Engineering with Computers (2006) 22: 61–74
DOI 10.1007/s00366-006-0015-0

ORIGINAL ARTICLE

Lori Freitag Diachin · Patrick Knupp
Todd Munson · Suzanne Shontz

A comparison of two optimization methods for mesh quality improvement

Received: 22 September 2004 / Accepted: 8 November 2005 / Published online: 19 May 2006

Abstract We compare inexact Newton and block coordinate descent optimization methods for improving the quality of a mesh by repositioning the vertices, where the overall quality is measured by the harmonic mean of the mean-ratio metric. The effects of problem size, element size heterogeneity, and various vertex displacement schemes on the performance of these algorithms are assessed for a series of tetrahedral meshes.

Keywords Mesh quality improvement · Mesh optimization · Mesh smoothing

1 Introduction

Mesh vertex repositioning algorithms have been used for many years to improve solution accuracy and efficiency; see, for example, [1–4]. Repositioning techniques vary widely in the time required to implement and modify the algorithm and in the computational cost and effectiveness when applying the algorithm, usually with a tradeoff between these criteria. Laplacian smoothing, for example, is easy to implement and inexpensive to apply but can produce tangled meshes. Moreover, this method is limited to the creation of smooth meshes, while vertex repositioning can address other meshing needs such as equidistribution of volumes [5], shape improvement [6], or adaptive *r*-refinement [7]. These more complex tasks can usually be posed as numerical optimization problems in which an objective function is defined measuring one or more mesh properties. This objective function can then be optimized by repositioning the vertices, leading to improvement in the mesh properties measured.

When approaching the vertex repositioning problem from an optimization perspective, a natural idea is to formulate a single objective function measuring global mesh quality. This global objective function is typically constructed by accumulating contributions from each local measure of quality into a scalar function of the positions of all free vertices in the mesh.[1] We consider two approaches for numerically optimizing the global objective function: an *all-vertex* approach where the positions of all free vertices are moved simultaneously within a single iteration, and a *single-vertex* approach where the position of only one vertex is modified at a time. We employ an *inexact Newton* method as our all-vertex optimization algorithm and a *block coordinate descent* method as our single-vertex algorithm in which a Newton method is applied to solve each coordinate descent subproblem. The goal of this paper is to determine when one of these methods is preferable to the other, where preference can include the ease with which the method can be implemented and modified, the computational and memory requirements for applying the method, and the accuracy and quality of the mesh produced, perhaps as a function of computation time. A complete answer should consider all these characteristics.

The preferred method may differ depending on the circumstances. For example, the block coordinate descent method may be better suited to quickly finding an approximate solution, while the inexact Newton method may be more suitable for calculating a highly accurate solution. Factors that may be significant in determining

L. Freitag Diachin (✉)
Lawrence Livermore National Laboratory, Livermore, CA, USA
E-mail: diachin2@llnl.gov

P. Knupp
Sandia National Laboratories, Albuquerque, NM, USA
E-mail: pknupp@sandia.gov

T. Munson
Argonne National Laboratory, Argonne, IL, USA
E-mail: tmunson@mcs.anl.gov

S. Shontz
University of Minnesota, Minneapolis, MN, USA
E-mail: shontz@cs.umn.edu

[1]An important alternative to mesh optimization often used by the unstructured mesh community employs a series of local objective functions.

the preferred approach include the objective function, quality metric, desired accuracy in the resulting mesh, mesh type (structured vs. unstructured), dimension (planar vs. volume), element type (simplicial vs. non-simplicial), problem size, mesh heterogeneity and anisotropy, and the degree and manner in which the initial mesh differs from the optimal mesh. Algorithm implementations also have a significant impact, since a simple implementation can be much slower than a more sophisticated version.

In this paper we report the results of an initial exploration of these factors to determine the circumstances in which the inexact Newton method or the block coordinate descent method may be preferred. To make the study manageable, we limit the number of free parameters and consider a fixed mesh type, quality metric, and objective function template. In particular, we use tetrahedral meshes, the mean-ratio quality metric for isotropic elements, and a template targeting average quality improvement. The free parameters investigated are the problem size, element homogeneity, initial mesh configuration, and desired degree of accuracy in the resulting mesh.

2 Problem statement

2.1 Element and mesh quality

An unstructured mesh consists of a finite set of vertices $\mathscr{V}$ and elements $\mathscr{E}$, where $|\mathscr{V}|$ denotes the number of vertices and $|\mathscr{E}|$ the number of elements. The set of vertices on the boundary of the mesh is denoted by $\mathscr{V}_{\mathscr{B}}$, while the set of interior vertices, that is, those not on the boundary, is denoted by $\mathscr{V}_{\mathscr{I}}$. Let $x_v \in \Re^n$ denote the coordinates for vertex $v \in \mathscr{V}$. For surface and volume meshes $n = 3$, while for planar meshes $n = 2$ (in this paper we only consider volume meshes). Moreover, $x \in \Re^{n\times|\mathscr{V}|}$ refers to the collection of all vertex coordinates. Each element $e \in \mathscr{E}$ consists of a small subset of the vertices and the edges between these vertices, where $|e|$ is the number of vertices referenced by element e, $\mathscr{V}_e$ refers to the vertices referenced by e, and $x_e \in \Re^{n\times|e|}$ the matrix of vertex coordinates for e.

Associated with the mesh is a continuous function $q{:}\Re^{n\times|e|} \rightarrow \Re$ measuring one or more geometric properties of an element as a function of the vertex positions.[2] In particular, $q(x_e)$ measures the *quality* of element e, where we assume a larger value of $q(x_e)$ indicates a higher quality element. A specific function q is referred to as an element *quality metric*. Many functions can serve as quality metrics, so the quality of an element is not uniquely defined. For example, there are different metrics to measure the shape, size, and orientation of elements. In general, *useful* quality metrics possess other properties in addition to continuity, but a discussion of this topic is beyond the scope of the present study. See, for example, [8].

The overall quality of the mesh is measured by a *template function* $\mathscr{Q} : \Re^{|\mathscr{E}|} \rightarrow \Re$ taking as input the vector of element quality metrics, $\prod_{e\in\mathscr{E}} q(x_e)$, where $\prod$ denotes the Cartesian product. The mesh quality depends on both the choice of the specific element quality metric q and the particular template function $\mathscr{Q}$ used to combine them. Useful template functions can be constructed from the arithmetic or other means.

2.2 The Mean-ratio metric

An important variable in this study is the choice of quality metric. In general, we expect the study results could vary significantly depending on whether or not one were to choose shape metrics as opposed to size, smoothness, or other metrics. For this initial algorithm comparison, we focus on the mean-ratio shape-quality metric. Other shape metrics such as aspect ratio or condition number would likely give similar timing results; we plan to study these metrics and others not explicitly focused on shape in future work.

Let S be a $n \times n$ matrix with det $(S) > 0$. The mean ratio μ of S is the scalar

$$\mu(S) = \frac{n\det(S)^{2/n}}{\|S\|_F^2},$$

where $\|S\|_F = \sqrt{\mathrm{tr}(S^T S)}$ is the Frobenius norm. One can readily show that $0 < \mu(S) \le 1$. To apply the mean ratio to element quality, assume each vertex of the element is connected to n edges (and therefore n other vertices) belonging to the element.[3] Let x_i be the coordinates of vertex i, and let x_k be the coordinates of another vertex in the element connected to v_i by an edge. Construct the matrix $A^{(i)}$ whose columns are the vectors $x_k - x_i$ for each adjacent vertex v_k in the element. The columns are ordered to preserve element orientation so that the element has locally nonpositive volume if det $(A^{(i)}) \le 0$ for any vertex; such elements are called "inverted." Let W be a $n \times n$ reference matrix for the ideal element shape (e.g., an equilateral reference triangle is often used for triangular elements). This reference matrix is found by constructing W from the ideal element in the same way A is constructed for the mesh element. For each element vertex i let $\mu_i = \mu(A^{(i)} W^{-1})$ be the mean ratio at element vertex i.

