiency of these applications. Due to the low compute-to-memory ratio and irregular memory access patterns, the performance of sparse matrix kernels is often far away from the peak performance on modern processors. Alternative matrix representations have been proposed, where the matrix A is split into A_d and A_s, so that A_d contains all dense blocks of a specified form in the matrix, and A_s contains the remaining entries. This facilitates using dense matrix kernels on the entries of A_d, producing better memory performance. We study the problem of finding a maximum number of nonoverlapping rectangular dense blocks in a sparse matrix. We show that the maximum nonoverlapping dense blocks problem is NP-complete by a reduction from the maximum independent set problem on cubic planar graphs. We also propose a $2/3$-approximation algorithm for 2×2 blocks that runs in linear time in the number of nonzeros in the matrix. We discuss alternatives to rectangular blocks such as diagonal blocks and cross blocks and present complexity analysis and approximation algorithms.

Key words. Memory performance, memory-efficient data structures, high-performance computing, sparse matrices, independent sets, NP-completeness, approximation algorithms

AMS subject classifications. 65F50, 65Y20, 05C50, 05C69, 68Q17,68W25

1. Introduction.

Sparse matrices lie at the heart of many computation-intensive applications such as finite-element simulations, decision support systems in management science, power systems analysis, circuit simulations, and information retrieval. The performance of these applications relies directly on the performance of the employed sparse matrix kernels. The poor memory performance of sparse matrix operations on modern processors is arguably the most crucial problem in high performance computing. To overcome this memory bottleneck, alternative, memory-friendly data structures for sparse matrices have been investigated. One common approach is to exploit the special substructures in a sparse matrix, such as small dense matrices, to decrease the number of extra load operations. In this paper, we study the problem of finding a maximum number of nonoverlapping substructures in a sparse matrix, with the objective of improving the effectiveness of sparse matrix data structures that exploit dense blocks.

Conventional data structures for sparse matrices have two components: an array that stores floating-point entries of the matrix and arrays that store the nonzero structure (i.e., pointers to the locations of the numerical entries). Exploiting sparsity invariably requires using pointers, but pointers often lead to poor memory performance. One reason for the poor memory performance is that pointers cause an irregular memory access pattern and thus poor spatial locality. Another important reason, which is often overlooked, is the extra load operations. Each operation on a nonzero entry requires loading the location of that nonzero before loading the actual floating point number. For instance, sparse matrix vector multiplication, which is one of the most important kernels in numerical algorithms, requires three load operations for each multiply-and-add operation. And it has been observed that this overhead is usually more costly than the floating point operations [9].

*Received August 10, 2004. Accepted for publication November 8, 2005. Recommended by S. Toledo. This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract DE-AC03-76SF00098.

†Corresponding author. Computational Research Division, Lawrence Berkeley National Laboratory (apinar@lbl.gov).

‡Computer Science Department, Carnegie Mellon University (virgi@cs.cmu.edu).

ETNA
Kent State University
etna@mcs.kent.edu

$$
\begin{pmatrix}
x & x & x & & \\
 & x & & x & \\
x & x & & & \\
 & & & x & x \\
 & & x & x & x
\end{pmatrix}
=
\begin{pmatrix}
x & x & & & \\
 & & & & \\
x & x & & & \\
 & & & x & x \\
 & & x & x &
\end{pmatrix}
+
\begin{pmatrix}
 & & x & & \\
 & x & & x & \\
 & & & & \\
 & & & & \\
 & & & & x
\end{pmatrix}
$$

$$A = A_{12} + A_{11}$$

FIG. 1.1. *Matrix splitting.*

Recent studies have investigated improving memory performance of sparse matrix operations by reducing the number of extra load operations [9, 11, 13, 15]. Bik and Wijshoff propose algorithms to detect particular sparsity structures of a matrix, such as banded and blocked forms [4]. Toledo [13] studies splitting the matrix as $A = A_{12} + A_{11}$, where A_{12} includes 1×2 blocks of the matrix (two nonzeros in consecutive positions on the same row), and A_{11} covers the remaining nonzeros, as illustrated in Fig. 1.1. Notice that it is sufficient to store one pointer for each block in A_{12}. Pınar and Heath study the reordering problem to increase the sizes of these blocks [11]. They propose a graph model to reduce the matrix ordering problem to the traveling salesperson problem. Vuduc *et al.* study various blocking techniques to decrease load operations and improve cache utilization [15]. Significant speedups in large experimental sets have been observed, which motivates searching for larger blocks in the matrix for better performance. The splitting operation can be generalized to exploit various substructures. For instance, one can split the matrix into $A = A_d + A_s$, where A_d contains all specified substructures, and A_s contains the remaining entries. For a specified substructure, having more entries in A_d merits fewer load operations, thus better memory performance. This calls for efficient algorithms to find a maximum number of nonoverlapping substructures in a sparse matrix. A greedy algorithm is sufficient to find a maximum number of nonoverlapping $m \times n$ dense matrices when $m = 1$ or $n = 1$. However, this problem is much harder when $m, n \geq 2$.

We study the problem of finding a maximum number of nonoverlapping substructures of a sparse matrix, which we call the *maximum nonoverlapping substructures* problem. We focus on $m \times n$ dense blocks as a substructure, since they are common in sparse matrices arising in various applications, and their usage can effectively decrease the number of extra load operations. We call this problem the *maximum nonoverlapping dense blocks* problem. In Section 2, we define the problem formally and investigate its relation to the maximum independent set problem. We define a class of graphs for which the independent set problem is equivalent to the maximum nonoverlapping dense blocks problem. In Section 3, we use this relation to prove that the maximum nonoverlapping dense blocks problem is NP-complete. Our proof uses a reduction from the maximum independent set problem on cubic planar graphs and adopts orthogonal drawings of planar graphs. Section 4 presents an approximation algorithm for the problem. Since our techniques will potentially be used at application run-time, we are interested in fast and effective heuristics for the preprocessing cost to be amortized over the speedups in subsequent sparse matrix operations. Our algorithms require only linear time and space, and generate solutions whose sizes are within $2/3$ of the optimal for 2×2 blocks. In Section 5, we discuss alternative patterns to rectangular blocks. We show how the problems of finding rectangular and diagonal blocks can be transformed to each other to conclude that finding the maximum number of nonoverlapping diagonal blocks is NP-hard. We show how to use the approximation algorithm for rectangular dense blocks to obtain a linear 2/3-approximation algorithm for diagonal blocks. We also discuss cross

ETNA
Kent State University
etna@mcs.kent.edu

blocks and their variations. We present some open problems in Section 6 and conclude with Section 7.

This problem has only recently started to draw the attention of the sparse matrix community, but has been studied under different names as a combinatorial optimization problem. Fowler *et al.* [6] study this problem as a geometric embedding problem and prove it is NP-Complete by reduction from the 3-satisfiability problem (3SAT)[1]. Berman *et al.* [3] discuss a similar problem as the optimal tile salvage problem. In the optimal tile salvage problem, we are given an $\sqrt{N} \times \sqrt{N}$ region of the plane tiled with unit squares, some of which have been removed. The task is to find a maximum number of functional nonoverlapping $m \times n$ tiled rectangles. The difference between our problem and the optimal tile salvage problem is that in the tile salvage problem the tiles are allowed to be in any orientation ($m \times n$ or $n \times m$), whereas in our case the orientation is fixed (only $m \times n$). The NP-completeness proof of the tile salvage problem by Berman *et al.* is based on the flexibility in the orientation of the dense block, and thus is not applicable to our problem. Berman *et al.* describe a polynomial time approximation scheme, which for all $\delta > 0$, $\epsilon = O(1/\sqrt{\delta \log M})$, where M is the optimal solution value, gives an $(1-\epsilon)$-approximation. Their algorithm is based on maximum planar H-matching, which runs in $O(N^{1+\delta})$ steps, and can be applied to find square blocks where the two problems coincide. Baker [2] also has an algorithm for square blocks, which runs in $O(8^k N)$-time and $O(4^k N)$ space and produces a $(k-1)/k$-approximation. Hochbaum and Maass [8] also describe an algorithm for square blocks that gives a $(k-1)/k$-approximation, but runs in $O(m^2 k^2 N^{k^2})$ time to find $m \times m$ blocks on an $N \times N$ grid. Arikati *et al.* [1] study this problem as the two-dimensional pattern matching problem, and describe an approximation algorithm that runs in $O(N \lg N)$ time and produces solutions that are only $O(1/\sqrt{\log \log N})$ away from an optimal solution. They describe another algorithm that runs in $O(kN)$, and produces solutions that are within $(k-1)/k$ of the optimal. For our purposes, we need algorithms that are very fast and do not require auxiliary data structures. The greedy approximation algorithms we propose are very simple, space-efficient, and require only a single pass through the matrix.

