SIAM J. Sci. Comput.
Vol. 29, No. 2, pp. 827–841

GEOMETRIC OPTIMIZATION OF THE EVALUATION OF FINITE ELEMENT MATRICES*

ROBERT C. KIRBY† AND L. RIDGWAY SCOTT‡

Abstract. This paper continues earlier work on mathematical techniques for generating optimized algorithms for computing finite element stiffness matrices. These techniques start from representing the stiffness matrix for an affine element as a collection of contractions between *reference tensors* and an element-dependent *geometry tensor*. We go beyond the complexity-reducing binary relations explored in [R. C. Kirby, A. Logg, L. R. Scott, and A. R. Terrel, *SIAM J. Sci. Comput.*, 28 (2006), pp. 224–240] to consider geometric relationships between three or more objects. Algorithms based on these relationships often have even fewer operations than those based on complexity-reducing relations.

Key words. finite element, variational form, optimized algorithm, graph theory

AMS subject classifications. 65N30, 65Y20, 68R05, 68R10

DOI. 10.1137/060660722

1. Introduction.

In our recent work, we have presented mathematical and computational techniques for the automatic generation of optimized code for evaluating local stiffness matrices for finite elements. These techniques are applicable to multilinear variational forms over affine elements using general basis functions. Beginning in [10], we suggested that the local evaluation of the stiffness matrix for multilinear forms for a single affine element could be written as contractions of a set of *reference tensors*, with a single tensor encoding the geometry and coefficients.

This formulation allows us to generate code for the innermost part of a finite element computation. The FEniCS form compiler, FFC [11, 12], takes a variational form and approximating space as input. As output, it produces a function that maps the geometry and material coefficients for any element in an unstructured mesh to the element stiffness matrix. It also generates the higher-level code that loops over all elements in a mesh, computing the element stiffness matrices and inserting them into a global sparse matrix. The current implementation of FFC is fully functional for multilinear forms with arbitrary order Lagrange elements, and works for general unstructured meshes in one, two, or three space dimensions. In [12], we show how the technique can be applied to more general variational forms, curved elements, and $H(\mathrm{div})$ or $H(\mathrm{curl})$ elements. Although our techniques can be applied to quadrilateral or hexahedral meshes, spectral element techniques that work dimension by dimension are probably more effective.

The innermost computational kernel of the FFC-generated code is the collection of tensor contractions on each element of a mesh. Since this code is a single function executed over all elements of a mesh, it makes sense to find optimizations that allow us to perform the computations for an arbitrary element as efficiently as possible.

*Received by the editors May 24, 2006; accepted for publication (in revised form) October 11, 2006; published electronically April 24, 2007.
http://www.siam.org/journals/sisc/29-2/66072.html

†Department of Computer Science, University of Chicago, Chicago, IL 60637-1581 (robert.c.kirby@ttu.edu). This author's work was supported by the U.S. Department of Energy Early Career Principal Investigator Program under award DE-FG02-04ER25650.

‡The Computation Institute and Departments of Computer Science and Mathematics, University of Chicago, Chicago, IL 60637-1581 (ridg@cs.uchicago.edu).

In [10], we noticed that relations exist between the reference tensors that allow one to perform all the contractions needed to compute a single element matrix in many fewer arithmetic operations than would otherwise seem necessary. Our rather crude heuristic for exploiting these dependencies allowed us to generate code that was notably faster than the standard method for forming local stiffness matrices on our test problems.

Then, in [13], we formalized part of the optimization process in [10] by introducing the concept of *complexity-reducing relations*. These are derived from metrics that model the cost of computing one of the tensor contractions once another contraction has been computed. This theory effectively translates optimizations based on pairwise dependencies between the reference tensors into a minimum spanning tree. While the strategy is quite effective at reducing the operation count for many variational forms, it does not address how to utilize relations between three or more reference tensors, such as when one reference tensor may be written as a linear combination of two others. In this paper, we study efficient algorithms for detecting such geometric relations and hence finding fast algorithms for building finite element stiffness matrices.

Our attempts to mechanize the optimization of evaluating finite element matrices bear strong resemblance to projects in other areas. In [2], a tool is developed for synthesizing high-performance FORTRAN code for high-dimensional structured tensor contractions arising in quantum chemistry. The authors search for ways of reducing operation counts by heuristics for common subexpression elimination. While our techniques seem to search for a wider class of arithmetic dependencies, their work pushes harder on data locality and other architecture and performance-related issues. Other attempts to mathematize the derivation of efficient numerical code include the FLAME project of van de Geijn and coworkers for dense linear algebra [8, 3] and the SMART project of Egner and Püschel in signal processing [6]. The latter project has developed a code that takes a (symbolic representation of) a matrix such as the discrete Fourier transform as input and factors it via representation theory into a short sequence of sparse structured matrices. Their techniques automatically replicate the Cooley–Tukey algorithm [4], but also find fast algorithms for other, more complicated, transforms. Many other projects, such as ATLAS [20] and SPIRAL [19], attempt to map a given algorithm as closely as possible to a given computer architecture.

We begin the remainder of this paper by reviewing the formulation of the local matrix computation as tensor contractions and stating the abstract computational problem to optimize in section 2. Then, we study our geometric optimization process in section 3. This includes both an expected quadratic-time algorithm for detecting coplanarity among a set of input vectors as well as using such relations to speed up the finite element computation. These ideas generalize readily to higher-dimensional dependencies, which we do at the end of this section. In section 4, we study the actual reduction of operation count generated by our algorithms. In many cases for both the Laplacian and weighted Laplacian, it gives reduction in arithmetic superior to that of complexity-reducing relations. This leads us to consider how the two approaches can be combined into a single optimization process. While we do not have a good answer to this yet, we discuss a simple heuristic and also phrase the issue as a combinatorial optimization problem in section 5. Throughout the paper, readers familiar with the subjects of finite linear spaces and ordered ternary relations may find many of the concepts somewhat familiar. To focus on the exposition of the algorithms and application to finite element computation, we defer these issues and make several remarks on these relationships in section 6.