Finally, the mean ratios of the element vertices are averaged to form an element quality metric symmetric in the element vertex indices.[4] We use the arithmetic mean,

[2]For hybrid meshes, the exact definition of q can change depending on the element type. However, we assume that the quality metric, shape, for example, is the same for every element.

[3]This approach excludes elements such as pyramids but includes triangles, tetrahedra, wedges, quadrilaterals, and hexahedra.

[4]We show in [6] that averaging is unnecessary in the case of triangular or tetrahedral elements.

although different means could also be applied. The shape quality of element e is then

$$q_e = \frac{1}{|e|} \sum_{i \in \mathscr{V}^e} \mu_i.$$

As one would expect, this metric is scale, translation, and rotation invariant. Furthermore, $0 < q_e \leq 1$ with $q_e = 1$ only when the element attains the ideal reference shape. We do not define the mean ratio for matrices with nonpositive determinants. Therefore, the shape quality of "inverted" elements is not defined. For further details on shape metrics see [9].

2.3 Quality improvement problem

To improve the overall quality of the mesh, we assemble the local element qualities using a template function $\mathscr{Q}$. We compute an $x^* \in \Re^{n \times |\mathscr{V}|}$ such that x^* is an optimal solution to

$$\max_x \quad \mathscr{Q}\left(\prod_{e \in \mathscr{E}} q(x_e)\right) \tag{1}$$

subject to the constraint that $x_{\mathscr{V}_\mathscr{B}} = \bar{x}_{\mathscr{V}_\mathscr{B}}$, where $\bar{x}_{\mathscr{V}_\mathscr{B}}$ are the coordinates for the boundary vertices. Note that additional constraints can be added if the locations for some of the interior vertices also need to be fixed. If the objective function for this optimization problem is uniformly concave as a function of $x_{\mathscr{V}_\mathscr{I}}$, that is, the Hessian matrix, $\nabla^2_{x_{\mathscr{V}_\mathscr{I}}, x_{\mathscr{V}_\mathscr{I}}} \mathscr{Q}(\cdot)$, is uniformly negative definite, then an x^* solving this optimization problem exists and is unique. If the objective function is not concave, then one can only hope to find a local maximizer for the optimization problem and may instead compute a critical point.

We use the harmonic mean template for all our numerical results. This template produces the objective function

$$\mathscr{Q}_{\text{hm}} := \frac{|\mathscr{E}|}{\sum_{e \in \mathscr{E}} 1/q_e(x_e)}.$$

This objective function is maximized precisely when the denominator is minimized. Therefore, the optimization problem we solve is

$$\min_x \quad \mathscr{F}_{\text{hm}}(x) := \frac{1}{|\mathscr{E}|} \sum_{e \in \mathscr{E}} \frac{1}{q_e(x_e)} \tag{2}$$

subject to the same boundary constraints as (1). The objective function in this case is continuous on the set of noninverted meshes and bounded below by one and minimizes the average inverse mean-ratio metric. We further assume the initial set of coordinates is feasible; that is, the corresponding mesh does not contain inverted elements. We also require the improved mesh to be noninverted, which translates to the implicit constraint det $(A^{(i)}) > 0$ for every element vertex. There is no need to implement these constraints explicitly, however, because the denominator in the inverse mean ratio acts as a barrier to element inversion.

3 Improvement algorithms

Many algorithms can be applied to compute a solution to the quality improvement problem (2).[5] In this paper, we consider the block coordinate descent and inexact Newton methods [10, 11]. The block coordinate descent method optimizes the location of a single vertex at a time by applying an optimization algorithm to a restricted problem in which only the coordinates for the given vertex are allowed to move. This optimization step is repeated for each of the other vertices in the mesh. This iterative repositioning stops when the norm of the gradient of the global objective function is small. The inexact Newton method, on the other hand, constructs a quadratic approximation to the global objective function at the current iterate and computes a solution to this quadratic program by solving a large system of equations. A new iterate is then found for which the objective function has decreased.

The block coordinate descent algorithm solves a sequence of small optimization problems to improve the global objective function but has a slow asymptotic convergence rate, while the inexact Newton method solves a large quadratic optimization problem at every iteration but has a fast asymptotic convergence rate. If the global objective function in (2) is uniformly convex in the free variables, then both algorithms converge to the same solution x^* [10]. However, if the objective function is not convex, as is often the case in mesh optimization, we can only say that if the block coordinate descent method converges to x^*, then x^* is a critical point for the optimization problem (2) and x^* may not be a local minimizer.

Moreover, when the global objective function is not convex, the optimization subproblems solved by the block coordinate descent method may have either no solution or many local minimizers. However, for the inverse mean-ratio metric, even though the global objective function is not convex everywhere, one can prove that the objective function for each subproblem of the block coordinate descent method is strictly convex [12, 13] and the feasible region is compact. Therefore, each of the subproblems has a unique solution. Note that the inexact Newton and block coordinate descent algorithms may not converge to the same critical point.

[5]Recall that (2) minimizes the inverse mean-ratio objective function, so the stated algorithms use minimization terminology. However, the same algorithms can be used for the maximization problem (1).

3.1 Block coordinate descent method

The block coordinate descent method modifies a single vertex at a time by applying one iteration of a Newton method to the subproblem obtained by fixing the rest of the vertices at their current coordinates. That is, we compute a direction d where for vertex $v \in \mathscr{V}_{\mathscr{I}}$, the vth component of d is obtained by solving the system of equations

$$\nabla^2_{v,v}\mathscr{F}_{\text{hm}}(x^k)d_v = -\nabla_v\mathscr{F}_{\text{hm}}(x^k).$$

The remaining components of d are set to zero. Note that we need only the Hessian matrix with respect to vertex v, so the complete Hessian matrix for the global objective function does not need to be computed. The direction is obtained by directly factoring the $n \times n$ local Hessian matrix and applying it to the right-hand side of the problem. An iterative method is not needed in this case because the system of equations is very small. For the metric used, the Hessian matrix is positive definite, so the factorization can always be computed [12, 13].

Having obtained a search direction, we then use an Armijo linesearch [14] to obtain a new iterate with improved quality. In particular, we compute the smallest nonnegative integer m such that

$$\mathscr{F}_{\text{hm}}(x^k + \beta^m d) \leq \mathscr{F}_{\text{hm}}(x^k) + \sigma\beta^m\nabla\mathscr{F}_{\text{hm}}(x^k)^T d,$$

where $\beta \in (0,1)$ and $\sigma \in (0,1)$ are chosen constants. When searching along the direction, all points where the resulting mesh is degenerate or inverted are rejected; the objective function value is treated as positive infinity in these cases. Most of the time, the full step is accepted, and we perform only one function evaluation. To judge progress, we need to consider the quality of only the elements in which vertex v appears, since the quality of elements outside this set does not change when the location of vertex v is modified. Hence, the Armijo linesearch is computationally inexpensive. To make both the Armijo linesearch and Hessian matrix computations efficient for the block coordinate descent method, we precompute a mapping from each vertex to the elements referencing the vertex.