2. Preliminaries. In this section we define the problems formally, and present definitions and some preliminary results that will be used in the following sections.

2.1. Problem Definition. We investigate the problem of finding a maximum number of nonoverlapping matrix substructures of prescribed form and orientation.

DEFINITION 2.1. *An* $m \times n$ pattern *is a* 0-1 $m \times n$ *matrix* σ. *An* oriented σ-substructure *of a matrix* A *is an* $m \times n$ *submatrix* A_1 *in* A *so that* $A_1(i,j) \neq 0$ *if* $\sigma(i,j) = 1$ *for* $1 \leq i \leq m$, *and* $1 \leq j \leq n$. *Two substructures* A_1 *and* A_2 *overlap if they share nonzero entry in* A_1 *with coordinates* (i_1, j_1) *in* A_1 *and* (i_2, j_2) *in* A_2 *and* $\sigma(i_1, j_1) = \sigma(i_2, j_2) = 1$.

Given a particular pattern σ, we define the *maximum nonoverlapping σ-substructures* (MNS) problem as follows.

> *Given an $M \times N$ matrix A and integer K, does A contain K disjoint σ-substructures?*

In this paper, we mostly focus on *dense blocks* due to their simplicity and their effectiveness in speeding up sparse matrix operations. A *dense block* of a matrix is a submatrix of specified size, all of whose entries are nonzero, i.e., it is a σ-substructure where σ is the all 1s matrix. We associate a dense block with its upper left entry. Two blocks overlap if they share a matrix entry. Formally,

> *Given an $M \times N$ matrix $A = (a_{ij})$, we say b_{ij} is an $m \times n$ dense block in A iff $a_{kl} \neq 0$ for all k and l such that $i \leq k < i+m \leq M$ and*

[1]This has been pointed to us by a reviewer after the completion of this work.

ETNA
Kent State University
etna@mcs.kent.edu

$j \leq l < j + n \leq N$. Two $m \times n$ blocks b_{ij} and b_{kl} overlap iff $|k - i| < m$ and $|l - j| < n$.

We define the *maximum nonoverlapping dense blocks* (MNDB) problem, which restricts the MNS problem to dense blocks as follows.

Given an $M \times N$ matrix A, positive integers m and n that define the block size, and a positive integer K, does A contain K disjoint $m \times n$ dense blocks?

2.2. Intersection Graphs. Although it is easy to find all specified patterns in a matrix, what we seek is a subset of *nonoverlapping* blocks. In this sense, the MNS problem is related to the *maximum independent set* (MIS) problem, which is defined as finding a maximum cardinality subset of vertices I of a graph G such that no two vertices in I are adjacent. We reveal the relation between the independent set and the nonoverlapping blocks problems using *intersection graphs* defined below.

DEFINITION 2.2. *A graph G is an intersection graph of the σ-substructures of a matrix A if there is a bijection ϕ between the vertices of G and the substructures of A, such that there is an edge in G between $\phi(s_1)$ and $\phi(s_2)$ if and only if s_1 and s_2 overlap in A.*

We use $G(A, m, n)$ to refer to the intersection graph of dense $m \times n$ blocks in matrix A. A maximum independent set on $G(A, m, n)$ gives a maximum number of nonoverlapping blocks in A. Thus the MNDB problem can be reduced to the maximum independent set problem, which is not even constant factor approximable. However, MNDB is not as hard as the general MIS problem, and some block intersection graphs have special structures, which can be exploited for efficient solutions. For instance, a greedy algorithm is sufficient to find a maximum number of nonoverlapping $1 \times n$ and $m \times 1$ blocks, since these problems reduce to a family of disjoint maximum independent set problems on interval graphs.

We will now define the class of graphs that constitute block intersection graphs. An intersection graph of a set of 2×2 dense blocks is an induced subgraph of the so called X-grid which consists of the usual 2 dimensional grid, and diagonals for each grid square. In general, the intersection graph of a set of $m \times n$ dense blocks is an induced subgraph of the X_{mn} grid. Below, we first define an X_{mn} grid, and then restrict the definition to define the graph class $X\Gamma_{mn}$ that represent graphs that can be intersection graphs for matrices.

DEFINITION 2.3. *An $M \times N$ X_{mn} grid is a graph with vertex set V and edge set E, so that*

- $V = \{v_{ij} : 1 \leq i \leq M - m + 1; 1 \leq j \leq N - n + 1\}$
- $E = \{(v_{ij}, v_{kl}) : v_{ij}, v_{kl} \in V;\ \ |i - k| < m \ \text{ and } \ |j - l| < n\}$

In an X_{mn} grid, vertex v_{ij} corresponds to the block b_{ij} in the matrix, and edges correspond to all possible overlaps between blocks. However, not all induced subgraphs of the X_{mn} grid are intersection graphs of a matrix. For example, if b_{ij} and $b_{i+m,j}$ are blocks in the matrix, then $b_{i+1,j}, \ldots, b_{i+m-1,j}$ should also be in the intersection graph to ensure the intersection graph represents all blocks. Therefore, we define a graph class $X\Gamma_{mn}$, which adds a closure property to an X_{mn} grid to cover such cases.

DEFINITION 2.4. *A graph $G = (V, E)$ is in the graph class $X\Gamma_{mn}$ if and only if it is an induced subgraph of an X_{mn} grid and has the closure property so that*

$$v_{ij} \in V \text{ if } \forall i \leq k < i + m, j \leq l < j + m; \exists v_{st} : s \leq k < s + m \text{ and } t \leq l < l + n$$

ETNA
Kent State University
etna@mcs.kent.edu

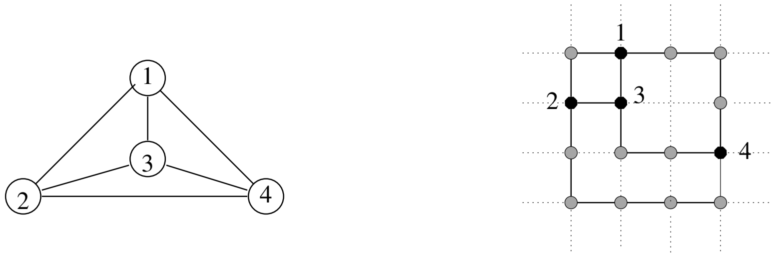

FIG. 2.1. *Planar orthogonal drawing*

The closure property enforces that there is a vertex in the graph for each dense block in the matrix. Being an induced subgraph of an X_{mn} grid guarantees that there is an edge for each overlap. The graphs in this class are exactly the intersection graphs of the $m \times n$ blocks in a matrix, thus finding a maximum independent set of a graph in this class is equivalent to solving the MNDB problem of the corresponding dense matrix blocks. This claim is formalized by the following lemma.

LEMMA 2.1. *An instance of the MNDB problem for finding $m \times n$ nonoverlapping dense blocks in a matrix A is equivalent to an instance of MIS for a graph in $X\Gamma_{mn}$.*

Proof. We show a one-to-one correspondence between intersection graphs, and graphs in $X\Gamma_{mn}$. Each dense block b_{ij} corresponds to the vertex v_{ij} in $G(A, m, n)$. By definition of the class $X\Gamma_{mn}$, $G(A, m, n) \in X\Gamma_{mn}$, and thus any instance of an MNDB problem can be reduced to an independent set problem in a graph in $X\Gamma_{mn}$.

