Heuristics for the Automatic Construction of Coarse Grids in Multigrid Solvers for Finite Element Matrices

Mark Adams *

January 29, 1998

Abstract

The finite element method (FEM) has proven to be a popular spatial discretization technique in the simulation of complex physical systems in many areas of science and engineering. Implicit solution schemes, used in the time discretization, require the application of the inverse of an operator which is usually linear or linearized; thus, after the FEM discretization, a linear set of discrete equations must be solved. The solution of these equations is a dominate cost of FEM. Direct solvers are easy to use; that is they are relatively invariant to condition number and effective (fast) for small scale problems. But for the solution of large complex systems, iterative methods are vary attractive due to their potential $O(n)$ time and space complexity. *Multigrid* (MG) is a powerful solver, or preconditioner, and is ideally suited for FEM matrices. MG performance, however, is significantly influenced by the quality of the coarse grids; *maximal independent sets* (MISs) are a useful and popular tool in the automatic construction of these coarse grids on unstructured FE meshes. The inherent flexibility common to all MIS algorithms, allow for the use of heuristics to improve their quality. We will present heuristics and methods to optimize the quality of MISs, and the meshes constructed from them, for use in multigrid solvers for 3D continuum mechanics.

Key words: maximal independent sets, multigrid, parallel solvers

1 Introduction

This paper presents a set of algorithms and heuristics for the construction of *maximal independent sets* (MISs) of finite element (FE) meshes in 3D solid mechanics for use in multigrid equation solvers. This work is motivated by the success of the finite element method (FEM) science and engineering, coupled with the wide spread availability of ever more powerful computers, which has lead to the need for efficient equation solvers for large sparse systems. The matrices from FE simulations tend to be very poorly conditioned; this fact has made the use of direct solvers popular in both research and industry as they are relatively unaffected by the condition number of the matrix. Direct methods however possess suboptimal time and space complexity when compared to iterative methods. As larger and faster computers are becoming more widely available, to a larger number of research and industrial institutions, the use of iterative methods is becoming increasingly more necessary. Thus given the computational resources that are available today, and that are continually becoming more available, the asymptotics of direct methods are overwhelming the relatively small constants in their complexity, resulting in the need to resort to iterative methods.

Iterative methods are notoriously unreliable on most FE problems of interest. Many iterative methods perform poorly on the matrices produced by accurate FE simulations of complex geometric domains - these difficulties tend to be amplified by many of the sophisticated, state of the art, FE formulations that are used in research today and will certainly be used more by industry in the future. Multigrid is one of a family of "optimal" multilevel domain decomposition methods [13]. Multigrid is known to be a highly effective methods to solve FE matrices, although its application to the unstructured meshes, that are the hallmark of

*Department of Civil Engineering, University of California Berkeley, Berkeley CA 94720 (madams@cs.berkeley.edu). This work is supported by DOE grant No. W-7405-ENG-48

FEM, has not been well developed. This paper will discuss methods for the effective application of multigrid in the solution of the large sparse system of equations that arise from FE simulations.

Multigrid is known to be *the* optimal serial solution method for the finite difference Poisson equation [4]. But unlike the optimal parallel method to solve Poisson's equation (FFT in the P-RAM complexity model), multigrid can be applied effectively to general second order PDE finite element problems in elasticity [11, 3] and plasticity [6, 9], as well as fourth order problems [5, 16]. The application of MG to unstructured FE meshes is not transparent however. MG requires two types of operators in its application: first, *restriction* and *interpolation* operators, which can be implemented with a rectangular matrix (R and R^T respectively); and second the PDE operator, or sparse matrix on the coarse mesh (the fine grid matrix is the one to be solved).

The coarse grid matrix can be formed in one of two ways - either algebraically ($A_{coarse} \leftarrow RA_{fine}R^T$) or by creating a new FE problem on the coarse grid thereby letting the FE implementation construct the matrix. There are advantages and disadvantages to each approach, the algebraic method has the advantage that it places less demand on the user by not requiring that a good coarse mesh be provided. The construction of good quality meshes is a very challenging and expensive part of using FE and requiring good coarse meshes may be an onerous responsibility for the solver to place on the user. Additionally mesh generators, be they automatic or semi-automatic, are not accustomed to approximating the domain automatically (i.e. not strictly maintaining the topology of the domain) which is often required - especially on the coarsest grids. The explicit construction of a new FE problem on the coarse grid may however provide better quality coarse grid operators as well as better restriction and interpolation operators, at least in some cases, but we are not aware of any direct comparison of the two methods. We have opted for the algebraic approach - this requires that we construct only the restriction matrices, and all of the operators that MG requires can be constructed from these.