We then update $x^k = x^k + \beta^m d$ and choose a different vertex to optimize. The order in which the vertices are traversed in each pass is determined by reverse breadth-first search [15] starting from the vertex farthest from the origin. This reverse breadth-first search is applied to the given mesh to improve the locality of reference, making the code more efficient.

An iteration of the block coordinate descent method consists of a single repositioning of each interior vertex. Once each of the interior vertices has been repositioned, we proceed to the next iteration by setting $x^{k+1} = x^k$. The local improvement process is repeated until the gradient of the global objective function is less than some tolerance.

3.2 Inexact Newton method

The inexact Newton method computes a direction d by solving the system of equations

$$\nabla^2_{x_{\mathscr{V}_{\mathscr{I}}},x_{\mathscr{V}_{\mathscr{I}}}}\mathscr{F}_{\text{hm}}(x^k)d = -\nabla_{x_{\mathscr{V}_{\mathscr{I}}}}\mathscr{F}_{\text{hm}}(x^k)$$

by applying the conjugate gradient method with a block Jacobi preconditioner [15], where x^k is the current iterate. When the Hessian matrix is indefinite, the conjugate gradient method can terminate with a direction of negative curvature. This direction of negative curvature is used as our search direction. Having obtained a search direction, we then use an Armijo linesearch [14] to obtain a new iterate with a sufficiently improved global objective function value. This linesearch is the same as the block coordinate descent linesearch but must consider the improvement in the global objective function instead of the improvement to only the elements referenced by a single vertex. Many global objective function evaluations can be performed to obtain sufficient decrease when far from a solution, particularly when a direction of negative curvature is encountered. In a neighborhood of a solution, the full step is taken, and only one global objective function evaluation is performed.

The gradient and Hessian of the objective function are calculated by assembling the gradients and Hessians for each element function into a vector and symmetric sparse matrix. Only the upper triangular part of the Hessian matrix is stored in a block compressed sparse row format. Each block corresponds to a coordinate in the original problem. The gradient and Hessian elements corresponding to fixed vertices on the boundary of the domain are ignored.

The preconditioner consists of the Hessian with respect to the (i,i) coordinates, resulting in a block diagonal preconditioner, where each block consists of a $n \times n$ matrix. An LDL^T factorization of each diagonal matrix is performed when calculating the preconditioner. The preconditioner is applied by setting $y = L^{-T}(D^{-1}(L^{-1}x))$. We store D^{-1} so that the middle product consists of a few multiplications, instead of a few divisions. Each diagonal block of the Hessian matrix is positive definite even though the overall Hessian is indefinite in general [12, 13], so the preconditioner can always be computed.

3.3 Implementation characteristics

Our implementations of the block coordinate descent method and the inexact Newton method have been coded with a bias toward achieving high performance with minimal memory requirements. These algorithms were coded using the same infrastructure and have been extensively refined so that we can draw comparisons between them. Both codes use analytic gradient and Hessian evaluations, since finite-difference approximations for the inverse mean-ratio metric are inefficient by comparison.

To make both the inexact Newton and block coordinate descent methods run faster, instead of computing with W^{-1} in the mean ratio metric, we precompute a QR factorization of W, where Q is an orthogonal matrix with determinant equal to one, R is an upper triangular matrix, and $W^{-1} = R^{-1}Q^T$. The Q^T matrix can then be ignored in the mean-ratio metric when using this form of W^{-1} due to properties of the Frobenius norm and determinant. Hence, $\mu(A^{(i)}\ W^{-1})$ is equivalent to $\mu(A^{(i)}\ R^{-1})$. The latter definition is computationally advantageous, because the function, gradient, and Hessian matrices take fewer operations to compute than if W^{-1} were stored as a general dense matrix because the fact that R^{-1} is an upper triangular matrix can be exploited.

One of the main computational tasks associated with the inexact Newton method is obtaining an efficient evaluation for the Hessian of the global objective function. This computation requires obtaining the Hessian for each individual element function. The code for calculating the gradient of the element function uses the reverse mode of differentiation [16] on the element quality metric. The Hessian calculation uses the forward mode of differentiation on the gradient evaluation and matrix–matrix products for efficiency. For the general case with an upper triangular matrix, it takes 750 floating-point operations to compute the function, gradient, and Hessian for each element. The inexact Newton method specialized for an equilateral reference element uses 576 floating-point operations per element in the function, gradient and Hessian evaluations. The other significant computational task is performing the matrix–vector products required by the conjugate gradient method to compute the search direction.

Good locality of reference in the Hessian evaluation and matrix–vector products is obtained by reordering the vertices and elements in the initial mesh by applying a reverse breadth-first search. The ordering starts by selecting the (boundary) vertex farthest from the origin. A breadth-first search of the vertices in the mesh is then performed and the order they are visited is tracked. We then reverse the order in which the vertices were visited to obtain the reordering for the problem. Once the vertices are reordered, the elements are then reordered according to when they are visited by the Hessian evaluation. This reordering is used by both the block coordinate descent and inexact Newton methods.

Each iteration of the block coordinate descent method consists of computing the gradient and Hessian for each subproblem, obtaining the direction, and computing each improving point. The gradient and Hessian evaluation is the most expensive operation. To minimize the number of floating-point operations performed per iteration of the block coordinate descent method, separate evaluation routines for taking the gradient and Hessian with respect to each vertex in the inverse mean-ratio metric are used because we only need to compute a small portion of the gradient and Hessian. An iteration of the block coordinate descent method calls each of the four routines once per element because we need to get gradient and Hessian information for each vertex in the mesh. Using the original function, gradient, and Hessian calculation from the inexact Newton algorithm would cost 3000 floating-point operations per element in the general case with an upper triangular weight matrix, while an implementation using four separate routines in which only the unnecessary computations are removed from the original calculation consumes 1,447 floating-point operations per element.

More improvement can be made to these local routines by applying an even permutation to the input data (coordinates for both the trial and reference elements) to put the desired coordinate in the last position, computing the QR factorization for each permuted reference element, and then taking the gradient and Hessian with respect to the last vertex of this equivalent function definition. When all these operations are performed offline for the given weight matrices and the permuted weight matrices are stored, the cost is reduced to 520 floating-point operations per element in the general case with an upper triangular weight matrix. This strategy is memory efficient only when the number of weight matrices is small. If the number of weight matrices is large (there is a different weight matrix for each element, for example) and memory consumption is a concern, then the permutation and factorization can be performed by the code as needed without explicitly storing the permuted weight matrices. This change leads to a cost of approximately 900 floating-point operations per element, depending on the technique used to compute the QR factorization. The savings attributed to using this approach are significant when compared to using a single Hessian evaluation routine; the cost per iteration of an implementation using the original calculation is over three times that of an efficient implementation.

The local routines can be further refined when using an equilateral reference element to reduce operation counts. In particular, for the equilateral weight matrix, R^{-1} is the same for each of the permuted reference elements. The block coordinate descent method specialized for an equilateral reference element uses 480 floating-point operations per element in the function, gradient and Hessian evaluations.