Given a graph G in $X\Gamma_{mn}$, define $A = (a_{ij})$, so that a_{ij} is a nonzero iff $k \leq i < k + m$ and $l \leq j < l + n$ for some vertex v_{kl} in G. Observe that any dense block in A must be represented by a vertex in G due to the closure property. Also, for any two adjacent vertices in G, corresponding blocks intersect in A, and no other blocks overlap, due to the definition of edges in X_{mn}. Thus, a maximum cardinality subset of nonoverlapping blocks in matrix A corresponds to a maximum independent set in $G \in X\Gamma_{mn}$. ∎

The following lemma shows that removing a subset of the vertices along with their neighbors preserves the characteristics of the graph, providing the basis for greedy approximation algorithms as will be presented in Section 4.

LEMMA 2.2. *Let $G = (V, E)$ be a graph in $X\Gamma_{mn}$, $S \subseteq V$ a subset of vertices, and $N(S) = \{u \mid (u, v) \in E,\ v \in S,\ u \notin S\}$ be the neighborhood of S in G. Then the graph G' induced by $V \setminus (S \cup N(S))$ is still in $X\Gamma_{mn}$.*

Proof. Removing a vertex and its neighbors in G corresponds to removing all nonzeros in a block in the corresponding matrix. The remaining graph is the intersection graph of the resulting matrix. ∎

2.3. Planar Graphs and Orthogonal Drawings. A graph G is *planar* if and only if there is an embedding of G on the sphere such that no two edges have a point in common besides possibly common end points. G is *cubic planar* if every vertex has degree 3.

An *orthogonal drawing* of a graph G is an embedding of G onto a 2-dimensional rectangular grid such that every vertex is mapped to a grid point and every edge is mapped to a continuous path of grid line segments connecting the end points of the edge. When G is planar, the edge paths do not cross. An example of orthogonal embedding of a planar graph is illustrated in Fig. 2.1. No two edges share a grid point, and no edge path can go through a vertex unless this vertex is an end point of the edge corresponding to the path and is an end point of the path itself. Kant [10] showed that every planar graph G with n vertices and maximum degree 3 can be drawn orthogonally on an $O(n) \times O(n)$ grid in polynomial time.

ETNA
Kent State University
etna@mcs.kent.edu

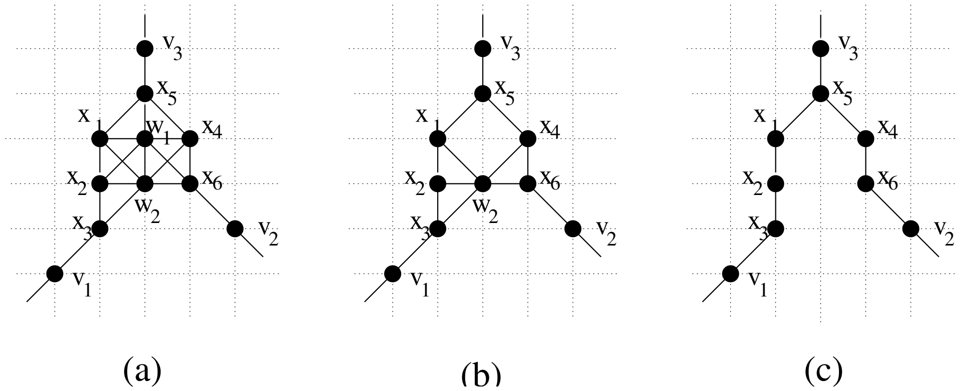

FIG. 3.1. *Transformation to preserve closure properties*

The NP-completeness proof in the next section uses a reduction from the maximum independent set (MIS) problem on cubic planar graphs and adopts orthogonal drawings.

3. Complexity. This section proves that the MNDB problem is NP-complete using a reduction from the independent set problem on cubic planar graphs, which is NP-complete [7]. The same result has been reported by Fowler *et al.* [6], by using a reduction from 3-satisfiability. The technique used here is significantly different than Fowler *et al.*'s. In this section, we will use $X\Gamma$ to refer to $X\Gamma_{22}$ for simplicity. The next lemma explains how we can retain independent set characteristics of the problem after transformations.

LEMMA 3.1. *Let $G = (V, E)$ be a graph, and u, v be two adjacent vertices in G, so that all neighbors of u besides v are also neighbors of v. Let $G' = (V', E')$ be the graph G after vertex v is removed. The size of the maximum independent set in G is equal to the size of the maximum independent set in G'.*

Proof. If vertex v is in a maximum independent set I, then none of its neighbors are in I. Thus $I' = I \cup \{v\} \setminus \{u\}$ is an independent set in G and in G', and $|I'| = |I|$. ◻

The following corollary will be used in our NP-completeness proof, as the structures in Fig. 3.1(a) arise in our construction.

COROLLARY 3.2. *Let $G \in X\Gamma$ contain the graph H in Fig. 3.1(a) as an induced subgraph so that all vertices except for possibly v_1, v_2 and v_3 have all of their neighbors in H. Then any instance $(G,\ K)$ of MIS is equivalent to the instance $(G',\ K)$ of MIS for the graph $G' = G \setminus \{w_1, w_2\}$.*

Proof. By Lemma 3.1, we can remove w_1 from the graph since all neighbors of x_1 are neighbors of w_1 as well. The reduced graph is illustrated in Fig. 3.1(b). Again using Lemma 3.1, we can remove w_2 since it covers all neighbors of x_2. Furthermore, we can apply the same transformation in reverse order to add vertices w_1 and w_2 to the graph in Fig. 3.1(c). ◻

The following lemma describes how edges of a graph can be replaced by even length paths, while preserving independent set characteristics.

LEMMA 3.3. *Let $G = (V, E)$ be a graph and $e = (v_i, v_j) \in E$ be an edge. Let $G_{e,k}$ be the graph G with the edge e substituted by a simple path $v_i, w_1, w_2, \ldots, w_{2k}, v_j$ where $k \in \mathbf{Z}^+$ and w_i are new vertices not in the original graph. Then there exists an independent set of size K in G iff there exists an independent set of size $K + k$ in $G_{e,k}$.*

Proof. We present the proof for $k = 1$, and the result follows by induction.

Sufficiency: Let I be an independent set in G, then either $v_i \notin I$ or $v_j \notin I$. Without loss of

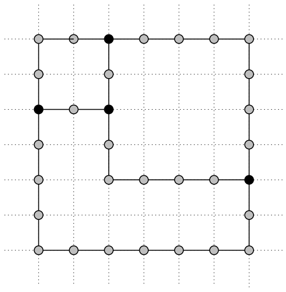

FIG. 3.2. *Enlargement operation on Fig. 2.1 (right) for $k = 1$.*

generality, assume $v_i \notin I$, then $I' = I \cup \{w_1\}$ is an independent set in $G_{e,k}$.
Necessity: Let I' be an independent set in $G_{e,k}$. If $w_1 \in I'$, then $v_i \notin I'$, and $I = I' \setminus \{w_1\}$ is an independent set in G. Symmetrically, if $w_2 \in I'$, then $v_j \notin I'$, and $I = I' \setminus \{w_2\}$ is an independent set in G. If $w_1, w_2 \notin I'$, then $I = I'$ is an independent set in G. ☐

We first analyze the complexity for 2×2 blocks for clarity of presentation, and then extend our result to $m \times n$ blocks for $m, n \geq 2$.

THEOREM 3.4. *Problem MNDB is NP-complete for* 2×2 *blocks.*

Proof. MNDB is clearly in NP since it is equivalent to a special case of MIS.

To show NP-hardness, we use a reduction from the independent set problem on cubic planar graphs, which is NP-complete [7]. We first embed a cubic planar graph orthogonally onto a grid as discussed in Section 2.3. Then we transform the embedded graph so that it is in $X\Gamma$. Our transformations preserve independent set characteristics so that an independent set in the transformed graph can be translated to an independent set in the original graph. Finally we use Lemma 2.1 to relate the independent set problem on a graph in $X\Gamma$, to the MNDB problem.