2. Background.

As in previous work, we let $\{V_i\}_{i=1}^r$ be a given set of finite-dimensional function spaces defined on a triangulation $\mathcal{T} = \{e\}$ of a domain $\Omega \subset \mathbb{R}^d$. We consider multilinear forms a defined on the product space $V_1 \times V_2 \times \cdots \times V_r$:

$$a : V_1 \times V_2 \times \cdots \times V_r \to \mathbb{R}. \tag{2.1}$$

In finite element methods, a is an integral of products of (derivatives) of its arguments.

We are interested in forming the element tensor A^K for each $K \in \mathcal{T}_h$. This element tensor depends on the basis chosen for each function space. We let $\{\varphi_i^1\}_{i=1}^{M_1}, \{\varphi_i^2\}_{i=1}^{M_2}, \dots, \{\varphi_i^r\}_{i=1}^{M_r}$ be bases of $V_1, V_2, \dots, V_r$ over K, and let $i = (i_1, i_2, \dots, i_r)$ be a multi-index. The multilinear form a then defines a rank r tensor given by

$$A_i^K = a(\varphi_{i_1}^1, \varphi_{i_2}^2, \dots, \varphi_{i_r}^r). \tag{2.2}$$

For affine elements such as triangles and tetrahedra, A^K is evaluated by mapping each K to a fixed reference element $\hat{K}$. As we have written extensively about this representation elsewhere, we recall the Laplacian as an example here. For other examples and a more general representation theorem for a class of variational forms, we refer the reader to [11, 12].

We let F_K be the affine map from $\hat{K}$ to K. We denote by x the coordinates on K and by X the coordinates on $\hat{K}$. We identify $\phi_i^j = \Phi_i^j \circ F_K^{-1}$, and by the chain rule we obtain

$$\begin{aligned} A_i^K &= \int_e \nabla\varphi_{i_1}^{e,1}(x) \cdot \nabla\varphi_{i_2}^{e,2}(x)\,\mathrm{d}x \\ &= \det F_e' \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \int_E \frac{\partial \Phi_{i_1}^1(X)}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2(X)}{\partial X_{\alpha_2}}\,\mathrm{d}X = A_{i\alpha}^0 G_e^\alpha. \end{aligned} \tag{2.3}$$

In [13], we abstracted the computation of A^K to the following scenario. Let $V \subset \mathbb{R}^d$ with $|V| = n < \infty$. By default, for any $g \in \mathbb{R}^d$, we may compute $\{v^t g : v \in V\}$ in nd multiply-add pairs (hence MAPs). However, the kinds of V that arise in finite element methods often possess special structure, so that the overall process may be performed on a general g in many fewer MAPs. In order to formalize an optimization procedure based on pairs of vectors in V, we introduced the notion of *complexity-reducing relations*. These are distance measures ρ such that if $\rho(u, v) = k$, then $u^t g$ may be computed using $v^t g$ in no more than k MAPs. For example, if u, v are colinear, then we may define $\rho(u, v) = 1$, since if $u = \alpha v$, then $u^t g = \alpha(v^t g)$. However, we left unstudied how to exploit relations between three or more members of V, such as when three vectors lie in a two-dimensional subspace.

The point of this work is to formalize and employ these geometric relations in optimizing the computation of element stiffness matrices. Throughout this work, we shall present illustrations of our geometric notions on a particular set of vectors. In this case, we will work with the vectors from the Laplacian reference tensor for cubic Lagrange elements over triangles. We will use the symmetry transformation described in [13] to map from $\mathbb{R}^4$ to $\mathbb{R}^3$ and work only with the triangular part of the matrix. A table of the vectors can be found in Table 2.1. In particular, we recall that many of the vectors are zero or else equal or colinear to other vectors. Hence, the number of vectors among which we need to consider coplanar relations is actually much smaller. In Table 2.2, we show the entries of the reference tensor after removing the zero entries and all but one of any collection of equal or colinear vectors.

Table 2.1
Reference tensor entries for Laplacian with cubic polynomials on triangles.

(0, 0)	0.42500	0.85000	0.42500
(0, 1)	-0.08750	-0.08750	0.00000
(0, 2)	0.00000	-0.08750	-0.08750
(0, 3)	-0.03750	-0.07500	-0.03750
(0, 4)	-0.03750	-0.07500	-0.03750
(0, 5)	0.03750	0.37500	0.33750
(0, 6)	0.03750	-0.63750	-0.67500
(0, 7)	-0.67500	-0.63750	0.03750
(0, 8)	0.33750	0.37500	0.03750
(0, 9)	0.00000	0.00000	0.00000
(1, 1)	0.42500	0.00000	0.00000
(1, 2)	0.00000	0.08750	0.00000
(1, 3)	0.03750	0.71250	0.00000
(1, 4)	0.03750	-0.30000	0.00000
(1, 5)	-0.03750	0.00000	0.00000
(1, 6)	-0.03750	0.00000	0.00000
(1, 7)	0.33750	0.30000	0.00000
(1, 8)	-0.67500	-0.71250	0.00000
(1, 9)	0.00000	0.00000	0.00000
(2, 2)	0.00000	0.00000	0.42500
(2, 3)	0.00000	-0.30000	0.03750
(2, 4)	0.00000	0.71250	0.03750
(2, 5)	0.00000	-0.71250	-0.67500
(2, 6)	0.00000	0.30000	0.33750
(2, 7)	0.00000	0.00000	-0.03750
(2, 8)	0.00000	0.00000	-0.03750
(2, 9)	0.00000	0.00000	0.00000
(3, 3)	1.68750	1.68750	1.68750
(3, 4)	-0.33750	0.67500	-0.33750
(3, 5)	0.33750	0.33750	0.00000
(3, 6)	0.33750	0.33750	0.00000
(3, 7)	0.00000	0.33750	0.33750
(3, 8)	0.00000	-1.68750	-1.68750
(3, 9)	-2.02500	-2.02500	0.00000
(4, 4)	1.68750	1.68750	1.68750
(4, 5)	-1.68750	-1.68750	0.00000
(4, 6)	0.33750	0.33750	0.00000
(4, 7)	0.00000	0.33750	0.33750
(4, 8)	0.00000	0.33750	0.33750
(4, 9)	0.00000	-2.02500	-2.02500
(5, 5)	1.68750	1.68750	1.68750
(5, 6)	-0.33750	-1.35000	-1.35000
(5, 7)	0.00000	-0.33750	0.00000
(5, 8)	0.00000	-0.33750	0.00000
(5, 9)	0.00000	2.02500	0.00000
(6, 6)	1.68750	1.68750	1.68750
(6, 7)	0.00000	1.68750	0.00000
(6, 8)	0.00000	-0.33750	0.00000
(6, 9)	-2.02500	-2.02500	0.00000
(7, 7)	1.68750	1.68750	1.68750
(7, 8)	-1.35000	-1.35000	-0.33750
(7, 9)	0.00000	-2.02500	-2.02500
(8, 8)	1.68750	1.68750	1.68750
(8, 9)	0.00000	2.02500	0.00000
(9, 9)	4.05000	4.05000	4.05000