In addition to computational effort, we are also interested in evaluating the memory footprint of each method as the problem size increases. Our implementation of the block coordinate descent method for tetrahedral elements has a steady-state memory requirement of approximately $23|\mathscr{V}| + 12|\mathscr{E}|$ integer values. The formula for memory usage of the inexact Newton method is more complicated due to the storage requirements for the Hessian matrix and is given by $64|\mathscr{V}| + 18|\mathscr{E}| + 19N$ integer values, where N denotes the number of off-diagonal blocks in the Hessian matrix. On the tetrahedral meshes tested, the number of off-diagonal blocks is bounded above by the number of elements in the mesh. Therefore, the memory usage is approximately $64|\mathscr{V}| + 37|\mathscr{E}|$ integer values for the inexact Newton method, about three times the storage required for the block coordinate descent method.

Conclusion 1: Analytic comparison of time per iteration, memory requirements, and coding effort. Our block coordinate descent method for solving the mesh improvement problem with the inverse mean-ratio metric specialized to the equilateral reference element case is faster per iteration and consumes less memory than the inexact Newton method but has a slow asymptotic convergence rate. This conclusion can be drawn by looking at the number of floating-point operations performed per element by the function, gradient, and Hessian evaluation routines for each method, 480 for the block coordinate descent method and 576 for the inexact Newton method. Furthermore, *the inexact Newton method requires a higher initial coding investment*. In particular, routines to assemble the global Hessian matrix from the element Hessian matrices, construct the preconditioner, and perform the preconditioned conjugate gradient method to compute the direction need to be written. Once this infrastructure has been built, however, changing to a new metric requires only an efficient computation of the gradient and Hessian for the entire element. To change the metric for the block coordinate descent method, four different routines need to be implemented, one for computing the gradient and Hessian with respect to each vertex of the element. Moreover, if the new metric cannot exploit the permutation and QR factorization scheme, then a different technique must be devised to obtain an efficient block coordinate descent method.

Note that if a different weight matrix were associated with each element, then the inexact Newton method could be faster per iteration if the permuted weight matrices are not stored (approximately 900 floating-point operations would be needed in the function, gradient, and Hessian evaluations for the block coordinate descent method versus 576 for the inexact Newton method) or could consume less memory if the permuted matrices are stored ($23|\mathscr{V}| + 60|\mathscr{E}|$ integers for the block coordinate descent method versus $64|\mathscr{V}| + 49|\mathscr{E}|$ integers for the inexact Newton method).

4 Numerical experiments

In this section, we report the results of numerical tests designed to determine when the block coordinate descent and inexact Newton techniques are preferred using a subset of the criteria given in Sect. 1. The implementations specialized for an equilateral reference element are used for all the numerical tests. We solve the optimization problem (2) on a series of tetrahedral meshes generated with the CUBIT [17] and GRUMMP [18] mesh generation packages. We consider two different computational domains, duct and clipped cube, and show sample meshes on these geometries in Fig. 1. In this paper, we study the effects of three different problem parameters on the time taken to reach x^*: problem size, element heterogeneity, and initial mesh configuration. For each parameter studied, we create a suite of test meshes in which we isolate the parameter of interest and allow it to vary, while simultaneously holding the other parameters as constant as possible.

Because the objective function used for these problems is not convex, it is likely that different local solutions to the optimization problem exist. It is therefore possible that the solutions would not be the same for the block coordinate descent and inexact Newton methods. To ensure that this is not affecting our study, we computed the difference between corresponding vertex locations computed by the two solution techniques. If the difference between two vertices was less than 1% of the average element edge length of the mesh, we considered them to be in the same location. In all cases, the maximum difference between vertex locations computed by the inexact Newton and block coordinate descent method solutions was well within our tolerance; a typical value was 0.0001% or less. Thus the two techniques are converging to the same solution. To study the effect of initial mesh configuration, we perturb the vertices in different ways from the optimal mesh. For these cases, we also compared the final solutions to the initial optimal mesh from which the vertices were perturbed. Again, the differences are well within our 1% tolerance level with the maximum percent difference being 0.21%. Thus the inexact Newton and block coordinate descent methods methods are finding the same optimal solution that is found in the unperturbed case, even when the perturbation is large.

In each of the following subsections, we give the problem characteristics of the test suite in terms of the number of vertices and elements, initial mesh quality as evaluated by the inverse mean-ratio metric, and specific parameter values used such as the perturbation of the optimal mesh. We then provide performance results for both the inexact Newton and block coordinate descent methods. For the inexact Newton method, the maximum number of solver iterations is 500, and the maximum number of conjugate gradient subiterations is 100, while for the block coordinate descent method, the maximum number of iterations (sweeps over the free

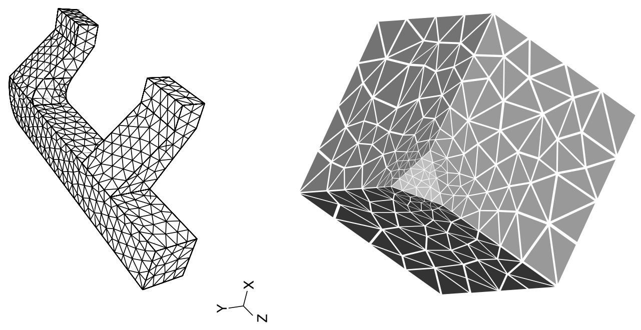

Fig. 1 Sample meshes on the duct and clipped cube geometries

vertices) is 1,000. In both cases, the solution is considered to be optimal when the two-norm of the gradient of the global objective function is less than 1.0×10^{-6}. All experiments were run on a dedicated 1.6 GHz Linux workstation running Red Hat Enterprise 3.

4.1 Increasing problem size

To test the effect of increasing the problem size, we use CUBIT to generate tetrahedral meshes with uniform quality and element size but with an increasing number of vertices for the duct geometry shown in Fig. 1. In Table 1, we give the number of vertices and elements, along with the average, median, and standard deviation normalized by the average value, denoted σ_n, for the inverse mean-ratio metric and element volume. One can see that within each mesh, we achieve roughly uniform element size and shape quality distributions while increasing the problem size from 4,104 elements to 965,759 elements. In addition, the element quality characteristics are similar across this suite of initial meshes as the problem size increases. In particular, the initial mesh quality is quite good, with an average inverse mean-ratio value of 1.2 (the optimal is one) and a maximum value ranging from 2.2 to 5.

In Table 2, we give the number of iterations, I, and time, T_{100}, required to achieve the optimal solution. In all cases, both I and T_{100} are significantly smaller for the inexact Newton solver because of its superior asymptotic convergence rate. As the problem size increases, the disparity in time to solution increases monotonically from a factor of 6.4 to a factor of 40.

However, a highly accurate solution is often not required in mesh smoothing applications. Therefore, the time required to reach a partially improved mesh is also of interest. As a particular example, we include the time required to achieve 50% of the optimal solution as defined by the global objective function value, T_{50}, in Table 2. For every mesh in this test suite, the block coordinate descent method takes less time than the inexact Newton method to reach this suboptimal solution, typically by a factor of 1.5.

To examine this behavior more closely, we recorded the objective function and gradient values at each iteration. A typical time history is shown for the Duct15 mesh in Fig. 2. Because the inexact Newton method converges to the optimal solution much more quickly than the block coordinate descent method, we show the complete time history of the inexact Newton solver and only the corresponding portion of the block coordinate descent method. Because the initial mesh quality is very good, both methods make significant progress toward the optimal solution in their first few iterations. However, significant setup overhead is associated with computing the sparsity pattern of the Hessian matrix for the inexact Newton method because the edges in the mesh need to be sorted. During this setup time, the block coordinate descent method completes one iteration through the mesh, which is sufficient to achieve 46% of optimality. Clearly, there is a point in time at which the inexact Newton solution is closer to optimal than the block coordinate descent solution. We call this point the *crossover point*, and it is highlighted with an asterisk in Fig. 2. In the Duct15 case, the mesh is 96% optimized when the crossover point occurs.