Our transformations are local. We first *enlarge* the grid to make room for these transformations by inserting k new grid points between adjacent points in the original grid. An example is illustrated in Fig. 3.2 for $k = 1$. After the enlargement, each edge is now replaced by a path of k vertices (which we distinguish from the original vertices by calling them *marks*). Two adjacent vertices in the original graph are now at a distance $k + 1$, which generates a $k/2 \times k/2$ area around each vertex for local transformations. This enlargement guarantees that different transformations do not interfere with each other. In this proof, it is sufficient to use $k = 100$.

Our transformations consist of 2 steps. The first step guarantees that the transformed graph is in $X\Gamma$, to satisfy Definition 2.4. The second step ensures that each edge in the original graph is replaced by an even length path after the orthogonal embedding and transformations. Together, these steps transform the independent set problem on the cubic planar graph to an independent set problem on a graph in $X\Gamma$, and we can then conclude the NP-completeness of the MNDB problem using Lemma 2.1.

Since the underlying graph is cubic, its orthogonal embedding can be decomposed into paths, bends (illustrated in Fig. 3.3 (left)), and *T- junctions* (illustrated in Fig. 3.4 (left)). Bends are marks for which an edge changes direction, and T-junctions are the actual vertices of the cubic planar graph. While bends and T-junctions require transformations to convert the embedded graph into a graph in $X\Gamma$, paths will not cause such problems.

Consider a bend v_{ij} connected to two other marks v_{i-1j} and v_{ij+1}. In a graph in $X\Gamma$, there must be an edge between v_{i-1j} and v_{ij+1}. We can remove v_{ij}, and connect v_{i-1j} and v_{ij+1} as in Fig. 3.3 (right).

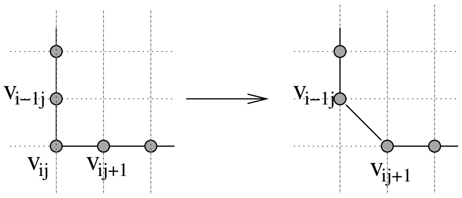

FIG. 3.3. *Bend transformation*

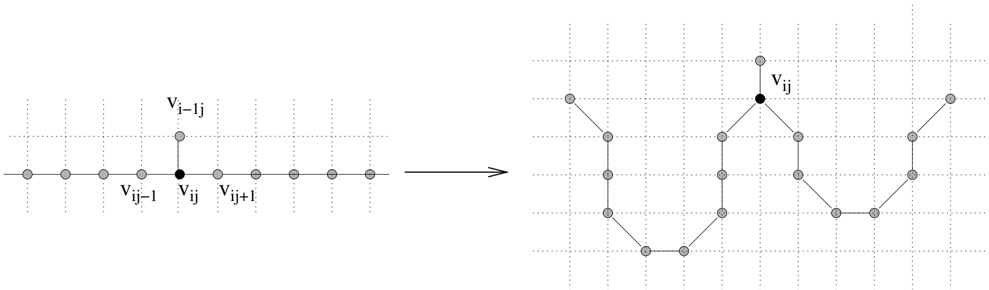

FIG. 3.4. *T-junction transformation*

Now consider a T-junction with vertex v_{ij} at the center, as illustrated in Fig. 3.4. The neighborhood of v_{ij} consists of (up to a rotation) v_{ij-1}, v_{ij+1}, and $v_{i-1\ j}$, none of which is a vertex in the original graph. As in the case of a bend, the problem is the absence of edges between v_{ij-1} and $v_{i-1\ j}$, and between $v_{i-1\ j}$ and v_{ij+1}, for which the associated blocks overlap. Also, v_{ij} must be a vertex of the original graph, and cannot be eliminated. We can make the transformation illustrated in Fig. 3.4. However, the resulting graph is still not in $X\Gamma$, since it has missing vertices and does not satisfy the closure property. We use Corollary 3.2 to add vertices to the graph as depicted in Fig. 3.1 (in reverse order, from (c) to (a)), so that the resulting graph is in $X\Gamma$.

By Lemma 3.3, we need each path replacing an edge of the planar graph to have even length. Because of the extra space we created for our local transformations, for each edge going through an odd number of marks there is a straight line segment going through at least 7 marks. We replace this 7 vertex segment with an 8 vertex segment, as illustrated in Fig. 3.5, to guarantee that each edge is replaced with an even length path.

These polynomial time transformations reduce the independent set problem for cubic planar graphs to an independent set problem in a graph in class $X\Gamma$. By Lemma 2.1, the independent set problem on a graph in $X\Gamma$ is equivalent to a MNDB problem in a matrix, thus concluding our proof. $\square$

Our proof is a template for the NP-completeness proofs of alternative substructures. Below, we generalize our result for arbitrary $m \times n$ blocks. In Section 5, we will use the same template to prove NP-completeness of the MNS problem for cross and diagonal blocks.

THEOREM 3.5. *Problem MNDB is NP-complete for $m \times n$ blocks for $m, n \geq 2$.*

Proof. We give a reduction from MIS on cubic planar graphs. Without loss of generality, we assume $n \geq \max\{m, 3\}$. Given a cubic planar graph $G_P = (V_P, E_P)$, we first embed the graph onto an $|V_P| \times |V_P|$ grid and then enlarge this grid by $k = 100$ to get G_s. This allows our local transformations to be mutually disjoint. For clarity of presentation, in this proof we use $v(i, j)$ to refer to v_{ij}. In G_s, we transform each T-junction that has two vertical edges to a T-junction with two horizontal edges, as illustrated in Fig. 3.6(a). Then by using

ETNA
Kent State University
etna@mcs.kent.edu

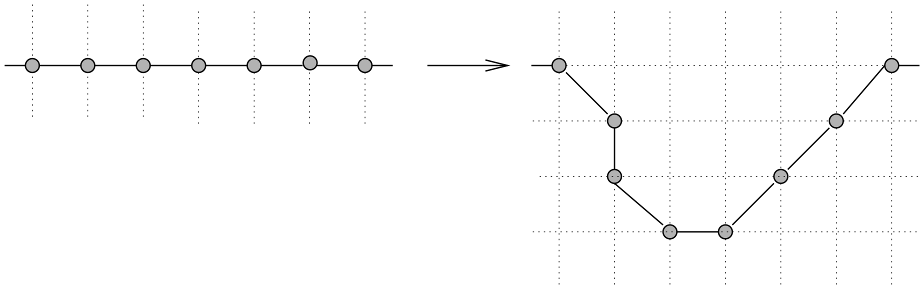

FIG. 3.5. *Odd-to-even length transformation to preserve independent set characteristics.*

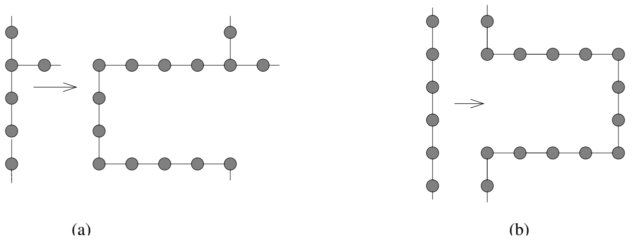

FIG. 3.6. *Transformations on G_s. (a) replace a T-junction with two vertical edges with another with two horizontal edges. (b) add a horizontal edge to each path in G_P.*

the transformation in Fig. 3.6(b), we make sure that each path replacing an edge in G_P has at least one horizontal edge away from bends and T-junctions.

In the next step, we map G_s to a larger $4mM \times 4nN$ grid G_L, so that $v(i, j)$ on the small grid is mapped to $v(i(4m-2), 4j(n-1))$ on the larger grid. This second enlargement allows us to control the overlaps, and thus define the paths of the graph. For each horizontal edge $(v(i, j), v(i, j+1))$ in the enlarged G_s, we add vertices $v(i(4m-2)+m-1, (4j+1)(n-1))$,$v(i(4m-2)+2(m-1), (4j+2)(n-1))$, and $v(i(4m-2)+(m-1), (4j+3)(n-1))$, as illustrated in Fig. 3.7(a). A similar transformation is illustrated in Fig. 3.8 (a) for vertical edges. We use different transformations for horizontal and vertical edges, since m might be 2. To avoid problems due to bends, we use the mirror images of the transformation in Figs. 3.7(a) and 3.8(a), as illustrated in, respectively, Figs. 3.7(b) and 3.8(b).