Our work thus centers on constructing good quality restriction operators. To do this we utilize an algorithm [7, 3] applied recursively to produce a coarse grid, and its attendant operators, from a "fine" grid. A high level view of the algorithm is as follows:

- The vertex set at the current level (the "fine" mesh) is automatically and *evenly* coarsened, with a MIS algorithm, to produce a much smaller subset of vertices.
- This vertex set is then automatically remeshed with tetrahedra.
- These tetrahedra are used with standard finite element shape functions to produce the restriction matrix.
- This restriction matrix is then used to construct the coarse grid matrix from the fine grid matrix - $A_{coarse} \leftarrow RA_{fine}R^T$.
- Finally these matrices are used in one iteration of full multigrid as a preconditioner for a Krylov subspace method.

This paper will discuss methods and heuristics useful in optimizing the quality of these restriction operators. These methods will use coordinate data available in all FE simulations, we will also employ element data that is available when continuum elements are used. We will show how to use this data to categorize topological elements of the FE mesh (i.e. corners, ridges, surfaces, and interiors), and to use this information in a logical way to modify the graph that is used in a MIS algorithm. We will also show how these heuristics can be applied globally on parallel platforms, as well as a simple method to get the coarse grids to more effectively "cover" the fine grids.

This paper will proceed as follows: in §2 we describe our methods, in §3 we present numerical results on some representative problems in solid mechanics, and we conclude in §4.

2 Automatic Coarse Grid Creation with Unstructured Meshes

This section will introduce the components that we will be using for the automatic construction of coarse grids on unstructured meshes, but first we need to state what we want our coarse grids to be able to do.

The goal of the coarse grids in MG is to approximate the low frequency error in the current grid. Each successive grid's FE function space should (with a drastically reduced vertex set) approximate, as best it can, the highest frequency (or eigen functions) of the current grid - that it *can* approximate well. That is, with say 1/10th the vertices it is natural that one could only represent the lowest 10% of the fine grid spectra well - the coarse grid functions should approximate the the *highest* part of this *lower* part of the spectrum as well as possible. It is not possible to satisfy this criterion directly (on unstructured grids), but a natural heuristic is to represent the geometry as well as possible, in some sense, with a much smaller set of vertices. One promising approach is to use computational geometry techniques to characterize features and maintain them on the coarser grids [14]. An alternative and popular method is to use a *maximal independent set* as a heuristics to evenly coarsen the vertex set - if vertices are added to the MIS randomly then the MIS is expected to be good representation of the fine grid in the sense of *evenly* coarsening the grid points and maintain the feature characteristics of the mesh [14]. A MIS is not unique in general, and an arbitrary MIS is not likely to perform well as will show.

We motivate our approach by first looking at the *structured* MG algorithm. We can characterize the behavior of multigrid on structured meshes, as shown in figure 1, as: select every other vertex (starting from the boundary), in each dimension, for use in the coarse grid.

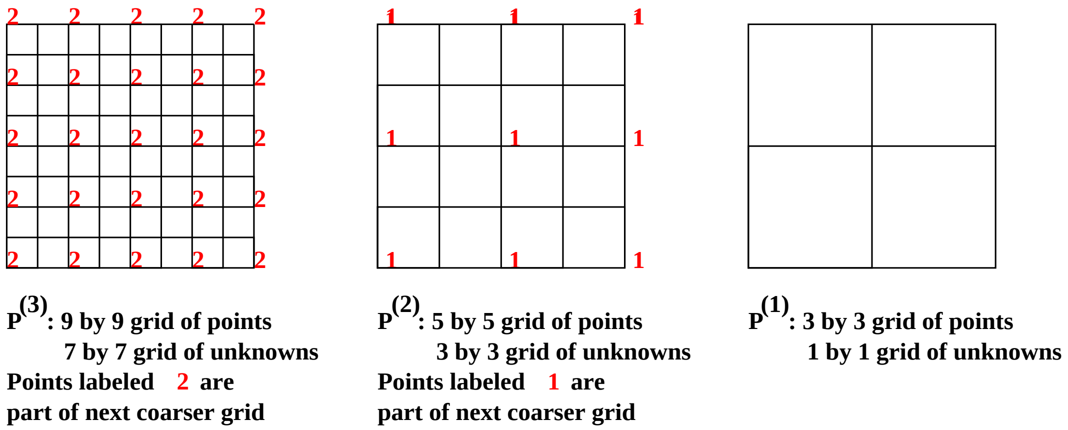

Figure 1: Multigrid coarse vertex set selection on structured meshes

To apply MG to unstructured meshes it is natural to try to imitate the "look" of the structured algorithm in hopes of imitating its success. The coarse grids in figure 1, in addition to evenly coarsening the vertex set, also pay special attention to the boundaries. A more accurate description of MG meshes on regular grids is: place each vertex v in the *topological category* of the lowest dimension to which v belongs; select about $1/2^d$ vertices (e.g. a MIS) from each category. Define *topological categories*, each with an inherent dimension d, and place each vertex v in the category of lowest dimension, to which v belongs. For instance a natural set of topological categories is: corners ($d = 0$), ridges ($d = 1$), surfaces ($d = 2$), and interiors ($d = 3$). Now we can see that each category of vertices, in the regular mesh case, is reduced in number from one grid to the next by about a factor of $1/2^d$, and is a MIS (assuming a nine point stencil is in use). Now this is all well and good, but how can these observations be made incarnate in an algorithm - to do this we first need to look at standard methods used in the construction of a MIS.