Table 2.2
Reference tensor entries for Laplacian using cubics on triangles, after zero entries and all but one entry from any set of equal or colinear entries have been removed.

(0, 3)	-0.03750	-0.07500	-0.03750
(0, 5)	0.03750	0.37500	0.33750
(0, 6)	0.03750	-0.63750	-0.67500
(0, 7)	-0.67500	-0.63750	0.03750
(0, 8)	0.33750	0.37500	0.03750
(1, 3)	0.03750	0.71250	0.00000
(1, 4)	0.03750	-0.30000	0.00000
(1, 6)	-0.03750	0.00000	0.00000
(1, 7)	0.33750	0.30000	0.00000
(1, 8)	-0.67500	-0.71250	0.00000
(2, 3)	0.00000	-0.30000	0.03750
(2, 4)	0.00000	0.71250	0.03750
(2, 5)	0.00000	-0.71250	-0.67500
(2, 6)	0.00000	0.30000	0.33750
(2, 8)	0.00000	0.00000	-0.03750
(3, 3)	1.68750	1.68750	1.68750
(3, 4)	-0.33750	0.67500	-0.33750
(4, 7)	0.00000	0.33750	0.33750
(5, 6)	-0.33750	-1.35000	-1.35000
(5, 9)	0.00000	2.02500	0.00000
(6, 9)	-2.02500	-2.02500	0.00000
(7, 8)	-1.35000	-1.35000	-0.33750

3. Geometric relations.

3. Geometric relations. As with complexity-reducing relations, we will consider the following abstract problem. Suppose that $\{v_i\}_{i=1}^n \equiv V \subset \mathbb{R}^d$ is a finite collection of vectors. We desire to derive a process by which $\{v_i^t g\}_{i=1}^n$ may be constructed efficiently. This process must take an arbitrary $g \in \mathbb{R}^d$ as an input. Effectively, this corresponds to a matrix-vector multiplication Ag, where the elements of V form the rows of A. Such a process may then be encoded in some low-level C or FORTRAN program and used as the innermost computation for building finite element matrices. As finding such a process is a "compile-time" process that need only be performed once per variational form and set of basis functions, one may be willing to perform a relatively expensive computation to obtain it.

If any $u, v \in V$ are equal or colinear, then $u^t g$ may be computed in zero or one operation using $v^t g$. If such relations occur in V, we need only search for coplanarity among the noncolinear elements of V. Similarly, when we search for higher-dimensional dependencies, we will want to filter out coplanarity.

DEFINITION 3.1. *$V \subset \mathbb{R}^d$ with $k+1 \leq |V| < \infty$ is k-*independent *if there exists no linearly dependent subset of $k+1$ vectors.*

DEFINITION 3.2. *Let $U \subset V$ have a k-dimensional span. Then U is k-*maximal *if there exists no $U \subset U'$ such that U and U' have the same span.*

It stands to reason that if there are many linear dependencies among the vectors, this should allow us to construct $\{v_i^t g\}_{i=1}^n$ efficiently, for if $w = \alpha u + \beta v$, then $w^t g$ may be computed in only two MAPs once $u^t g$ and $v^t g$ are known. In the rest of the section, we study a process to make thorough use of these dependencies. We first examine the case of two-dimensional dependencies, and then move on to present the general case.

Now, suppose that V is 1-independent (no two vectors are colinear or equal). Two issues must be resolved to make use of coplanar dependencies in computing the dot products. First, we must enumerate all of the 2-maximal subsets of V. Second, we must find a way to construct $\{v_i^t g\}_{i=1}^n$ in such a way that as many of the dot products are computed by using linear relations as possible. Equivalently, as few dot products as possible should be performed explicitly.

3.1. Search algorithms.

The first step one might take in finding the 2-maximal sets is obvious; loop over all triples of vectors. This process runs in $O(n^3)$ time, and the innermost check for coplanarity will typically require $O(d^3)$ operations (for example, Gaussian elimination or singular value decomposition). It then remains to determine the 2-maximal sets from the coplanar triples. To do this, let (G, E) be a graph where $G = \{\{i,j,k\} : \dim \operatorname{span}\{v_i, v_j, v_k\} = 2\}$ and an edge is drawn between $\{i,j,k\}$ and $\{i',j',k'\}$ if $|\{i,j,k\} \cap \{i',j',k'\}| = 2$. Then the connected components of this graph encode the 2-maximal sets. Actually, the vertices of a connected component are a set of sets of vector labels; one must take the union of all sets of labels in each connected component.

The 2-maximal sets may be enumerated much more effectively. By assumption, each pair of elements in V spans a plane. If we consider two pairs of vectors (these pairs may share a common vector), then we should be able to determine whether they correspond to the same plane or not.