Due to our transformation in G_s, we only have T-junctions with two horizontal edges. For a T-junction with a "downward" vertical edge, we can use transformation in Fig. 3.8(a) and mirror images of transformations in Figs. 3.7(a), as illustrated in Fig. 3.9(a). For a T-junction with an "upward" vertical edge, we use the transformations in Figs. 3.8(b) and 3.7(a), as illustrated in Fig. 3.9(b). Due to our initial enlargement to obtain G_s, all these transformations will be mutually disjoint.

We define the edge set of G_L, so that it is an induced subgraph of X_{mn}. The closure property is satisfied by construction, thus G_L is in $X\Gamma_{mn}$. The reduction will be complete when we guarantee that each edge in the original 3-planar graph G_P is replaced by an even-length path in G_L. If an edge in G_P is replaced by an odd-length path in G_L, we replace a horizontal edge transformation in Fig. 3.7(a) with the one in Fig. 3.10, which inserts four vertices, instead of three. We can choose this edge to be far from a bend or a T-junction to avoid unwanted overlaps. $\square$

4. Approximation Algorithms. In this section, we discuss approximation algorithms for the maximum nonoverlapping dense blocks problem. This problem has been studied un-

ETNA
Kent State University
etna@mcs.kent.edu

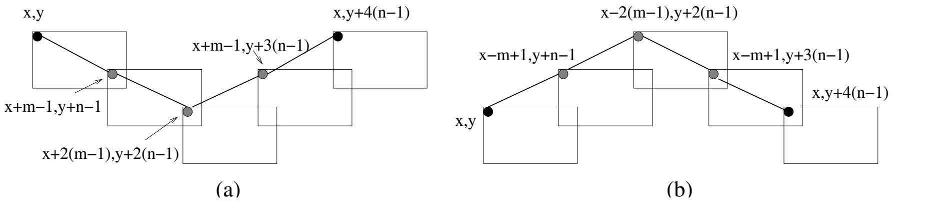

FIG. 3.7. *Replacing horizontal edges in G_s with (a) regular transformation and (b) its mirror image. Dark nodes are for original vertices, and shaded nodes correspond to auxiliary vertices to replace an edge between them. Dark edges correspond to edges in G_L, and rectangles are also drawn to illustrate overlaps.*

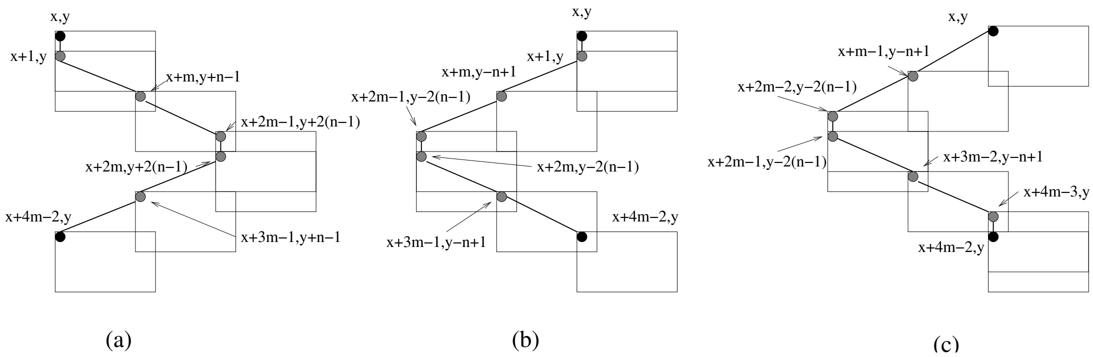

FIG. 3.8. *Two transformations to replace vertical edges in G_s. (a) The regular vertical edge transformation, (b) its mirror image and (c) version only used for an upward edge of a T-junction. Dark nodes are for original vertices, and shaded nodes correspond to auxiliary vertices to replace an edge between them. Dark edges correspond to edges in G_L, and rectangles are also drawn to illustrate overlaps.*

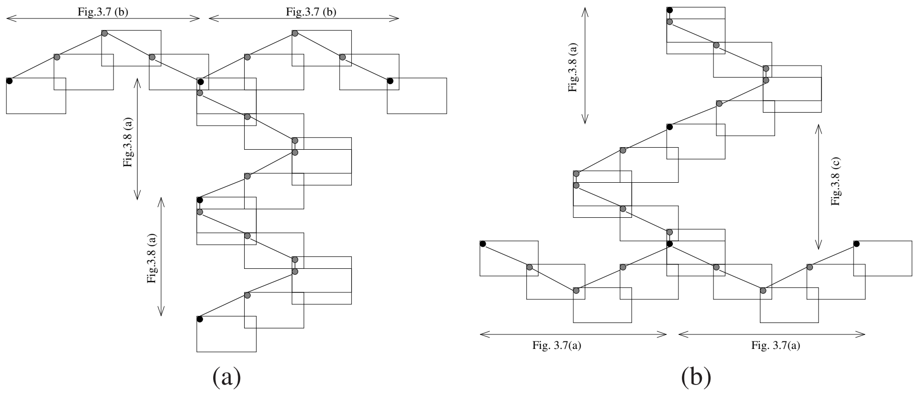

FIG. 3.9. *Transforming T-junctions (a) with an upward edge, (b) with a downward edge.*

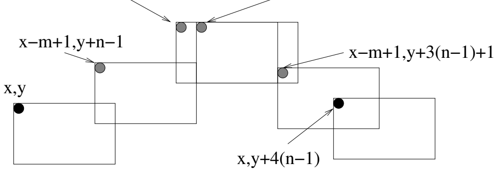

FIG. 3.10. *Even to odd length transformation.*

der different names in the literature. The optimal tile salvage problem is defined as follows. Given a $\sqrt{N} \times \sqrt{N}$ region in the plane tiled with unit squares, some of which are disfunctional, find a maximum number of functional $m \times n$ rectangles (in any orientation). This problem is equivalent to MNDB for square dense blocks. Berman *et al.* [3] describe a polynomial time approximation scheme for the optimal tile salvage problem, *i.e.* for any $\delta > 0$, $\epsilon = O(1/\sqrt{\delta \log M})$, an $(1-\epsilon)$-approximation algorithm running in time polynomial in N and exponential in δ. Here M is the optimal solution value. Their algorithm is based on maximum planar H-matching which runs in $O(N^{1+\delta})$ steps for $\delta > 0$. Baker [2] also has an algorithm for square dense blocks, which runs in $O(8^k N)$-time and $O(4^k N)$ space and produces a $(k-1)/k$-approximation. Hochbaum and Maass also describe an algorithm for square blocks that gives an $(k-1)/k$-approximation, but runs in $O(m^2 k^2 N^{k^2})$ time to find $m \times m$ blocks on an $N \times N$ grid [8]. While these algorithms are asymptotically efficient, their practicality will be limited for our purposes. We need algorithms that are extremely fast and require very limited extra memory, since our methods will be used in a preprocessing phase, which may appear as late as the application runtime, and their runtimes need to be amortized by the speedup in subsequent operations.

Arikati *et al.* [1] study this problem as the two-dimensional pattern matching problem, and describe an approximation scheme inspired by the Lipton-Tarjan method of computing approximate independent sets in graphs. Their algorithm runs in $O(N \lg N)$ time and produces solutions that are only $O(1/\sqrt{\log\log N})$ away from an optimal solution. They describe another algorithm that uses the shifting strategy of Baker [2] and Hochbaum and Maas [8]. Their algorithm decomposes the matrix into supercolumns of width $n-1$, and then for each i, $0 \le i \le k$, the problem is separated into disconnected subproblems by removing supercolumns with numbers congruent to $i \mod (\mathrm{k}+1)$. Each subproblem can be solved optimally in linear time, by algorithms that find a maximum independent set in tree-width bounded graphs [5, 12]. Arikati *et al.* show that using this they can obtain a solution which is within $\frac{k}{k+1}$ of the optimal.