2.1 Maximal Independent Set Algorithms

An *independent set* is a set of vertices $I \subseteq V$ in a graph $G = (V, E)$, in which no two members of I are adjacent (i.e. $\forall v, w \in I, (v, w) \notin E$); a *maximal independent set* is an independent set for which no proper superset is also an independent set. A MIS of a graph is not unique, in general, and its calculation is a relatively easy task to perform. The simple serial greedy algorithm in figure 2 will construct a MIS. Here we endow vertices v with a mutable data member *state*, $state \in \{selected, deleted, undone\}$. All vertices begin in the *undone* state, and ending in either the *selected* or *deleted* state; the MIS is defined as the set of *selected* vertices. Each vertex v will also be given a list of adjacencies *adjac*, defined by $v.adjac = \{v1 \mid (v, v1) \in E\}$.

forall $v \in V$
  if $v.state = undone$ **then**
    $v.state \leftarrow selected$
    forall $v1 \in v.adjac$
      $v1.state \leftarrow deleted$
$I \leftarrow \{v \in V \mid v.state = selected\}$

Figure 2: Basic MIS Algorithm for the serial construction of a MIS

There is a great deal of flexibility in the order in which vertices are chosen in each iteration of the algorithm. Herein lies a simple opportunity to apply a heuristic, as the first vertex chosen will always be *selectable* and the probability is high that vertices which are chosen early will also be selectable. Thus if an application can identify vertices that are "important" then those vertices can be ordered first and so that a less important vertex can not delete a more important vertex. We can now decide that corners are more important than ridges and ridges are more important than surfaces and so on, and order the vertices with all corners first, then ridges, etc. With this heuristic in place basic MIS algorithm in figure 2 will guarantee, for instance, that the number of ridge vertices on the coarse grid (in each ridge segment) will satisfy $\left|V^{coarse}_{ridge}\right| \geq \frac{\left|V^{fine}_{ridge}\right|-2}{3}$; whereas a valid MIS could remove *all* ridge and corner vertices from the graph, which would clearly be disastrous.

2.2 Parallel Maximal Independent Set Algorithms

For parallel processing we partition the vertices onto processors and define the vertex set V_p owned by processor p of P processors. Thus $V = V_1 \cup V_2 \cup ... \cup V_P$ is a disjoint union, and for notational convenience we give each vertex an immutable data member *proc* after the partitioning is calculated to indicate which processor is responsible for it. We can now use a random algorithm [8] where each vertex is given a random number and vertices can only be selected which are connected to strictly *lower* numbered vertices. Alternatively we can use a partition based algorithm [1] where a vertex v can only be selected if $\{v1 \in v.proc, v1.state \neq deleted \mid v1.proc > v.proc\} = \emptyset$.

The order in which each processor traverses the local vertex can be governed by our heuristics although the global application of a heuristic requires an alteration in the algorithm. In the random algorithm the number space can be divided into continuous segments, each category can simply generate random numbers in its own segments of the number line - with the highest segments given to the corners, the next highest to the ridges, and so on. The partition based algorithm requires that vertices v are give an immutable data member *v.topo*; in the MIS algorithm, processor p can select a vertex v only if:

$$\{v1 \mid v1 \in v.adjac, v1.state \neq deleted \mid v1.topo > v.topo \;\; or \;\; (v1.topo = v.topo \;\; and \;\; v1.proc > v.proc)\} = \emptyset$$

Note that with these modifications we no longer need to iterate over the vertex list in figure 2 in any particular order as these additions to the algorithms will insure similar behavior.

2.3 Topological Classification of Vertices in Finite Element Meshes

Our methods are motivated by the well known fact [7, 3] that coarse grids of multigrid methods must represent the boundary of the domain well in order to approximate the function space of the fine mesh well, which is necessary for multigrid methods to be effective. Intuitively this can be done by emphasizing the vertices that "define" the *domain*. Note, we will define domain in a slightly non-standard way as to mean a contiguous region of the real FE domain with a particular material property, thus for our discussion the boundary of the PDE proper will be augmented with boundaries between different material types. If the domain is convex then a convex hull is useful in reasoning about how to best classify the vertices. Vertices that are on the convex hull should be given more emphasis the those that are not (i.e. interior vertices). Vertices that are required to define the convex hull are likewise more important than vertices that simply lay on the convex hull. Domains of interest are by no means necessarily convex, but the idea of emphasizing vertices by their contribution to defining the boundary of a domain is useful in the exposition of our methods.