Rather than searching for equality among the planes themselves, we will work a *necessary* condition for equality, leading to Algorithm 1 below. Let $\Pi : \mathbb{R}^d \to \mathbb{R}^3$ be some full-rank linear transformation. We compute $\{\Pi v_i\}_{i=1}^n$. Then, if $\{v_i, v_j\}$ and $\{v_{i'}, v_{j'}\}$ encode the same two-dimensional subspace, then of necessity so do $\{\Pi v_i, \Pi v_j\}$ and $\{\Pi v_{i'}, \Pi v_{j'}\}$. Conveniently, each is a pair of vectors in $\mathbb{R}^3$, so its cross product provides a vector normal to the spanned plane. If $\{v_i, v_j\}$ and $\{v_{i'}, v_{j'}\}$ span the same plane, then $n_{i,j} = \Pi v_i \times \Pi v_j$ and $n_{i',j'} = \Pi v_{i'} \times \Pi v_{j'}$ must be colinear. We have already seen in [13] that we may search for colinearity among a set of vectors in expected linear time by using hash tables. We form a table T whose keys n are the unique normal vectors to the pairs of projected vectors and whose values are sets of vector indices. Consequently, this entire process runs in expected (thanks to hashing) $O(n^2)$ time and has a lower dependence on d than the brute force algorithm. Furthermore, this hash table maps the normal vector to the set of all vectors that lie in the projected plane, so we do not need to find connected components as above.

Of course, some information is lost when mapping vectors from $\mathbb{R}^d$ down to $\mathbb{R}^3$.

It is possible that the mappings of vectors are coplanar while the vectors themselves are not. The candidates for 2-maximal sets are table values $T[n]$ containing more than two indices. We must first verify that each set of vectors is indeed 2-dependent. If it is not, we may recur to find its 2-independent subsets by selecting a new Π. In practice, however, the size of these sets is much smaller than $|V|$, so even using brute force followed by connected components is effective.

Algorithm 1. One phase of projection-based algorithm for determining 2-maximal sets. This process finds candidates for 2-maximal sets, and we verify and possibly recur on each.

```
Given $\{v_i\}_{i=1}^n \subset \mathbb{R}^d$, $n < \infty$, 1-independent.
Compute $\Pi \in \mathbb{R}^{3,d}$, a full-rank transformation.
Let $T$ be an (initially empty) hash table mapping vectors in $R^3$ to sets of integers in $[1, n]$.
**for** $i$=1 to $n$ **do**
  Compute $\Pi v_i$
**end for**
**for** $i = 1$ to $n$ **do**
  **for** $j = i + 1$ to $n$ **do**
    $n_{i,j} = \Pi v_i \times \Pi v_j$
    Normalize $n_{i,j}$ to have the unit $\infty$-norm and first nonzero entry positive.
  **end for**
**end for**
**for** $i = 1$ to $n$ **do**
  **for** $j = i + 1$ to $n$ **do**
    **if** $n_{i,j}$ is a key of $T$ **then**
      $T[n_{i,j}] \leftarrow T[n_{i,j}] \cup \{i, j\}$
    **else**
      $T[n_{i,j}] \leftarrow \{i, j\}$
    **end if**
  **end for**
**end for**
Output $T[n]$ for each key $n$ of $T$ such that $|T[n]| > 2$.
```

It is interesting to know how close to being optimal this algorithm is. To know this requires knowing just how many common planes there can be. Consider a set of vectors in three dimensions, for simplicity, in the positive orthant ($x \geq 0$, $y \geq 0$, $z \geq 0$). Now consider the projection of the vectors on the triangle T defined by

$$x + y + z = M, \quad x \geq 0, y \geq 0, z \geq 0, \tag{3.1}$$

where $M > 0$ could be arbitrary, but we will take it to be sufficiently large to simplify our notation. Three such vectors lie in a plane through the origin if and only if the projections onto T are collinear. We now construct a set of n points with $\mathcal{O}(n^2)$ common planes.

Let k be a positive integer, and consider the points in the rectangular lattice

$$(i, j), \quad i = 1, \ldots, 2k, \quad j = 1, 2, 3. \tag{3.2}$$

We see that for each point with $j = 0$ we can associate k lines going through three points, and thus there are at least $2k^2$ common planes. Figure 3.1 shows an example with $k = 4$ showing only four of the eight sets of four planes for $i = 1, 2, 3, 4$.

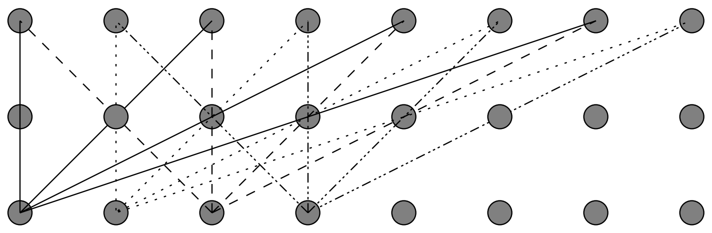

FIG. 3.1. *Example of lattice with $k = 4$. For each point on the lower line, there are exactly four planes. Only the planes for $i = 1, 2, 3, 4$ are shown.*

Since the number of planes to be determined is quadratic in the number of initial vectors, a quadratic algorithm for determining them is the best we would expect as a worst-case bound.

3.2. Generators. Now, let $\mathcal{P}$ be the set of all 2-maximal subsets of V. We would like to use V and $\mathcal{P}$ to infer an efficient process for computing $\{v_i^t g\}_{i=1}^n$ for arbitrary g. In order to do this, we will identify a (hopefully small) subset $S \subset V$ by which we may recursively construct the rest of the members of V via pairwise linear combinations. Once this construction is found, it will tell us how to efficiently compute the dot products. That is, we start with S and will compute $\{v^t g : v \in S\}$ by brute force. Then, we find all $v \in V - S$ with v in the span of two members of S. We then let S_1 be the union of all such v and S. This implies that we can compute each element of $\{v^t g : v \in S_1 - S\}$ in two MAPs. We then identify all $v \in V - S_1$ spanned by two members of S_1, letting S_2 be the union S_1 with all such v. Again, if $v \in S_2 - S_1$, then $v^t g$ may be computed in two MAPs. If S is chosen suitably, then this process will yield $S_i = V$ for some $i < \infty$. We will call such a set S a *generator*. Then, the total arithmetic cost for computing $\{v_i^t g\}_{i=1}^n$ is no greater than $d|S|$ MAPs. The goal is then to find a generator with as few elements as possible.