The special case for $k = 2$ of the Arikati *et al.* algorithm was also pointed out to us by one of the reviewers. The algorithm can be summarized as follows. Given an input I to $(2,2)-$MNDB, construct three new instances I_0, I_1, I_2 such that instance I_j contains all blocks from I except those with upper row index $j \mod 3$. Each instance I_j can then be solved optimally in linear time. Consider an optimal solution B to I. Every subset of B included in I_j is a solution to I_j, and since each block from B is removed from exactly one of the three new instances, some instance I_j must include at least $2/3$ of the blocks in B. Therefore returning the maximum of the optimum solutions to I_0, I_1, I_2 gives at least a $2/3-$approximation. This elegant algorithm gives the same running time and approximation ratio as the algorithm presented in this paper. Nevertheless, our $2/3-$approximation algorithm can be implemented to use slightly less extra space since it only needs to maintain one independent set instead of three.

We begin by presenting a simple linear time $1/2$-approximation to the MNDB problem with 2×2 blocks which can be generalized for all σ-substructures we present. The algorithm proceeds by finding the leftmost block in the topmost row, adding it to the current independent set, and then repeating the same operation after removing this vertex and all its neighbors. At most two of the vertices can be independent among those removed from the graph, and so we have a $1/2$-approximation algorithm. In this section we show how to improve this approximation result by looking at an extended neighborhood of the leftmost vertex in the uppermost row. Our algorithm is based on choosing a set of vertices in the neighborhood of the leftmost vertex in the uppermost row, so that the size of this set is no less than $2/3$ of a maximum independent set in the induced subgraph of those vertices removed from the graph.

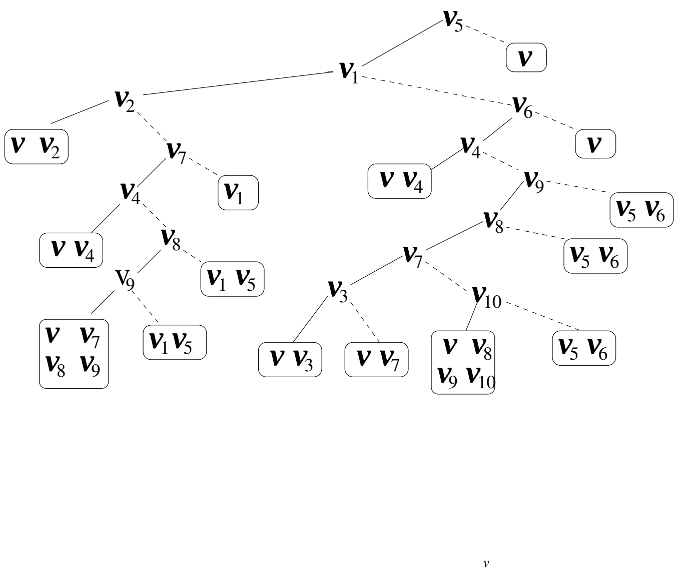

FIG. 4.1. *Decision tree for algorithm MNDB-APX. v corresponds to the leftmost vertex in the uppermost row, and the neighboring vertices in the X-grid are marked in Fig. 4.2. We take the left branch if the label vertex is in V, and the right branch otherwise. We proceed until we reach a leaf, which contains the set S that will be added in the independent set.*

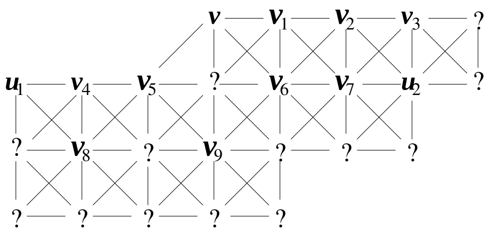

FIG. 4.2. *Vertex neighborhood considered for each call to BinTreeDecision. The positions v_i are used in the decision tree, while the positions u_i are only used in the analysis.*

This generates a final solution that is $2/3$ of the optimal, since all greedy decisions are at least $2/3$ of the local optimal. By Lemma 2.2, the graph after removing a vertex along with all its neighbors still has the intersection graph characteristics of the original by Lemma 2.2.

We present the pseudocode of our algorithm below. The algorithm is based on the procedure **BinTreeDecision**, which is depicted as a binary decision tree in Fig. 4.1. In this tree, internal nodes indicate conditions, and the leaves list the vertices added to the independent set. Our algorithm traverses this decision tree from the root to a leaf, taking the left branch if the label vertex is in V, and the right branch otherwise. For instance, at the root of the tree, we will take the left branch v_5 is in the graph, and the right branch if it is not. The leaves contain the sets S that will be added in the independent set.

LEMMA 4.1. *Algorithm MNDB-APX runs in linear time in the number of blocks in the matrix.*

Proof. Each iteration of the algorithm requires a traversal of the binary decision tree from the root to a leaf, which takes at most 8 steps, thus $O(1)$ time. Also at least one vertex is removed from the graph at each iteration. Thus the time for the decision process is linear in the number of vertices in the graph. The only other operation that affects the cost is finding

ETNA
Kent State University
etna@mcs.kent.edu

the leftmost vertex in the uppermost row. In a preprocessing step one can go through the matrix in a left to right fashion and store pointers to the blocks so that v_{ij} appears before v_{kl} iff $i < k$ or $i = k$ and $j < l$. After this it takes constant time to find the current leftmost vertex on the uppermost row. ∎

Algorithm MNDB-APX
$I \leftarrow \emptyset$
  while $V \neq \emptyset$
    $v \leftarrow$ leftmost vertex on the uppermost row
    $S \leftarrow$ **BinTreeDecision**(v)
    $I \leftarrow I \cup S$
    remove S and its neighborhood from G
  endwhile
return I

LEMMA 4.2. *The size of the maximal independent set returned by Algorithm MNDB-APX is no smaller than* $2/3$ *of the size of maximum independent set on the intersection graph.*

Proof. The proof is based on case by case analysis. We show that **BinTreeDecision**(v) of Fig. 4.1 always returns an independent set S such that $N(S)$ contains no independent set larger than $1.5\,|S|$, where $N(S)$ denotes the neighborhood of S, i.e., the set of vertices in S or adjacent to a vertex in S. Below we examine the binary search tree case by case:

- $\underline{v_5 \notin V}$ $S = \{v\}$, and v and its neighbors form a clique with MIS size 1.
- $\underline{v_5 \in V}$
  - $\underline{v_1 \notin V}$ By the closure property $v_2 \notin V$, and we have the following:
    - $\underline{v_6 \notin V}$ $S = \{v\}$, and v and its neighbors form a clique with MIS size 1.
    - $\underline{v_6 \in V}$
      - $\underline{v_4 \in V}$ $S = \{v, v_4\}$, and $N(S)$ has MIS size at most 3.
      - $\underline{v_4 \notin V}$ By the closure property $u_1 \notin V$. In this case, if one of v_9 or v_8 is not in V, then $S = \{v_5, v_6\}$, since their neighborhood has MIS size at most 3. Otherwise, $v_8, v_9 \in V$:
        - $\underline{v_7 \notin V}$ This implies $u_2 \notin V$ and:
          - $\underline{v_{10} \notin V}$ $S = \{v_5, v_6\}$ and $N(S)$ has MIS size at most 3.
          - $\underline{v_{10} \in V}$ $S = \{v, v_8, v_9, v_{10}\}$, and $N(S)$ has MIS size at most 6.
        - $\underline{v_7 \in V}$
          - $\underline{v_3 \in V}$ $S = \{v, v_3\}$, and $N(S)$ has MIS size at most 3.
          - $\underline{v_3 \notin V}$ $S = \{v, v_7\}$, and $N(S)$ has MIS size at most 3.
  - $\underline{v_1 \in V}$
    - $\underline{v_2 \in V}$ $S = \{v, v_2\}$, and $N(S)$ has MIS size at most 3.
    - $\underline{v_2 \notin V}$ By the closure property $v_3 \notin V$, and
      - $\underline{v_7 \notin V}$ $S = \{v_1\}$, v_1 and its neighbors form a clique, and the MIS is of size 1.
      - $\underline{v_7 \in V}$
        - $\underline{v_4 \in V}$ $S = \{v, v_4\}$, and $N(S)$ has MIS size at most 3.
        - $\underline{v_4 \notin V}$ By the closure property $u_1 \notin V$, and if one of v_8 or v_9 is not in V, then $S = \{v_1, v_5\}$, and $N(S)$ has a MIS size at most 3. Otherwise if $v_8, v_9 \in V$, then $S = \{v, v_7, v_8, v_9\}$, and $N(S)$ has MIS size at most 6.