The first type of classification of vertices is to find the *exterior* vertices - if continuum elements are used then this classification is trivial. For non-continuum elements like plates, shells and beams, heuristics such as minimum degree could be used to find an approximation to the "exterior" vertices, or a combination if mesh partitioners and convex hull algorithms could be used. For the rest of this paper we will assume that continuum elements are used and so a "hull" of the domains, represented by a list of *facets*, can be defined. These exterior vertices will give us our first vertex classification: *interior* vertices are vertices that are not exterior vertices. Exterior vertices require further classification, but first we need a method of automatically identifying *faces* in our FE problems.

2.4 A Simple Face Identification Algorithm

To describe our algorithm we will assume that a list of facets $facet\_list$ has been created of the boundaries of the FE mesh. Assume that each facet $f \in facet\_list$ has calculate its unit normal vector $f.norm$. Assume that each facet f has a list of facets $f.adjac$ that are adjacent to it. With these data structures, and a list with with $AddTail$ and $RemoveHead$ functions with the obvious meaning, we can calculate a $face\_ID$ for each facet with the algorithm shown in figure 3.

forall $(f \in facet\_list)$ $f.face\_ID \leftarrow 0$
$Current\_ID \leftarrow 0$
forall $f \in facet\_list$
    if $f.face\_ID = 0$
        $list \leftarrow \{f\}$
        $norm \leftarrow f.norm$
        $Current\_ID \leftarrow Current\_ID + 1$
        while $list \neq \emptyset$
            $f \leftarrow list.RemoveHead$
            $f.face\_ID \leftarrow Current\_ID$
            forall $f1 \in f.adjac$ - - $-1 < TOL \leq 1$ is a user selected tolerance
                if $norm^T \cdot f1.norm > TOL$ and $f1.face\_ID = 0$
                    $list.AddTail(f1)$

Figure 3: Face identification algorithm

This algorithm simply repeats a breadth first search, of trees rooted at an arbitrary undone facet, which is pruned by the requirement that a minimum angle be maintained by all facets in the tree relative to the root. This heuristic is a simple way to identify *faces* (or manifolds that are somewhat "flat") of the boundaries in the mesh.

These faces are useful for two reasons:

- Topological categories for vertices, used in the heuristics of §2.4, can be inferred from these faces:
  - A node attached to only one face is a *surface*.
  - A node attached to two different faces is an *ridge*.
  - A node attached to more than two different faces is a *corner*.
- Vertices not associated with the same faces should not interact with each other in the MIS algorithm.

This second criterion will be discussed in the next section.

2.5 Modified Maximal Independent Set Algorithm

We now have all of the pieces that we need to describe the core of our method. First we classify vertices and ensure that a vertex of *lower* rank does not suppress a vertex of *higher* rank - this was done with a slight modifications to standard MIS algorithms in §2.1. Second we want to maintain the integrity of the "faces" in the original problem as best we can. The motivation for this second criterion can be seen in figure 4.

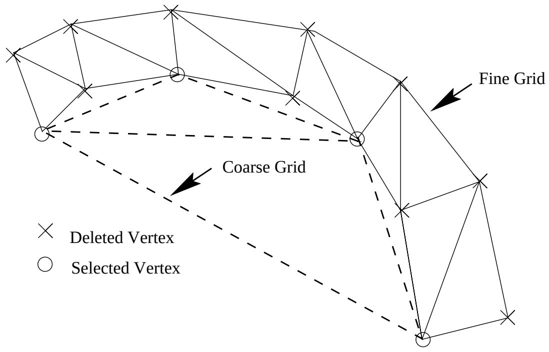

Figure 4: Poor MIS for multigrid of a "shell"

If the FE mesh has a thin region then the MIS as described in §2.1 can easily fail to maintain a cover of the vertices in the fine mesh. This comes from the ability of the vertices on one face to decimate the vertices on an opposing face, this phenomenon could be mitigated by randomizing the order that the vertices are added to the MIS, at least within a vertex type. But randomization is not good enough as these skinny regions tend to deteriorate the performance of iterative solvers, so we need to do something better.

A simple fix for this problem is to modify the graph to which we apply the MIS algorithm - we want to maintain the same vertices in the graph, but will reduce the edge set. To avoid the problems illustrated in figure 4, we can look at our method of classifying vertices again:

- A node attached to only one face is a *surface*.
- A node attached to two different faces is an *ridge*.
- A node attached to more than two different faces is a *corner*.

Now we claim that by *removing all edges between vertices that do not share a face*, we will force the MIS to be a more "logical" and economically representative of the fine mesh, as shown in figure 5. For instance we do not want a corners to delete a ridge vertex with which it does not share an exterior facet. Another example is that we do not want a ridge vertices to delete ridges with which it does not share a face (this is the most effective heuristic in this example). And finally we will augment our heuristic and not allow corners be deleted at all - this could be problematic on some meshes that have many initial "corners", as defined by our algorithm, and a reclassification of the remeshed vertices on coarse meshes may be advisable.

We are now free to run our MIS algorithm on this modified graph, figure 6 shows and example of a possible MIS and remeshing.