THEOREM 3.3. *For any $g \in \mathbb{R}^d$, the computation $\{v_i^t g\}_{i=1}^n$ may be computed in $d|S| + 2|V - S|$ MAPs, where S is a generator for V.*

We will initially develop these ideas geometrically rather than graph-theoretically, although later we will present the essential problem over a graph. However, our eventual goal for code generation is to find some sort of a *dependence graph* that tells us which vectors are constructed from which. The leaves (having no out-neighbors) are the members of S. All nodes not in S will have two out-neighbors. If v has two out-neighbors w, x, then $\{v, w, x\}$ must be 2-dependent; the computation of $v^t g$ will be performed using this linear relation rather than explicitly. A topological sorting of this graph such that each vertex follows its out-neighbors will allow all the dot products to be performed in an appropriate sequence. Straightline code for each dot product (corresponding to an entry of the element stiffness matrix) can then be generated. An example dependence graph is shown in Figure 3.2; this is for the vectors in Table 2.2 for the cubic Lagrange basis on triangles.

Without loss of generality, we will assume that every element of V is contained in at least one $P \in \mathcal{P}$. If some $v \in V$ does not lie in a plane with any two other items, then $v^t g$ must always be computed explicitly anyway. Such vertices are isolated points in a dependence graph, having no in- or out-neighbors.

We may refer to the above process of recursively taking linear combinations of S as a geometric *closure*.

DEFINITION 3.4. *Let $S \subset V$. The* geometric 2-closure *of S, denoted $\bar{S}$, is defined*

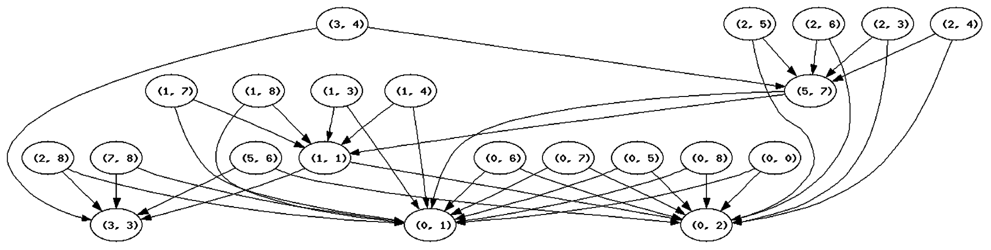

FIG. 3.2. *Generation graph for a* 1*-independent subset of vectors for the Laplacian using cubic Lagrange elements.*

as follows. First, if $v \in S$, then $v \in \bar{S}$. Second, if $v \in V$ and there exist any $w, x \in \bar{S}$ such that $\{v, w, x\}$ is 2-dependent, then $v \in \bar{S}$. We emphasize that this definition is recursive.

Remark 1. The term *closure* is appropriate, as the operation mapping S to its geometric 2-closure is an abstract closure operation. That is, the operation is increasing ($S \subseteq \bar{S}$), idempotent ($\bar{\bar{S}} = \bar{S}$), and monotone ($S \subseteq T \rightarrow \bar{S} \subseteq \bar{T}$).

DEFINITION 3.5. *If $\bar{S} = V$, then S* generates V *or is a* generator. *If no $S' \subset S$ also generates V, then S is a* minimal generator. *Finally, if no minimal generator has lower cardinality than S, then we say S is a* minimum generator.

Our terminology here is intentionally analogous to the maximal and maximum independent set problems from algorithmic graph theory. An *independent set* is a subset of a graph's vertices so that no two vertices are connected by an edge. A *maximal* independent set is an independent set not contained in any larger independent set. A *maximum* independent set is a maximal independent set with cardinality no less than any other maximal independent set. While simple efficient algorithms exist for finding a maximal independent set, finding a maximum independent set is NP-hard [9]. The algorithm we study below may have similar properties; while it might not always find a minimum generator, it finds a minimal one. The hardness of finding a minimum generator and the difference in size between the minimal generator that our algorithm finds and a minimum generator are all open questions; we will make an additional remark about this later.

Our algorithm is based on the *plane graph* of the 2-maximal sets.

DEFINITION 3.6. *Let $\mathcal{P}$ be the 2-maximal subsets of $\{v_i\}_{i=1}^n$. Then the* plane graph *$(\mathcal{P}, E)$ is a graph whose vertices are the 2-maximal sets. An edge is drawn between two 2-maximal sets P_1 and P_2 if and only if $|P_1 \cap P_2| = 1$.*

Equivalently, we may consider the plane graph to be a complete but weighted graph, where the edge weights indicate the number of items shared between two 2-maximal sets. This perspective will be more helpful in the next section, where we consider higher-dimensional geometric dependencies.

In certain simple circumstances, it is possible to infer or at least bound the size of a minimum generator based on properties of the plane graph. We present these results here.

THEOREM 3.7. *Let $\mathcal{P}' \subseteq \mathcal{P}$ cover V. That is, each $v \in V$ is contained in at least one $P \in \mathcal{P}'$. Then there exists a generator for V containing no more than $2|\mathcal{P}'|$ elements.*

Proof. Let S contain two elements from each $P \in \mathcal{P}'$. Then clearly $\bar{S} = V$, since $\mathcal{P}'$ covers V. □

THEOREM 3.8. *Let $\mathcal{P}' \subseteq \mathcal{P}$ cover V, and let the associated subgraph of the plane* *graph be connected. Then there exists a generator for V containing no more than $1 + |\mathcal{P}'|$ elements.*

Proof. Choose two elements $u, v \in P \in \mathcal{P}'$. Let $u, v \in S$. The closure of $\{a, b\}$ will contain all of P and one member of each neighbor of P. Select one additional member of each neighbor to add to S. Continue breadth first through the whole (connected) graph. □

These bounds are pessimistic; frequently much smaller generators can be found. It seems that the exact size depends not only on the connectivity of the plane graph, but also on how vectors are shared between planes. We leave this question aside and turn to algorithms for finding a generator.

3.3. Finding a minimal generator. Algorithm 2 is a simple greedy algorithm for finding a minimal generator. This works by assigning priorities to each 2-maximal set in $\mathcal{P}$. These priorities indicate how many items should be selected from that set to "close" it. Initially, these priorities will all be 2. When the priority of some P is two, then no vectors in it have been found by the process yet. If the priority is 1, then one vector has been found; selecting another allows the remainder of P to be added to the closure. At each step of our algorithm, we select a 2-maximal set P with minimal priority. Then, we add elements from P to build the generator set G. After updating the generator and what has been found, the priorities of neighboring planes are updated. Note that a simple modification of this algorithm builds a dependence graph. When items of a plane are selected as generators, they are added as roots of the dependence graph. The rest of the items of a plane are inserted with appropriate arrows.