Based on these results, it is natural to ask the questions: ‘‘What is the percent improvement achieved at the crossover point?’’ and ‘‘What is the time required by each method to achieve a certain level of optimality?’’ as the problem size increases. To answer these questions, for each method we plot the percent improvement obtained, the number of block coordinate descent iterations, and the percentage of time spent in setup by each solver at the crossover point as a function of an increasing problem size in the top graph in Fig. 3. In all cases, the mesh is nearly optimal at the crossover point even though the number of block coordinate descent iterations completed is quite small, typically less than five. As the problem size increases, the setup time for the inexact Newton solver is greater than 25% of the time to reach the crossover point and typically less than 10% for the block coordinate descent method. In this case, the setup time is the primary factor in determining which solver reaches suboptimal solutions faster.

In the bottom graph in Fig. 3, we show the ratio of the time required by the inexact Newton solver and block coordinate descent solver to achieve certain levels of improvement. Each line in the graph represents a different problem size, and the flat line at one represents the point at which the preferred method changes. Data above this line indicates that the block coordinate descent method is faster; data below indicates the opposite. In this case, we see that the smaller problem sizes are

Table 1 Initial mesh characteristics for increasing problem size on the duct geometry

Mesh	$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{E}\rvert$	Inverse mean ratio				Element volume		
			Average	Median	σ_n	Maximum	Average	Median	σ_n
Duct20	1,067	4,104	1.208	1.176	0.115	2.2	1,167	1,176	0.285
Duct15	2,139	9,000	1.210	1.179	0.116	2.1	532	519	0.304
Duct12	4,199	19,222	1.209	1.182	0.111	2.1	249	237	0.327
Duct10	7,297	35,045	1.120	1.170	0.106	2.2	136	128	0.310
Duct8	13,193	65,574	1.19	1.162	0.105	2.4	73	68	0.320
DuctBig	177,887	965,759	1.18	1.160	0.109	4.9	4.1	2.91	0.587

Table 2 Number of iterations, total time, and time to achieve a 50% optimal solution as problem size increases

$\|\mathcal{V}\|$	Newton			Coordinate descent		
	I	T_{100}	T_{50}	I	T_{100}	T_{50}
1,067	4	0.05	0.015	33	0.32	0.005
2,139	5	0.13	0.025	46	1.1	0.011
4,199	5	0.34	0.056	74	4.2	0.037
7,297	5	0.69	0.106	105	11.6	0.081
13,193	5	1.4	0.213	146	31.0	0.152
177,887	8	44.3	4.52	548	1,738	2.47

more affected by the setup time differences, but as the problem size exceeds 20,000 elements, the methods behave similarly. In particular, it takes roughly twice as long to compute suboptimal meshes using the inexact Newton approach for a wide range of desired improvement percentages. As the improvement percentage increases, the inexact Newton method becomes more competitive, but only when nearly optimal meshes are desired does the inexact Newton method outperform the block coordinate descent method.

Conclusion 2: Performance comparison for optimization of homogeneous tetrahedral meshes. The results of the tests in this subsection show that *the block coordinate descent method typically outperforms the inexact Newton method when a suboptimal mesh is acceptable*. This was true for a wide range of problem sizes. We can explain the outperformance by noting that the inexact Newton method has high setup costs associated with computing the sparsity pattern of the Hessian matrix, which cannot be amortized over a large number of iterations because suboptimal meshes require relatively few iterations. *If an optimal mesh is required, then the inexact Newton method outperforms block coordinate descent*. The test problems used in this study were generated using the GRUMMP and Cubit tetrahedral meshing algorithms. Because these packages create, for the most part, reasonably well-shaped elements, the initial meshes used in these tests are not too far from optimal, as measured by the mean ratio shape-quality metric. Therefore, *when optimizing homogeneous, reasonably high-quality tetrahedral meshes to further improve shape quality, the block coordinate descent method is preferable because a few iterations creates a near-optimal mesh without the start-up costs of the inexact Newton method.* The reader is cautioned that this conclusion may not extend to other important applications of mesh optimization using the mean ratio, particularly if the initial mesh is far from optimal.

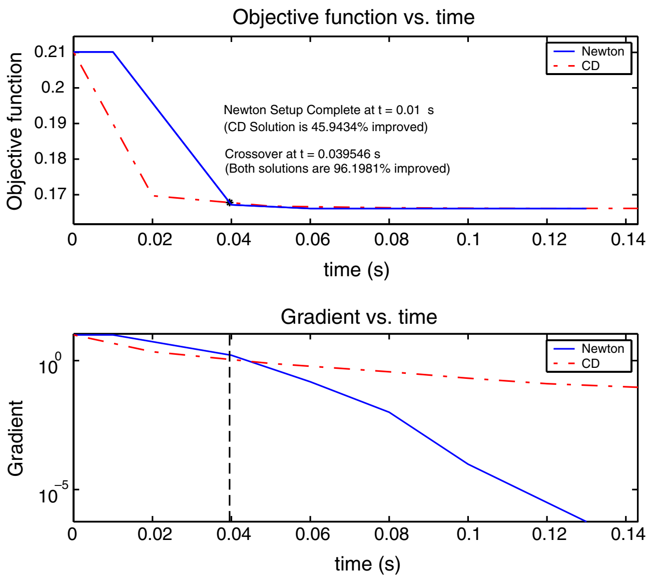

Fig. 2 Objective function value and gradient norm as a function of time for the Duct15 mesh

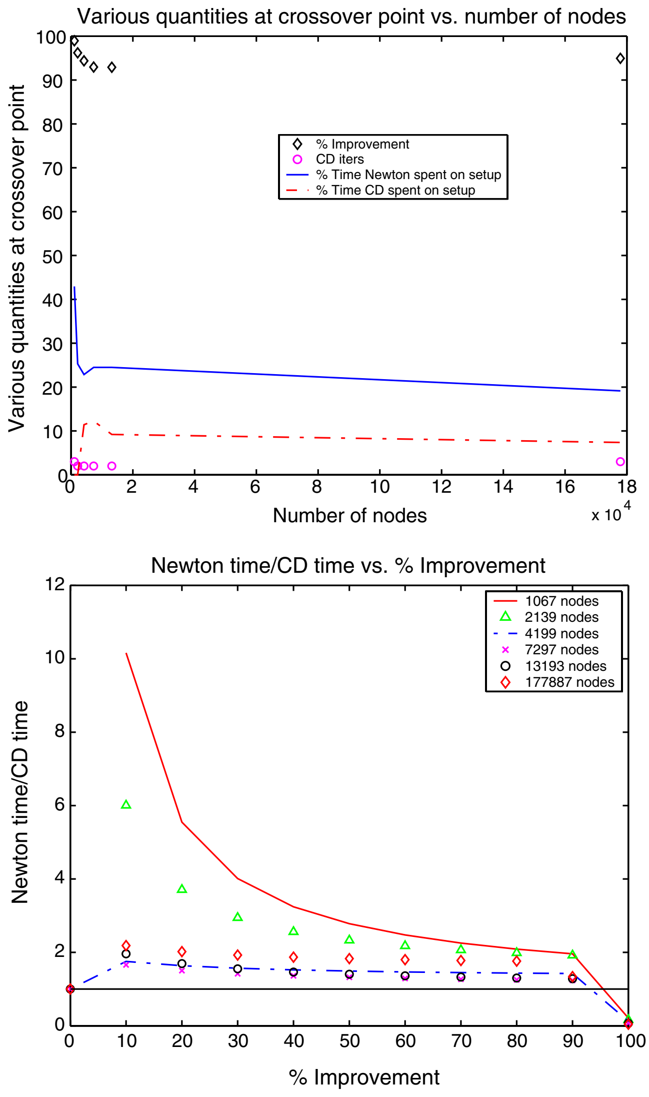

Fig. 3 Various quantities of interest at the crossover point as the problem size increases (*top*) and the ratio of the times required by the inexact Newton and block coordinate descent solvers to achieve certain levels of improvement (*bottom*)