2.6 Vertex Ordering in MIS Algorithm on Modified FE Graphs

An additional degree of freedom, in this algorithm as described thus far, is the order of the vertices within each category. Thus far we have implicitly ordered the vertices by topological category; the ordering within each category can also be specified. Two simple heuristics can be used to order the vertices: random order, or a "natural" order. Meshes will generally be initially ordered in either a block regular order (i.e. an assemblage of logically regular blocks), but this depends on the mesh generation method used. Initial vertex orders can also be ordered in a cache optimizing order [15] like Cuthill-McKee. Both of these ordering types are what we will call *natural* orders, and we assume that the "initial" order of our mesh is of this type (if not then we can make it so). The MISs produced from natural orderings tend to be rather dense, random

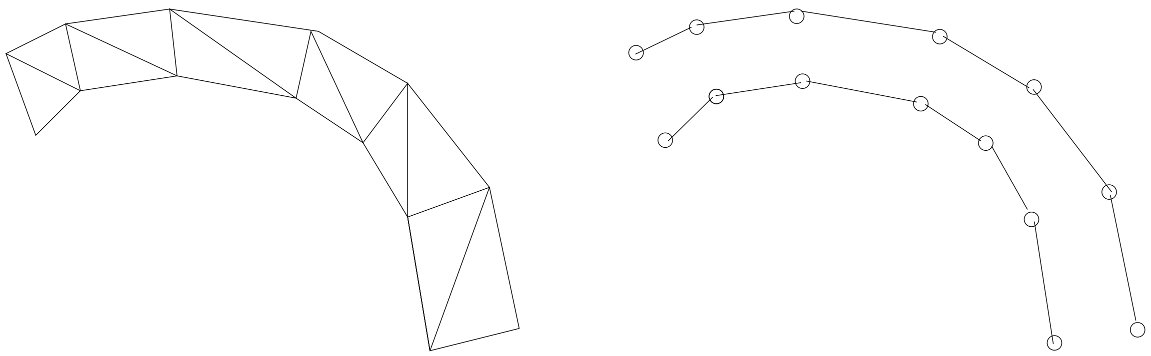

Figure 5: Original and Fully modified graph

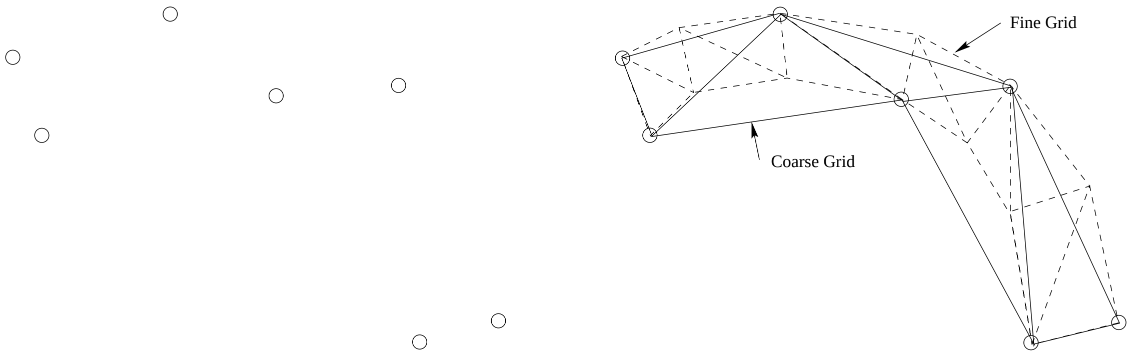

Figure 6: MIS and coarse mesh

ordering on the other hand will tend to be more sparse. That is the MISs with natural orderings will tend to be larger than those produced with random orders. Note that for a uniform 3D hexahedral mesh, the asymptotics of the size of the MIS is bounded from above by $1/2^3$, and from below by $1/3^3$; natural and random orderings are simple heuristics to approach these bounds.

Small MISs are preferable as this means that there is less work to be done on the coarser mesh, also fewer levels will be required before the coarsest grid is small enough to solve directly, but care has to taken to not degrade the convergence rate of the solver by compromising the quality if the coarse grid representation. In particular, as the boundaries are important to the coarse grid representation it may be advisable to use natural ordering for the exterior vertices and a random ordering for the interior vertices - we use this approach in our numerical experiments.

2.7 Meshing of Vertex Set on the Coarse Grid

The vertex set for the coarse grid remains to be meshed - this is necessary in order to apply FE shape functions to calculate the restriction operator. We use a standard Delaunay meshing algorithm to give use these meshes. This is done by putting the mesh inside of a bounding box, thus adding dummy vertices to the coarse grid set, and then meshing this to produce a mesh that covers all fine grid vertices. The tetrahedra attached to the bounding box vertices are removed and the fine grid vertices within these deleted tetrahedra are added to a list of "lost" vertices (*lost_list*). With this (usually) convex body we continue to remove tetrahedra from the mesh that connect points that were not *near* each other on the fine mesh (recall the vertex set are still nested), and that do not have any vertices that lie "uniquely" within the tetrahedron. Define a vertex v to lie uniquely in a tetrahedron if v lies completely within the tetrahedra and not on its surface, or there is no adjacent tetrahedra to which v can be added. More precisely if all of a vertices v shape function values are larger than some very small negative tolerance $-\epsilon$ (we use only linear shape functions), or there is not an adjacent tetrahedra that can "accept" v, then that tetrahedra is deemed necessary and not removed. We also use a more aggressive phase in which we use a more negative, though still small, tolerance, to try to remove more tetrahedra - but the "orphaned" vertices are added to the *lost_list*. The resolution of the vertices in the *lost_list* will be discussed in the next section