Algorithm 2. Finding a minimal generator.

```
Given $V, \mathcal{P}$, and the plane graph
Let $G = \{\}$ be the (initially empty) generator.
Let $Q$ be an empty priority queue.
Let $D = \{\}$ be the set of points generated by $G$.
for $P \in \mathcal{P}$ do
  $Q[P] \leftarrow 2$
end for
while $Q$ not empty do
  Let $P$ be an item from $Q$ with minimal priority $m$.
  Remove $P$ from $Q$.
  Select $\{v_{i_j}\}_{j=1}^{m}$ new generator elements from $P - D$.
  $G \leftarrow G \cup \{v_{i_j}\}_{j=1}^{m}$
  $D \leftarrow D \cup P$
  for $P'$ adjacent to $P$ in the plane graph do
    $Q[P'] \leftarrow \max(0, 2 - |P' \cap D|)$
  end for
end while
```

Algorithm 2 bears some similarity to some well-known algorithms. Like Dijkstra's [5] and Prim's [18] algorithms, it relies on a priority queue to make a locally optimal choice. However, it is not the vertices of the graph (the 2-maximal sets) but the members of those vertices that are of primary interest. Still, the similarity suggests that an optimality proof of this algorithm may be possible. On the other hand, we recall our remarks above regarding the similarity to maximum independent sets.

3.4. Exploiting higher-dimensional dependencies. The discussion of the earlier subsection readily generalizes to higher-dimensional dependencies. Interestingly, both Algorithms 1 and 2 are simple to extend. We explain both of these here.

To handle general k-dependencies, we assume as input a $(k-1)$-independent set. (The generator for a $(k-2)$-independent set is $(k-1)$-independent.) In practice, this means an iterative process by which we find a minimal generator for one order of linear dependence, remove the generated elements, and then proceed to the next higher order. Rather than pairs of vectors, we consider sets of k vectors, each of which spans a k-dimensional Euclidean space. We let $\Pi : \mathbb{R}^d \to \mathbb{R}^{k+1}$. Then we compute the generalized cross product that maps k elements of $\mathbb{R}^{k+1}$ to the unique (up to scaling) vector normal to the inputs. Equivalently, the full singular value decomposition could be used to find a normal vector. (It produces an orthonormal basis both for the column space of a matrix and its orthogonal complement.) Then these $O(n^k)$ normals are searched for colinearity via a hash table, and matches are verified and recurred upon as necessary.

The analogue of Algorithm 2 is likewise very similar. Instead of a plane graph, we make use of a "hyperplane graph" with edges weighted based on the length of intersection (between 0 and $k-1$, inclusive). The initial priorities are set to k rather than 2. The rest of the algorithm is exactly the same.

4. Empirical results. We present a study of using our geometric techniques for the Laplacian on tetrahedra, showing how many dependencies of each order we find. In Table 4.1, we report the number of vectors that are found to be equal or colinear to some other vector, and the number of vectors in the generator for each order

TABLE 4.1
Results of optimizing the evaluation of the Laplacian on tetrahedra using Lagrange elements of degrees two through four. For each degree k, the top portion of the table indicates the number of vectors (n), the length of each vector (d), the number of vectors that are zero (num zero), and the number of vectors removed from the search space because they were equal or colinear to another vector. The remaining three portions of the table describe the results of searching successively for linear dependencies of degrees two, three, and four. Each portion states the number of vectors with which the search process began, the number of vectors that have that order of dependency, the size of the generator (including independent elements), and the number of MAPs in the optimized algorithm. The final row indicates the number of MAPs in the algorithm generated using complexity-reducing relations, as described in [13].

	2	3	4
n	55	210	630
d	6	6	6
Num zero	0	0	0
Num equal	6	42	150
Num colinear	0	44	73
Size for 2-search	49	146	432
Num with 2-dependency	49	145	372
Generator size	6	8	67
MAPs	105	327	1112
Size for 3-search	6	8	67
Num with 3-dependency	0	0	41
Generator size	6	8	47
MAPs	105	327	1072
Size for 4-search	6	8	47
Num with 4-dependency	0	5	44
Generator size	6	7	18
MAPs	105	330	1045
CRR MAPs	101	370	1118

Table 4.2
Results of optimizing the first contraction stage for the weighted Laplacian on triangles. See Table 4.1 for more details about the row and column labels. Blanks indicate computations that could not be completed.

	2	3	4
n	63	165	360
d	6	10	15
Num zero	14	22	32
Num equal	15	23	33
Num colinear	2	2	2
Size for 2-search	33	119	294
Num with 2-dependency	32	67	112
Generator size	11	92	252
MAPs	110	910	3808
Size for 3-search	11	92	252
Num with 3-dependency	7	39	86
Generator size	9	78	217
MAPs	104	818	3394
Size for 4-search	9	79	
Num with 4-dependency	9	53	
Generator size	6	59	
MAPs	98	717	
CRR MAPs	115	683	3726

of dependency. We also report the number of MAPs in the optimized algorithm. Comparing to the results in [13], we see that geometry can be at least as effective at reducing arithmetic as complexity-reducing relations. On triangles, however, the vectors are already so short that reducing the cost by geometric relations is much less practical.

We remark that, going from third- to fourth-order dependencies, the number of MAPs increases. This is due to the fact that a vector with one nonzero entry was written as a linear combination of four other vectors; greater care in a production-level implementation would prevent this.