4.2 Element size heterogeneity

Our second test suite was generated using GRUMMP with the aim of testing the effect of element size (volume) heterogeneity on the two algorithms. A simple geometry consisting of the unit cube with a small tetrahedral volume clipped from one corner was used to create graded meshes with grid points clustered around that corner. GRUMMP parameters that determined the smallest element size and gradation of the mesh were manipulated to create a series of meshes with roughly the same number of vertices and element quality distribution but with different ranges of element sizes. We note that in generating the initial meshes for these test cases, we did not take advantage of GRUMMP's mesh quality improvement tools.

In Table 3, we give the statistics for these meshes in terms of numbers of vertices, elements, shape quality distribution, and element volume. The numbers of vertices, although not identical, are all within 6% of 10,470. The normalized standard deviation of the element volumes and the ratio of the maximum-sized element to the minimum-sized element show that the element size varies dramatically within a given mesh. The shape quality distributions across the meshes in the test suite are similar; the average shape quality is nearly the same as in the uniform element size test cases, but the normalized standard deviation is higher, indicating a wider range of individual element qualities. In particular, the maximum mean ratio of the heterogeneous element size meshes exceeds a value of 15 in all cases, whereas it is approximately 2.5 in the uniform element size test suite.

In Table 4, we give the number of iterations and the times to reach the optimal and 50% improved solutions as the element heterogeneity, measured by the ratio of maximum element volume to minimum element volume, increases. As with the uniform element distribution test suite, the inexact Newton method is significantly faster than the block coordinate descent method when the optimal solution is desired. If a 50% improved solution is desired, however, the block coordinate descent method outperforms the inexact Newton method by a factor that ranges from 1.5 to 4, compared to the factor of 1.5 for the uniform distribution case of the previous subsection.

We examine the convergence history of one particular case, Hetero4, to obtain insight into this result. Figure 4 shows the value of the objective function and gradient as a function of time for both the inexact Newton and block coordinate descent methods. The block coordinate descent method maintains a steep initial decrease in the objective function value, while the inexact Newton method has more difficulty. In particular, many of the initial iterations of the inexact Newton method encounter directions of negative curvature because the global objective function is not convex at those iterates. Typically, a small step is taken when such directions are found by the conjugate gradient method. Thus, the superior asymptotic convergence rates of the inexact Newton method are not evident until approximately two seconds have elapsed.

Because the inexact Newton method has difficulty with these problems in the initial iterations, the block coordinate descent method has the time to take several iterations, and the mesh in nearly 100% improved at the crossover point in all cases. As before, the inexact Newton method uses twice as much time in setup as does the block coordinate descent method. Unlike the uniform element size test suite, however, this is not the dominant factor in determining the crossover time because both methods use less than 5% of their total time in setup.

For this test suite, in Fig. 5 we show the ratio of the time required by the inexact Newton method and the time required by the block coordinate descent method to reach a number of different levels of improvement. For comparison, we also show the curves for the two uniform element size test cases, Duct10 and Duct8, which tightly bracket the number of elements in the clipped cube meshes. The normalized standard deviation values for the Duct 10 and Duct8 meshes are 0.310 and 0.320, respectively. In almost all cases, the block coordinate descent method is two to three times faster than the inexact Newton method to reach a desired level of improvement. In fact, in several cases, the ratio of the times required to achieve higher levels of improvement actually increases rather than decreases as a result of the steep initial convergence of the block coordinate descent method. Furthermore, this method is more competitive than the inexact Newton method on the heterogeneous element size meshes than on the uniformly sized element meshes.

Conclusion 3: Performance comparison for optimization of heterogeneous tetrahedral meshes. As element size heterogeneity increases, mesh quality as defined by the

Table 3 Initial mesh characteristics for heterogeneous element size distributions

Mesh	$\|\mathscr{V}\|$	$\|\mathscr{E}\|$	Inverse mean ratio			Element volume				
			Average	Median	σ_n	Maximum	Average	Median	σ_n	Maximum/ minimum
Hetero1	10,318	54,132	1.271	1.171	0.330	17.1	1.84×10^{-5}	1.10×10^{-5}	1.11	5.5×10^{3}
Hetero2	9,883	56,184	1.274	1.172	0.371	30.2	1.77×10^{-5}	6.04×10^{-6}	1.46	2.3×10^{5}
Hetero3	10,926	58,610	1.275	1.173	0.413	58.6	1.70×10^{-5}	3.89×10^{-7}	1.74	7.2×10^{7}
Hetero4	11,057	59,985	1.272	1.173	0.322	16.1	1.66×10^{-5}	1.59×10^{-6}	2.08	4.2×10^{6}

Table 4 Number of iterations, total time, and time to achieve 50% optimal solution as the element heterogeneity increases

Mesh	Newton			Coordinate descent		
	I	T_{100}	T_{50}	I	T_{100}	T_{50}
Hetero1	16	4.24	0.386	674	122	0.128
Hetero2	13	3.53	0.345	708	132	0.192
Hetero3	21	5.41	0.432	505	93	0.207
Hetero4	15	4.22	0.641	554	109	0.168

mean ratio metric decreases for these tests. Thus, the initial meshes used in this part of the study are further from optimal than were the initial meshes in the part of the study leading to Conclusion 2. Even so, *the inexact Newton method is still the preferred method if the optimal solution is desired. When suboptimal meshes are acceptable, the block coordinate descent method outperforms the inexact Newton method.* In contrast to Conclusion 2 however, where high start up costs are the primary factor, this conclusion is primarily due to the directions of negative curvature encountered by the inexact Newton method. Negative curvature typically leads to the linesearch taking a small step that does not improve the global objective function very much. In such cases, the global objective function is evaluated many times during the linesearch, leading to a significant increase in the time required to complete the iteration. The negative curvature is attributed to the lack of convexity in the objective function.