2.8 Coarse Grid Cover of Fine Grid

The final optimization that we would like to employ is to improve the cover of the coarse mesh on the fine vertex set. With these coarse grids constructed the interpolation operators are calculated by evaluating standard FE element shape functions, of the element to the which the fine grid vertex is *associated*, at the coordinate of the fine grid vertex. So each fine grid vertex must be associated with an element in the coarse grid - that is the element in which the fine grid vertex is physically located. In general however some fine grid vertices will fail to be "covered" by the coarse grid (the *lost_list* from the previous section). This problem can be solved in one of two ways: find a near by element and use it, or move the vertices on the coarse grid so as to cover all fine grid vertices. We can use the interpolation of an element that does not cover a fine mesh point, the interpolation values will simply not all be between zero and one. Intuition tells us however that interpolation will be of higher quality if the interpolation point is within an element - otherwise it is an extrapolation. Alternatively one can move the coarse grid vertex positions to cover the fine vertices in *lost_list*.

The optimal coarse grid vertex positions (or an approximation to it) could perhaps be constructed with the use of interpolation theory to provide *cost functions*, and linear or non-linear programming. We have instead opted for a simple, greedy algorithm that iteratively traverses the exterior vertices of the coarse mesh and applies a simple algorithm to *try* to cover the uncovered vertices that are near it. This algorithm first selects a coarse grid vertex c and finds the "uncovered" fine grid vertices in *lost_list* that are near c - call this list $lost\_list_c$. A unit vector ϕ is calculated: the weighted average of the normals of the facets connected to the coarse grid vertex ($c.facet\_list$) weighted by the facet area. The normal of each facet that does not have a positive inner product (with this ϕ vector) is added to ϕ until $\{f \in c.facet\_list \mid f.norm^T \cdot \phi < 0\} = \emptyset$. Now a standard geometric predicate [12], can be used to test if a fine grid vertex $v \in lost\_list_c$ is "outside" or "inside" of each facet $f = (a, b, c) \in c.facet\_list$, that is if the volume of the tetrahedra $t = (a, b, c, v)$ is positive then v is outside of f. A scale α is then calculated by finding that smallest (positive) α required, $\forall f = (a, b, c) \in c.facet\_list$ and $\forall v \in lost\_list_c$ such that:

$$\begin{vmatrix} a.x & a.y & a.z & 1 \\ b.x & b.y & b.z & 1 \\ c.x + \alpha \cdot \phi.x & c.y + \alpha \cdot \phi.y & c.z + \alpha \cdot \phi.z & 1 \\ v.x & v.y & v.z & 1 \end{vmatrix} \leq 0.0$$

A limit on the value of α is also imposed to avoid large motions that can most likely be avoided by letting other coarse grid vertices participate in the attempt to cover each fine grid vertex. This procedure is then applied to each exterior coarse grid vertex in one outer loop of the algorithm. We generally run about five outer iterations and then use the extrapolated shape function values for the fine grid vertices that remain uncovered.

Figure 7 shows an illustration of what our algorithm might do on our running example.

Figure 8 shows and example of our methods applied to a problem in 3D linear elasticity. The fine (input) mesh is shown with three coarse grids used in the solution.

3 Numerical Results

To demonstrate the effectiveness of the methods that we have discussed we will use three test problems in linear elasticity, shown in figure 9. The problems are chosen to exercise the primary problem features that we have tried to accommodate: material coefficient discontinuities, thin "shell" types of features, and curved surfaces. The first problem is of a hard sphere (Youngs modulus E = 1, Poisson ratio (v) = 0.35) encased in a soft rubber like material (E = 10^{-4}, v = 0.49). The second problems is a steel beam-column connection made of thin, poorly proportioned, elements. The third problem is of a tube fixed at one end and loaded at the other end like a cantilever - a thin slit of the rubber like material of the first problem runs

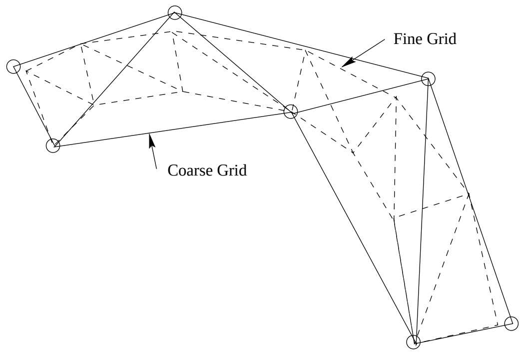

Figure 7: Coarse grid after vertices have been moved to cover all fine grid vertices