We also consider the weighted Laplacian on triangles as in [13],

$$a_w(v,u) = \int_\Omega w(x)\nabla v(x)\cdot\nabla u(x)\,\mathrm{d}x, \tag{4.1}$$

with reference and geometric tensors given by

$$A^0_{i\alpha} = \int_E \Phi_{\alpha_1}(X)\frac{\partial\Phi_{i_1}(X)}{\partial X_{\alpha_2}}\frac{\partial\Phi_{i_2}(X)}{\partial X_{\alpha_3}}\,\mathrm{d}X \tag{4.2}$$

and

$$G^\alpha_e = w_{\alpha_1}\det F'_e\frac{\partial X_{\alpha_2}}{\partial x_\beta}\frac{\partial X_{\alpha_3}}{\partial x_\beta}, \tag{4.3}$$

respectively. In [13], we remarked that G_e is the outer product of the coefficient with the geometric tensor for the standard Laplacian ($(G^L)_e$). With this in mind, we could perform all of the contractions (of larger tensors) as in the constant coefficient case and search for geometric relations. However, in this case, *no* geometric relations between vectors were discovered. This suggests that geometric optimization is likely not as good a default optimization strategy as complexity-reducing relations. On the other hand, we may also perform the computations in stages as in [13]. Here, we consider the case

$$A_i^e = \left(A^0_{i,(\alpha_1,\alpha_2,\alpha_3)} w_{\alpha_1} \right) \left(G^L \right)_e^{(\alpha_2,\alpha_3)}, \tag{4.4}$$

in which case we consider contractions with the coefficient first. This may be optimized, but the subsequent stage of contracting with G^L is performed at full cost. Note that we obtain some benefit, finding vectors that are zero, equal, or colinear. We also compare to Table 4.8 in [13] to see that geometric optimization with second order dependencies leads to a slightly better result than complexity-reducing relations for quadratics and almost as good for cubics. We also see that substantial additional improvements are made by searching for third- and fourth-order dependencies for cubics. Third-order dependencies are also significant for quartics, but even with our efficient search algorithms, fourth-order search was beyond the capacities of our prototype Python implementation.

5. Combined optimization.

In [13], we reduced the construction of an optimized algorithm based on binary relations to a well-known graph construction. In the geometric context, the problem is equivalent to finding a generator. It is natural to look for an effective way of using both binary and geometric relations to give a greater overall reduction of arithmetic than using either separately. Rather than describing an algorithm for finding the exact optimum, we describe first a combinatorial structure, which, if it were found, would be the optimum. Then we describe a heuristic modification of Prim's algorithm that outperforms binary relations alone. Finally, we pose the problem in terms of an optimization problem over all permutations of V.

Essentially, we want to construct a rooted, weighted tree or forest with all the vectors as nodes. Each node either will be a leaf node, if the corresponding dot product is performed explicitly, or will have one or more neighbors, depending on whether the corresponding computation is performed via a complexity-reducing relation or a linear combination. Each node is also assigned a weight based on the cost of using either explicit computation, a complexity-reducing relation, or a linear dependence. Of all possible spanning forests, we desire to find one with minimal weight.

While such a structure probably cannot be found efficiently, the idea appropriately generalizes both the dependence graph and the minimum spanning tree. If we neglect all the linear relations and focus only on a complexity-reducing relation, a (rooted) minimum spanning tree exactly solves the problem. On the other hand, if we were to neglect the complexity-reducing relation, we would wind up with a dependence graph (augmented slightly if we had found equality or colinearity among vectors). We have implemented a straightforward generalization of Prim's algorithm in which each time a vector v is added to the tree, if it and another member of the tree are coplanar with a vector w not already added in the tree, a labeled edge is drawn from v and w. While this approach finds (provably and in practice) more efficient algorithms than using complexity-reducing relations alone, the current naive implementation and search through coplanar relations at each step makes the code impractical for anything beyond very simple problems at the present time.

Finally, we may pose the optimal computation as a combinatorial optimization problem over all permutations of V. In effect, the exact solution to this problem allows any previous computation to be employed at each step. To each ordering $\{v_i\}_{i=1}^{|V|}$ of V we can assign a weight. This weight is a sum over weights for each v_i. The weight of v_1 is always $w(v_1) = m$. For $i > 1$, we define

$$w_R(v_i) = \begin{cases} 2, & \exists j, k < i : R(\{v_i, v_j, v_k\}), \\ m, & \text{otherwise}, \end{cases} \tag{5.1}$$

to indicate whether v_i is linearly dependent on two of its predecessors, and also define

$$w_\rho(v_i) = \min_{j<i} \rho(v_i, v_j) \tag{5.2}$$

to indicate how expensive it is to compute $v_i^t g$ using the complexity-reducing relation ρ. Finally, define

$$w(v_i) = \min(w_{R,i}, w_{\rho,i}). \tag{5.3}$$

Let $\mathcal{V}$ be the set of all permutations of V. Our goal is to solve the minimization problem

$$\min_{\mathcal{V}} \sum_{i=1}^{|V|} w(v_i). \tag{5.4}$$

6. Related mathematics. Our present context seems to be related to many other mathematical structures, especially when we consider the 2-maximal sets on a 1-independent set of Euclidean vectors. We describe these relations here, restricting ourselves to this case.

6.1. Finite linear spaces. Most strikingly, the set of vectors V and the set of all planes spanned by them are an instance of a *finite linear space* [1]. Finite linear spaces are a logical precursor to finite geometries; they specify an incidence relation between *points* and *lines* such that any two points lie on a unique line, and each line contains at least two points. In our situation, the vectors in V are the points, and the 2-maximal sets are lines. (Formally, we must also include with the set 2-maximal sets all pairs of vectors that do not already lie in some 2-maximal set.) The research literature on finite linear spaces seems focused on fundamental questions of structure and relationships, such as whether spaces with particular properties exist, or when they may be embedded into finite affine or projective planes; algorithms over finite linear spaces seem less studied. We have been unable to locate any notion of geometric closures or generators such as we have described above; the idea may be new.

On the other hand, we have found reference to the plane graph used in section 3. Finite linear spaces, and the weaker notion of *partial linear spaces* (in which any two points may or may not be colinear) are naturally interpreted as hypergraphs. In [14], Klein and Margraf restate the Erdős–Faber–Lovász conjecture (see [7]) that the chromatic index of a linear hypergraph on v points is at most v, in terms of properties of the *intersection graph* of the hypergraph. This graph is defined exactly in the same way as our plane graph.