∎

THEOREM 4.3. *Algorithm MNDB-APX is a linear time,* $2/3$*-approximation algorithm*

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \qquad \begin{pmatrix} & & a_{02} \\ & a_{01} & a_{12} \\ a_{00} & a_{11} & a_{22} \\ a_{10} & a_{21} & \\ a_{20} & & \end{pmatrix} \qquad \begin{pmatrix} a_{00} & & \\ a_{10} & a_{01} & \\ a_{20} & a_{11} & a_{02} \\ & a_{21} & a_{12} \\ & & a_{22} \end{pmatrix}$$

(a) (b) (c)

FIG. 5.1. *Matrix rotations. (a) the original matrix, (b) after Rotation 1, (c) after Rotation 2.*

for the MNDB problem.

Proof. Follows directly from Lemma 4.1 and Lemma 4.2. ∎

A generalization of our 2/3-approximation algorithm for larger blocks is still under investigation. We expect the runtime and the approximation ratio to depend on the block size.

5. Alternative Substructures. We have so far focused our discussions on finding dense rectangular blocks in a matrix. In this section, we will discuss generalizations of our results to alternative substructures that might be exploited to improve memory performance. We will first discuss diagonal blocks. Then we will introduce a *cross* substructure and its variants, and prove that the MNS problem is NP-complete for finding these substructures.

5.1. Diagonal Blocks. In many applications, nonzeros of the sparse matrix are lined around the main diagonal in the form of long diagonals. This makes diagonal blocks a nice alternative to rectangular blocks. We define a diagonal block as follows. Given an $M \times N$ matrix $A = (a(i,j))$, we say $d(i,j)$ is an $m \times n$ diagonal block in A iff

$$\forall k,l;\ \ i \le l < i+m;\ 0 \le k < n;\ \ a(l+k, j+k) \ne 0.$$

To find diagonal blocks in a sparse matrix, we can rotate the positions of the matrix entries to transform diagonal blocks to rectangular blocks and vice versa, so that our results for rectangular blocks can be applied to diagonal blocks. Our rotations are depicted in Fig. 5.1, and defined as follows.

Rotation 1: *Given an $M \times N$ matrix A, its rotated matrix A_R is an $(M+N-1) \times N$ matrix so that*

- $A_R(i+N-j-1, j) = A(i,j)$ for $i = 0,1,\dots M-1$ and $j = 0,1,\dots N-1$.
- All other entries of A_R are 0.

Rotation 2: *Given an $M \times N$ matrix A, its rotated matrix A_R is an $(M+N-1) \times N$ matrix so that*

- $A_R(i+j, j) = A(i,j)$ for $i = 0,1,\dots M-1$ and $j = 0,1,\dots N-1$.
- All other entries of A_R are 0.

THEOREM 5.1. *Given matrix A, let A_1 and A_2 be its rotated matrices under Rotation 1 and Rotation 2, respectively. $d(i,j)$ is a diagonal block in A, if and only if $d(i+N-j-1, j)$ is a rectangular block in A_1, and $d(i,j)$ is a rectangular block in A, if and only if $d(i+j, j)$ is a diagonal block in A_2*

Proof. By definition $d(i,j)$ is a diagonal block in A if and only if for all k,l: $0 \le k < m$; $0 \le l < n$, $A(i+k, j+l) \ne 0$. This translates to $A_1(i+k+N-j-l-1, j+l) \ne 0$ with Rotation 1, and $A_2(i+k+j+l, j+l) \ne 0$. Necessity follows from the definition of a diagonal block, and sufficiency follows from the fact that the only nonzeros in A_1 and A_2 are those defined by nonzeros in A. ∎

COROLLARY 5.2. *Algorithm MNDB-APX, composed with Rotation 1, is a linear time*

ETNA
Kent State University
etna@mcs.kent.edu

$$\begin{pmatrix} & x & \\ x & x & x \\ & x & \end{pmatrix} \qquad \begin{pmatrix} x & & x \\ & x & \\ x & & x \end{pmatrix} \qquad \begin{pmatrix} x & x & \\ & x & \\ & x & x \end{pmatrix}$$

(a) (b) (c)

FIG. 5.2. *(a) Cross block, (b) diagonal cross block, (c) jagged cross block*

$2/3$-*approximation algorithm for the problem of finding a maximum number of nonoverlapping diagonal blocks.*

COROLLARY 5.3. *Given a matrix A and a positive integer K, deciding if A has at least K nonoverlapping diagonal blocks is NP-complete.*

5.2. Cross Blocks. Various regular substructures in a sparse matrix can be exploited to improve memory performance of sparse matrix computations. One possibility is the cross blocks depicted in Fig. 5.2(a). We will identify a cross block with its center, that is, we say $c(i,j)$ is a cross block in a matrix A if A has nonzeros at positions (i,j), $(i,j-1)$,$(i-1,j)$, $(i,j+1)$, and $(i+1,j)$. Below, we prove that finding a maximum number of nonoverlapping cross blocks is NP-complete by using our proof of Theorem 3.4 as a template.

THEOREM 5.4. *Given a matrix A and a positive integer K, deciding if A has at least K nonoverlapping cross blocks is NP-complete.*

Proof. This problem can be reduced to the independent set problem, and thus it is in NP. For the NP-completeness proof we use a reduction from the independent set problem on cubic planar graphs. First we embed the cubic planar graph onto a grid and then enlarge the grid as we did for the proof of Theorem 3.4. We can replace each vertex on this grid with a cross pattern in the matrix. Formally, for an $M \times N$ grid, we define a $2M+1 \times 2N+1$ matrix, where grid point (i,j) is replaced by a cross centered at $(2i+1, 2j+1)$ in the matrix. A does not have any other nonzeros besides those in cross blocks corresponding to vertex points. There are no cross blocks in A, besides those representing grid points. Also observe that unlike the case for rectangular blocks, bends and T-junctions do not cause any problems, since the crosses to the left and below the corner vertex of a bend do not overlap.

The only problem is to make sure each edge in G is replaced by an even length path, for which we use the transformation in Fig. 5.3. This transformation replaces a chain of odd length with a chain of even length to guarantee each edge in G is replaced with even length paths. $\square$

We can use matrix rotations to reduce the problems of finding other blocks in Fig. 5.2 (b) and (c) to the problem of finding cross blocks as in Fig. 5.2(a). For instance, Rotation 1 transforms jagged crosses, which are illustrated in Fig. 5.2(c) to regular crosses, and $c(i,j)$ is a diagonal cross block in an $M \times N$ matrix, iff $c(i+N-j-1, j)$ is a jagged cross block (Fig. 5.2(c)) in its rotated matrix. Similar transformations can be used to transform cross blocks to other jagged blocks, and vice versa.

Rotation 3, as defined below and depicted in Fig. 5.4, transforms diagonal cross blocks of Fig. 5.2(b) to regular cross blocks.

Rotation 3: *Given an $M \times N$ matrix A, its rotated matrix A_R is an $(M+N-1) \times (M+N-1)$ matrix so that*

- $A_R(i-j+N-1, i+j) = A(i,j)$ for $i = 0, 1, \ldots M-1$ and $j = 0, 1, \ldots N-1$.
- All other entries of A_R are 0.