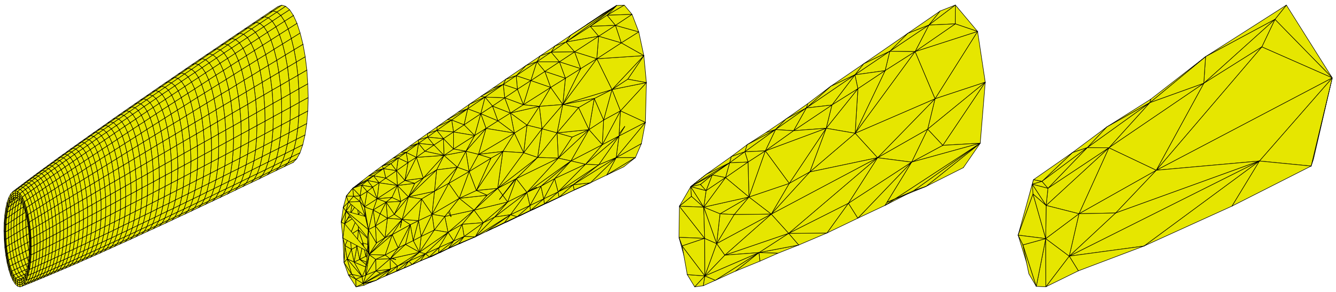

Figure 8: Fine (input) grid and coarse grids for problem in 3D elasticity

down the length of one side of the tube. The "sphere" problem has 40,000 equations, the "beam" has 36,000 equations, and the "tube" has 60,000 equations. All problems use eight node trilinear "brick" elements; the hard material is a standard displacement element and the soft material is a mixed formulation.

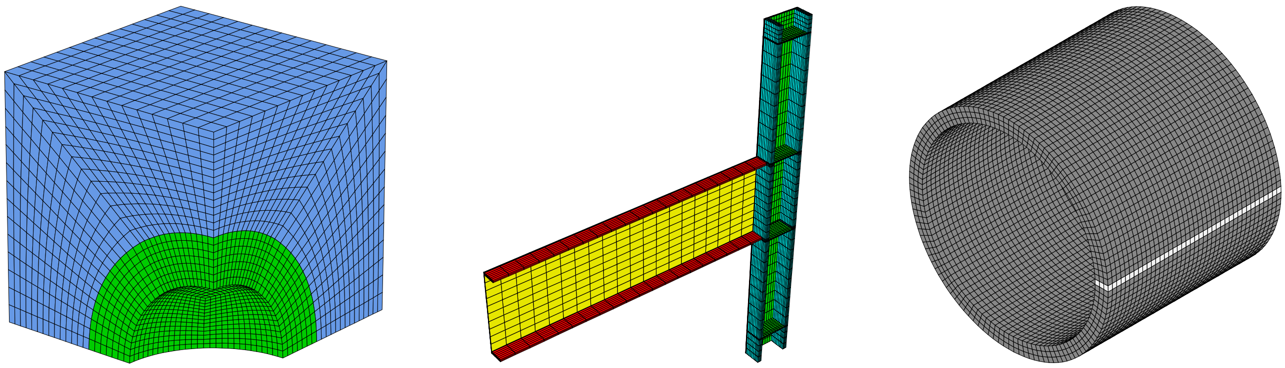

Figure 9: Test problems from linear elasticity: Sphere (40,000 dof), beam-column (36,000 dof), tube (60,000 dof)

Each problem is solved with conjugate gradient (CG) solver, to a relative tolerance of 10^{-6}. Full multigrid is used for the preconditioner, the smoother is CG preconditioned by block Jacobi, the number of blocks was reduced by a factor of eight at on each successive level. All problems used three coarse grids, so that the top grid (solved directly) was a few hundred equations in size. We present numerical experiments on an IBM SP (120 MHz PowerPC 604e CPU). Finite Element Analysis Program (FEAP)[17], is used to generate out test problems and produce our graphics. We use ParMetis [10] to calculate our partitions, and PETSC [2] for our parallel programming development environment. Our code is implemented in C++ and FEAP is implemented in FORTRAN, PETSc and ParMetis are implemented in C.

To demonstrate the effectiveness of our methods, we run these test problems with: Pure MIS (no optimizations); Pure MIS with the heuristics of exterior vertices ordered first, and interior ordered last and randomly; our modified MIS without the vertex cover heuristics; and finally all of our optimizations. The solution times and the iteration counts are shown in figure 10.