6.2. Ternary relations. We may consider coplanarity in Euclidean space as an unordered ternary relation, R. Given a set of three vectors, $\{x, y, z\}$, R gives truth if they are linearly dependent and falsity otherwise. Other work has made use of *ordered* ternary relations. The theory of ordered ternary relations has been used to provide a notion of *betweenness* for some time [15]. Recent applications have also appeared in [17], which is based on a technical report [16] that provides a historical survey of the use of ordered ternary relations as a type of geometry, as does [15]. In [15], ternary relations associated with a metric d are studied. More precisely, one can define $T(x, y, z)$ to hold if and only if $d(x, y) + d(y, z) = d(x, z)$. If we take $d(x, y)$ to be the acute angle between the lines x and y, then $T(x, y, z)$ is the coplanarity relation we consider here. However, the fact that y can be viewed as *between* x and z plays no significant role for us, so we have not further explored the theory of ordered ternary relations.

6.3. Topology. Our notion of geometric closure is, at least on the surface, different from topological closure. The union of two 2-maximal sets need not be geometrically closed (indeed, this is even beneficial, as it makes the generator much smaller and hence the algorithm we find more efficient). Therefore, the geometrically closed sets (and hence geometric closure) cannot be used to define a topology through closed sets. It may be possible to define some kind of filter or neighborhood base, but we have not pursued this further.

6.4. Matroids. As we deal heavily with linear independence (and more particularly linear dependence) among a set of vectors, it seems that there should be some connection to matroids [21]. Matroids also provide a notion of *closure*. For vector matroids, the closure involves anything that is linearly dependent on something in the set. On the other hand, our notion of closure requires a low-dimensional linear dependence; only linear dependence on two vectors in the set is allowed (recursively) rather than linear dependence on any number of items. It may be possible in the future to interpret our context as a nonstandard matroid. Doing so would likely open up new algorithmic perspectives on the issue.

7. Conclusions and future work.

Searching for linear dependence among the entries of the reference tensor can be a powerful technique for optimizing the evaluation of finite element stiffness matrices. The potential for improving our current implementation, integrating them with tools such as FFC [12, 11], and combining geometric and binary optimization is great. On the other hand, we have left several open issues, both practical and theoretical.

On the practical side, a more efficient implementation of the geometric search algorithms in C++ would allow us to study more detailed forms. Providing an interface to FFC would allow these optimizations to be used in practice, much as complexity-reducing relations are now available. While these issues are fairly straightforward, finding ways of tackling (or reformulating) the combinatorial optimization problem discussed above will be much more challenging.

On the theoretical side, we lack understanding about the structure of the minimum generator problem. If our algorithm does not always find a minimum, is finding the minimum truly difficult? How well does our greedy algorithm approximate the minimum? Perhaps more light will be shed on this problem by better understanding exactly what mathematical structures we are dealing with. As we saw in section 6, there is tantalizing similarity to finite linear spaces and matroids, among other structures. Making a more precise connection to an appropriate mathematical structure would clarify optimal generators greatly.

REFERENCES

[1] L. M. Batten and A. Beutelspacher, *The Theory of Finite Linear Spaces: Combinatorics of Points and Lines*, Cambridge University Press, Cambridge, 1993.

[2] G. Baumgartner, A. Auer, D. E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. J. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C.-C. Lam, Q. Lu, M. Nooijen, R. M. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, *Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models*, Proc. IEEE, 93 (2005), pp. 267–292.

[3] P. Bientinesi, E. S. Quintana-Ortí, and R. A. van de Geijn, *Representing linear algebra algorithms in code: The FLAME application programming interfaces*, ACM Trans. Math. Software, 31 (2005), pp. 27–59.

[4] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[5] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[6] S. Egner and M. Püschel, *Automatic generation of fast discrete signal transforms*, IEEE Trans. Signal Process., 49 (2001), pp. 1992–2002.

[7] P. Erdős, *On the combinatorial problems which I would most like to see solved*, Combinatorica, 1 (1981), pp. 25–42.

[8] J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn, *FLAME: Formal linear algebra methods environment*, ACM Trans. Math. Software, 27 (2001), pp. 422–455.

[9] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[10] R. C. Kirby, M. Knepley, A. Logg, and L. R. Scott, *Optimizing the evaluation of finite element matrices*, SIAM J. Sci. Comput., 27 (2005), pp. 741–758.

[11] R. C. Kirby and A. Logg, *A compiler for variational forms*, ACM Trans. Math. Software, 32 (2006), pp. 417–444.

[12] R. C. Kirby and A. Logg, *Efficient compilation of a class of variational forms*, ACM Trans. Math. Software, 33 (2007), to appear.

[13] R. C. Kirby, A. Logg, L. R. Scott, and A. R. Terrel, *Topological optimization of the evaluation of finite element matrices*, SIAM J. Sci. Comput., 28 (2006), pp. 224–240.

[14] H. Klein and M. Margraf, *On the linear intersection number of graphs*, preprint, 2003, http://arXiv.org/PS_cache/math/pdf/0305/0305073.pdf.

[15] R. Mendris and P. Zlatoš, *Axiomatization and undecidability results for metrizable betweenness relations*, Proc. Amer. Math. Soc., 123 (1995), pp. 873–882.

[16] K. Nehring, *A Theory of Qualitative Similarity*, Technical Report 10, U.C. Davis, Davis, CA, 1997; available online at http://www.econ.ucdavis.edu/working_papers/97-10.pdf.

[17] K. Nehring and C. Puppe, *Diversity and dissimilarity in lines and hierarchies*, Math. Social Sci., 45 (2003), pp. 167–183.

[18] R. Prim, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.

[19] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, *SPIRAL: Code generation for DSP transforms*, Proc. IEEE, 93 (2005), pp. 232–275 (special issue on "Program Generation, Optimization, and Adaptation").

[20] R. C. Whaley and J. Dongarra, *Automatically tuned linear algebra software*, in Proceedings of SuperComputing 1998: High Performance Networking and Computing, 1998 (CD-ROM); available online at http://www.cs.utsa.edu/~whaley/papers/atlas_sc98.ps.

[21] H. Whitney, *On the abstract properties of linear dependence*, Amer. J. Math., 57 (1935), pp. 509–533.