These transformations can be used to prove NP-completeness of deciding if there are a specified number of nonoverlapping jagged and diagonal cross blocks in a matrix. For brevity, we are not giving the details here. As an approximation solution, the greedy algorithm that

ETNA
Kent State University
etna@mcs.kent.edu

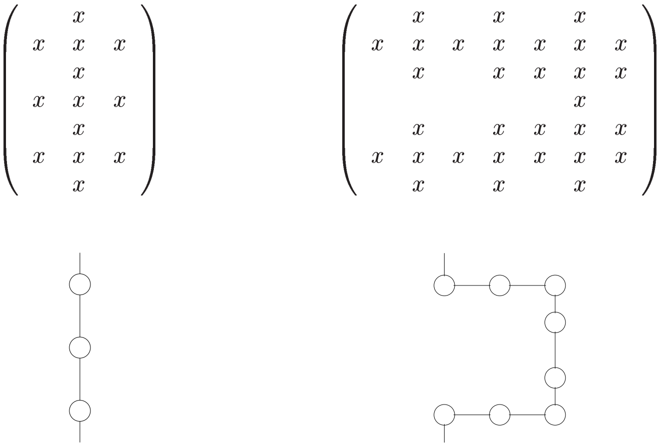

FIG. 5.3. *Odd- to even-length path transformation for cross blocks.*

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \qquad \begin{pmatrix} & & a_{02} & & \\ & a_{01} & & a_{12} & \\ a_{00} & & a_{11} & & a_{22} \\ & a_{10} & & a_{21} & \\ & & a_{20} & & \end{pmatrix}$$

(a) (b)

FIG. 5.4. *Matrix Rotations. (a) the original matrix, (b) after Rotation 3.*

chooses the leftmost block in the upper most row will yield a $1/2-$approximation algorithm for finding cross blocks and all its variations.

6. Open Problems. This work studies a new problem for the sparse matrix computations community, and brings forth many open problems. One interesting family of problems is the design of heuristics for larger blocks and different substructures, and developing better approximation algorithms. As we discussed in Section 4, it may be possible to generalize our $2/3$-approximation algorithm for larger blocks, where the runtime complexity is likely to depend on the block size. Another open problem is whether one can improve the approximation ratio by looking at a larger neighborhood of the leftmost vertex of the uppermost row. Finally, one may search for different heuristics to apply to different dense substructure problems. For instance, although the greedy left-uppermost block heuristic still gives a $1/2$-approximation, the neighborhood structure of the cross block is fairly different from that of the rectangular block, and thus our $2/3$-approximation algorithm cannot be applied directly.

Another approach to reducing memory indirection is selectively replacing structural zeros of the matrix with numerical zeros. Doing this would improve memory performance and may result in significant speedups, even though the number of floating point operations may increase [15]. This technique calls for another interesting combinatorial problem. In this case, we need to choose blocks to make sure all nonzeros are covered, and we try to do this by using as few blocks as possible. We call this problem the *minimum block cover problem* and define it as follows.

> *Given a sparse matrix A, and an oriented substructure α, place a minimum number of substructures on A, so that all its nonzeros are covered.*

Fowler *et al.* [6] proved that this problem is NP-complete. Nevertheless, good approxi-

ETNA
Kent State University
etna@mcs.kent.edu

mation algorithms for covering sparse matrices would be valuable.

Finally, in this paper we considered finding only one specified structure in the matrix. However, it is possible to split a matrix into three or more matrices (e.g., $A = A_d^2 + A_d^1 + A_s$), so that each matrix contains a different substructure. Vuduc did some empirical work on splitting into multiple matrices [14]. In such a decomposition, the objective is minimizing the total number of blocks in all matrices. Clearly, this problem is much harder, and even good approximation algorithms (provably or practically) would be valuable.

7. Conclusion. We studied the problem of finding maximum number of nonoverlapping substructures in a sparse matrix, which we called the *maximum nonoverlapping substructures* problem. Such substructures can be exploited to improve memory performance of sparse matrix operations by reducing the number of memory indirections. We focused on $m \times n$ dense blocks as a substructure (maximum nonoverlapping dense blocks problem) due to their frequency in sparse matrices arising in various applications, and to their effectiveness in decreasing extra load operations. We investigated the relation between the maximum independent set problem and the maximum nonoverlapping substructures problem, and defined a class of graphs where the independent set problem is equivalent to the maximum nonoverlapping dense blocks problem. We used this relation to prove the NP-completeness of the maximum nonoverlapping dense blocks problem. Our proof used a reduction from the maximum independent set problem on cubic planar graphs and adopted orthogonal drawings of planar graphs. We discussed generalizations of our results to alternative substructures and observed the relation between diagonal and rectangular blocks to show that the two MNS problems are equivalent and one can be reduced to the other by a matrix transformation. We also discussed cross blocks and proved that the MNS problem is NP-complete for cross blocks.

We presented an approximation algorithm for the maximum nonoverlapping dense blocks problem for 2×2 blocks. Our algorithm requires only linear time and space, generates solutions whose sizes are within $2/3$ of the optimal, and can be used to approximate MNS on diagonal blocks as well.

Acknowledgments. We are grateful to the three anonymous referees for their valuable comments on the earlier version of this paper.

REFERENCES

[1] S. R. ARIKATI, A. DESSMARK, A. LINGAS, AND M. V. MARATHE, *Approximation algorithms for maximum two-dimensional pattern matching*, Theoret. Comput. Sci., 255 (2001), pp. 51–62.

[2] B. BAKER, *Approximation algorithms for np-complete problems on planar graphs*, in Proc. 24th IEEE Symp. on Foundations of Computer Science, 1983, pp. 265–273.

[3] F. BERMAN, D. JOHNSON, T. LEIGHTON, P. SHOR, AND L. SNYDER, *Generalized planar matching*, J. Algorithms, 11 (1990), pp. 153–184.

[4] A. J. C. BIK AND H. A. G. WIJSHOFF, *Automatic nonzero structure analysis*, SIAM J. Comput., 28 (1999), pp. 1576–1587.

[5] H. L. BODLAENDER, *Dynamic programming on graphs of bounded treewidth*, Lecture Notes in Comput. Sci., 317, Springer, Berlin,(1988), pp. 105–118.

[6] R. J. FOWLER, M. S. PATERSON, AND S. L. TANIMOTO, *Optimal packing and covering in the plane are np-complete*, Inform. Process. Lett., 12 (1981), pp. 133–137.

[7] M. R. GAREY AND D. S. JOHNSON, *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences.*, W.H. Freeman and Company, San Francisco, Calif., 1979.

[8] D. S. HOCHBAUM AND W. MAASS, *Approximation schemes for covering and packing problems in image processing and vlsi*, J. Assoc. Comput. Mach., 32 (1985), pp. 130–136.

[9] E. IM, K. YELICK, AND R. VUDUC, *Sparsity: An optimization framework for sparse matrix kernels*, Int. J. High Performance Comput. Appl., 18 (2004).

[10] G. KANT, *Drawing planar graphs using the canonical ordering*, Algorithmica, 16 (1996), pp. 4–32.

[11] A. PINAR AND M. HEATH, *Improving performance of sparse matrix vector multiplication*, in Proc. IEEE/ACM Conf. on Supercomputing, 1999.

[12] N. ROBERTSON AND P. SEYMOUR, *Graph minors ii. algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[13] S. TOLEDO, *Improving the memory-system performance of sparse matrix vector multiplication*, IBM Journal of Research and Development, 41 (1997).

[14] R. VUDUC, *Automatic performance tuning of sparse matrix kernels*, PhD thesis, University of California, Berkeley, December 2003.

[15] R. VUDUC, J. DEMMEL, K. YELICK, S. KAMIL, R. NISHTALA, AND B. LEE, *Performance optimizations and bounds for sparse matrix-vector multiply*, in Proc. IEEE/ACM Conf. on Supercomputing, 2002.