Problem Names	Sphere	Beam-column	Tube
Number of equations	40,000	36,000	60,000
Condition $K(A)$ of matrix	$7.2 \cdot 10^6$	$1.0 \cdot 10^8$	$1.8 \cdot 10^5$
Number of pre (and post) smoother applications	2	3	3
Number of blocks in Jacobi smoother (fine grid)	240	32	256
Pure MIS (no optimizations)	116(58)	201(50)	296(69)
Pure MIS w/ exterior vertices ordered first, interior last and random	75.1(36)	225(54)	286(70)
Modified MIS w/o vertex cover heuristics	56.5(27)	91.0(25)	104(14)
Modified MIS w vertex cover heuristics (all optimizations)	56.5(27)	91.0(25)	52.8(9)
Matrix vector product Time on the fine grid (sec)	0.0727	0.057	0.0907

Figure 10: Solve Time (number of iterations)

These experiments show that our methods provide significant improvement over a random MIS, especially on complex domains. Moving coarse vertices, to cover fine ones, did not help on meshes that do not have curved surfaces, as is expected, and provide some improvement the mesh with curved surfaces. The vertex orderings, within each category when applicable, effects these results a small amount, particularly on the "Pure MIS" data; we see about a 10-15% variance with different randomization schemes. Thus one should consider that the standard deviation rather high for this data - but we have consistently observed the dramatic benefit of the modified MIS, that is reflected in this data. The ineffectiveness of ordering the exterior vertices first in the "Pure MIS" experiments is surprising. Other vertex orders (than the ones we used) provided small variations the iteration count - some better and some worse. Also this code is somewhat "tuned" for these problems, and a few others, in that we refined the algorithm the best average performance for these our test problems; but only one code (executable) was used in all of these experiments. Thus we feel that our modified MIS heuristics are effective - especially on domains with complex geometry.

4 Conclusion

We have presented a set of simple methods for the optimization of automatically generated coarse grids for multigrid equation solvers. Our numerical results have shown that our methods are effective, for the class of problems that we have addressed. Additionally we have shown that multigrid has promise in providing a robust and scalable solution methods for unstructured FE problems with complex domains and large jumps in material coefficients.

Some potential areas of future work are:

- Incorporate more sophisticated computational geometry techniques to investigate their effectiveness relative to our simple techniques.
- Compare the effectiveness of our algebraic methods of constructing the coarse grids with the restriction operators provided by user supplied coarse grids, and FE implementations construction of the coarse grid operator.

Acknowledgments. This work is supported by DOE, and we would like to thank Steve Ashby for his constant support of our efforts. We wish to thank Jim Demmel for his numerous suggestions and careful reading of the manuscript. We gratefully acknowledge Argonne National Laboratory for the use of their IBM SP for the program development and numerical results presented in this paper. Also, we would like to thank R.L. Taylor at the University of California, Berkeley, for his helpful comments, and his providing and supporting FEAP.

References

[1] Mark Adams. A parallel maximal independent set algorithm. Technical report, University of California, Berkeley, 1998.

[2] S. Balay, W.D. Gropp, L. C. McInnes, and B.F. Smith. PETSc 2.0 users manual. Technical report, Argonne National Laboratory, 1996.

[3] Tony F. Chan and Barry F. Smith. Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes. *Proceedings of the Seventh Annual International Conference on Domain Decomposition*, 1994.

[4] James Demmel. *Numerical Linear Algebra.* SIAM, 1997.

[5] J. Fish, V. Belsky, and S. Gomma. Unstructured multigrid method for shells. *International Journal for Numerical Methods in Engineering, Vol. 39, pg. 1181-1197*, 1996.

[6] J. Fish, M. Pandheeradi, and V. Belsky. An efficient multilevel solution scheme for large scale non-linear systems. *International Journal for Numerical Methods in Engineering, Vol. 38, pg 1597-1610*, 1995.

[7] Herve Guillard. Node-nested multi-grid with delaunay coarsening. Technical Report 1898, Institute National de Recherche en Informatique et en Automatique, 1992.

[8] Mark T. Jones and Paul E. Plassman. A parallel graph coloring heuristic. *SIAM J. Sci. Comput., Vol. 14 No. 3, pp. 654-669*, 1993.

[9] S. Kacau and I. D. Parsons. A parallel multigrid method for history-dependent elastoplacticity computations. *Computer methods in applied mechanics and engineering, Vol.108*, 1993.

[10] George Karypis and Kumar Vipin. Parallel multilevel k-way partitioning scheme for irregular graphs. *Supercomputing*, 1996.

[11] J. Ruge. *AMG for Problems of Elasticity.* 1986.

[12] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. Technical Report CMU-CS-96-140, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1996.

[13] Barry Smith, Petter Bjorstad, and William Gropp. *Domain Decomposition.* Cambridge University Press, 1996.

[14] Dafna Talmor. *Well spaced points for numerical methods.* PhD thesis, Carnegie Mellon University, Pittsburgh PA 15213-3891, 1997.

[15] Sivan Toledo. Improving memory-system performance of sparse matrix-vector multiplication. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[16] P. Venek, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. In *7th Copper Mountain Conference on Multigrid Methods*, 1995.

[17] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method, 4th ed.* McGraw-Hill, London, 1989.