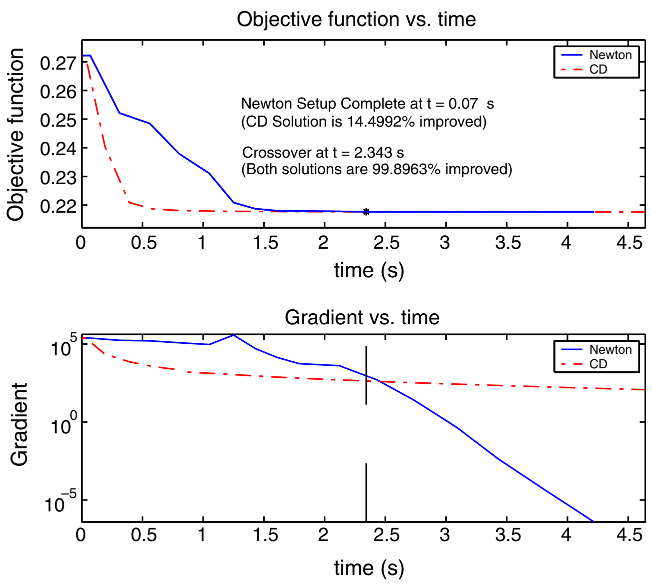

Fig. 4 Objective function value and gradient norm as a function of time for the Hetero4 mesh

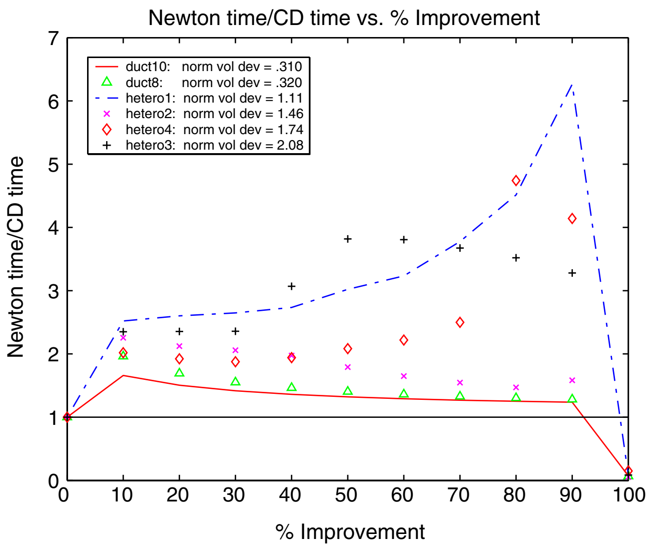

Fig. 5 Ratio of the inexact Newton and block coordinate descent times for various levels of improvement in the objective function

4.3 Initial mesh configuration

The final mesh test suite was designed to investigate the effect of the degree and manner in which the initial mesh differs from the optimal mesh. To address this issue, our approach was to apply systematic or random perturbations of the optimal positions of the interior mesh vertices. We started with the optimized DuctBig mesh and applied three different perturbation schemes that involved random, translational, and oscillatory movement of the mesh vertices. In all three cases, we consider perturbations applied to all of the vertices and to randomly chosen vertex subsets that contained 1, 10, and 25% of the total number of vertices. The formulas for the perturbations are as follows:

Random: $x_v = x_v + \alpha\, r$, where r is a vector containing random numbers generated using the function `rand` and α is a multiplicative factor controlling the degree of perturbation. For this test suite, we chose $\alpha = 0.001$, 0.005, 0.01, and 0.05.

Translational: $x_v = x_v + \alpha\, s$, where s is a direction vector giving the coordinates to be shifted and α is again a multiplicative factor controlling the degree of perturbation. In this case we considered a "right" shift (R) with $s = [1\ 0\ 0]^T$ and a "northeast" shift (NE) with $s = [1\ 1\ 0]^T$ and chose $\alpha = 0.1$, 0.2, and 0.3. In the case of the NE shift, we also considered a series of meshes with large values of $\frac{2}{\sqrt{2}}\alpha = 1$, 2.5, 5, 7.5, 10, 12.5, and 15 which resulted in very large perturbations of the optimal mesh. Results for the latter perturbation are discussed in Sect. 4.3.2, while results for all other perturbations can be found in Sect. 4.3.1.

Oscillatory: $x_v = x_v + \alpha \sin(\omega\, x_v)$, where α and ω control the amplitude and frequency of the perturbation, respectively, and sine is performed on each component of x_v. For this test suite, we considered three different frequencies $\omega = 0.01$, 0.05, and 0.1, and for each frequency, two different amplitudes $\alpha = 0.01$ and 0.05.

These perturbations can be characterized in terms of both amplitude α and wavelength λ. The random perturbation corresponds to a zero-wavelength perturbation and the translational to an infinite-wavelength perturbation. The wavelength of the oscillatory perturbation can be computed from the frequency by $\lambda = 2\pi/\omega$, and for the values of ω used here, λ ranges from 63 to 628. As a point of comparison the the average edge length of the mesh which is approximately 3.6.

We considered two series of tests. In the first, we perturbed all or some of the vertices by a small amount to determine the effect of the type of perturbation on the mesh. In the second test suite, we perturbed all of the vertices a large amount to change the scale of the perturbation.

4.3.1 *Small perturbations*

All vertices perturbed. For this test suite, we perturbed all of the vertices a small amount (the values of α given above that are less than 0.5). The resulting quality characteristics of the meshes as the perturbations increase do not vary significantly, and we do not include the details. In particular, the average inverse mean ratio value is approximately 1.13, the ratio of the standard deviation to the average is approximately 0.093, and the maximum inverse mean ratio is approximately 2.21 in all cases.

In Table 5, we give the iterations and time to reach the 100 and 50% improved solutions for the cases in which we perturb all of the vertices in the mesh according to the formulas and parameters given above. As with the other test suites, if a highly accurate solution to the optimization problem is sought, the inexact Newton method outperforms the block coordinate descent method in every case. If the 50% improved solution is sought, the block coordinate descent method outperforms the inexact Newton method for all the test cases.

In Fig. 6, we examine the ratio of the time required by the inexact Newton method and the block coordinate descent method as the desired degree of optimality increases. In all cases considered, as a more improved solution is sought, the inexact Newton method looks increasingly attractive. For the random test suite, however (top left), the block coordinate descent method always outperforms it up to 90% improvement. For the translational and oscillatory test suites (top middle and top two right), the performance of the block coordinate descent method is not as good. In these cases, the local nature of the block coordinate descent method can only slowly eliminate the long-wavelength errors introduced by the perturbation scheme. In contrast, the inexact Newton method has access to global information and is able to overcome this difficulty. Thus the block coordinate descent method still outperforms the inexact Newton method for approximate solutions, but at best the mesh is only 75% optimal at the crossover point. This degrades to approximately 63% for the oscillatory case as the degree of perturbation increases. The setup time is again a contributing factor in determining the crossover point and requires about 30% of the solution time for the inexact Newton method compared to approximately 10% for the block coordinate descent method.

Table 5 Number of iterations and total time to achieve 100 and 50% of the optimal solution as the element heterogeneity increases

Perturbation	α	Newton			Coordinate descent		
		I	T_{100}	T_{50}	I	T_{100}	T_{50}
Rand ($\lambda = 0$)	0.001	3	11.8	7.46	325	104	3.90
	0.005	4	23.6	7.53	387	324	3.84
	0.01	4	23.3	7.53	414	427	3.89
	0.05	4	22.6	7.42	476	664	3.84
Oscillatory ($\lambda = 63$)	0.01	4	22.7	7.56	279	244	5.59
	0.05	4	21.2	7.50	338	368	5.63
Oscillatory ($\lambda = 125$)	0.01	4	24.1	7.95	374	353	4.77
	0.05	5	22.4	8.02	435	550	4.74
Oscillatory ($\lambda = 628$)	0.01	4	25.4	8.29	415	425	4.75
	0.05	4	24.7	8.32	477	655	4.76
Translational (R) ($\lambda = \infty$)	0.1	5	30.6	8.31	502	789	4.72
	0.2	7	36.7	8.35	528	904	4.79
	0.3	10	47.2	7.42	545	956	8.24
Translational (NE) ($\lambda = \infty$)	0.1	5	31.1	8.32	496	800	4.64
	0.2	7	38.1	8.66	522	905	4.66
	0.3	11	59.4	13.5	539	962	7.99

Some vertices perturbed. To determine whether the number of vertices that we perturb affects the relative performance of the two solvers, we give the ratio of the inexact Newton method and block coordinate descent method times to achieve various levels of improvement (see the bottom two rows of Fig. 6) when a subset of the vertices is perturbed. The percentage of vertices that were randomly selected to be perturbed is 1, 10, and 25% or 1,779, 17,789, and 44,472 vertices, respectively. We show results for only a subset of the cases analyzed; the results for the cases not shown are qualitatively the same. In all cases, the inexact Newton method is unable to outperform the block coordinate descent method when a suboptimal solution (90% improved or less) is sought. This result is particularly interesting for the oscillatory and translational perturbations (bottom two middle and right, respectively). In these cases, because only a subset of the vertices were perturbed, the long-wavelength errors that affected the performance of the block coordinate descent method are no longer evident, and the block coordinate descent method is again able to quickly approach the optimal solution.

Conclusion 4: Sensitivity of optimization methods to the initial mesh. Three different types of perturbations (random, translational, and oscillatory) from the optimal mesh were used in the test problems in this subsection to study the sensitivity of each method to the initial mesh. *For all types of perturbations, wavelengths, and amplitudes, the numerical results show once again that the block coordinate descent method outperforms the inexact Newton method if suboptimal solutions to improving the mean ratio metric are acceptable.* This conclusion is due in part to the high setup costs associated with the inexact Newton method. That said, as evidenced in Table 5, long-wavelength perturbations present difficulties for the block coordinate descent

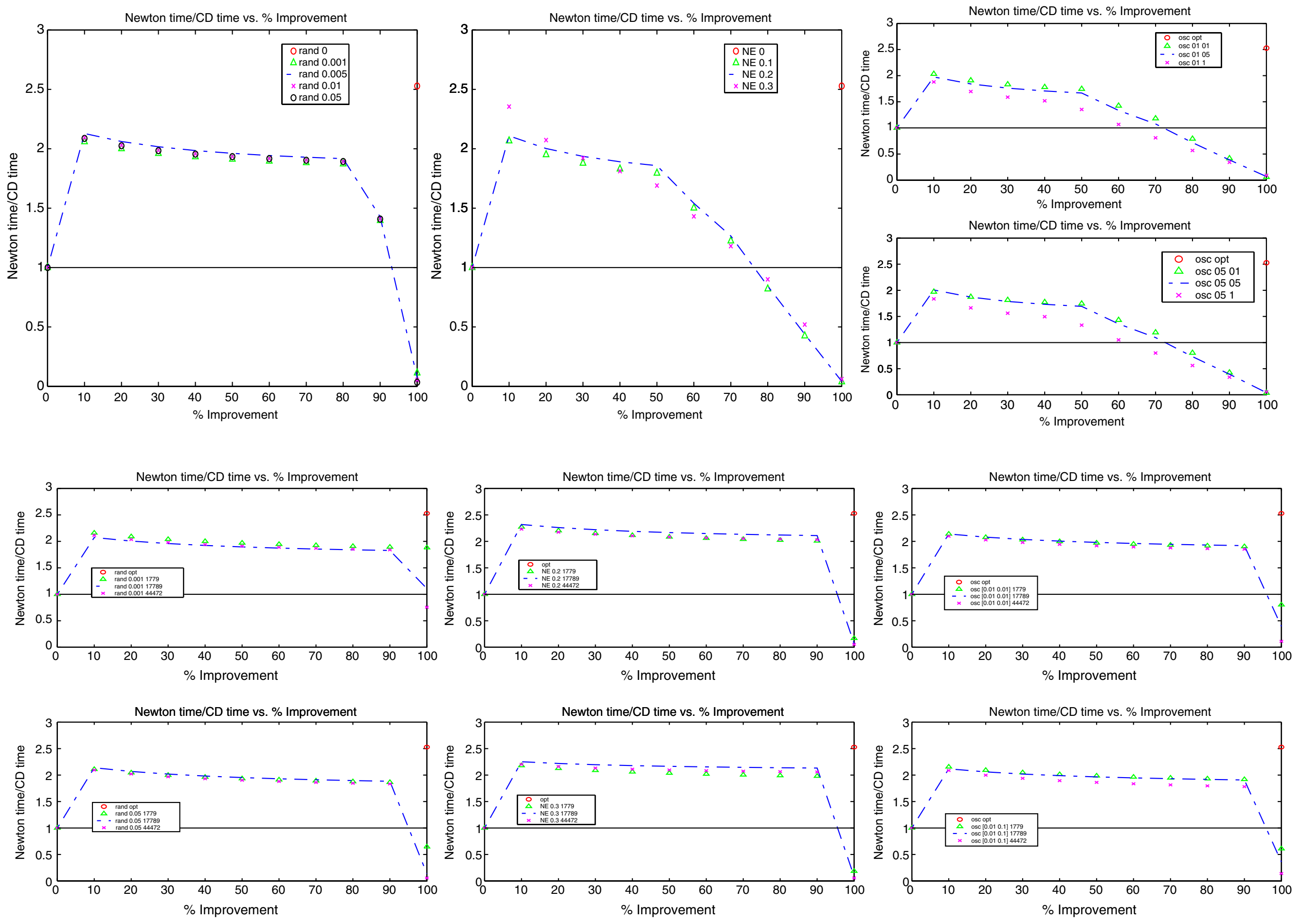

Fig. 6 Ratio of the times required by the inexact Newton and block coordinate descent solvers to achieve certain levels of improvement for the random (*top left*), NE translational (*top middle*), and oscillatory (*top two right*) perturbations and for the cases where a subset of vertices are perturbed randomly (*bottom two left*), NE translational (*bottom two middle*), and oscillatory (*bottom two right*)

method due to the lack of access to global information. *The inexact Newton method outperforms block coordinate descent if an optimal solution is required.*

4.3.2 Large perturbations

For this test suite, we considered large perturbations of the NE translational type. In Table 6, we give the initial mesh quality characteristics for the DuctBig meshes with translational perturbations corresponding to $\frac{2}{\sqrt{2}}\alpha = 1$, 2.5, 5, 7.5, 10, 12.5, and 15. For comparison, we note that the average edge length in the mesh is 3.6. The value of α corresponds to the maximum amount a node was moved in the mesh. Some nodes may move a smaller distance, determined by an iterative backtracking procedure, to prevent inverted elements in the initial mesh. As α increases, the quality of the initial mesh clearly decreases although that degradation is not monotonic. The average mean ratio metric value goes from an average value of 1.14 to over 2.63 with the worst quality element exceeding a mean ratio metric value of 1.80×10^5. Element volumes vary similarly.

The solution time required for each method was considerably more than what was needed for smaller perturbations. In general, as the perturbation amplitude increased and the mesh quality degraded, the total time to solution increased. It is interesting to note that as the perturbation amplitude increased, the time and number of iterations required by the inexact Newton method to find an optimal solution also increased, but remained

Table 6 Initial mesh characteristics for increasing NE shift perturbation of the duct geometry

$\frac{2}{\sqrt{2}}\alpha$	Inverse mean ratio			Element volume	
	Average	σ_n	Maximum	Maximum	σ_n
Optimal	1.135	0.093	2.21	30.2	0.540
1	1.155	0.179	2.72	30.2	0.546
2.5	1.423	2.25	283	30.2	0.593
5	2.062	8.95	1.59×10^4	50.1	0.755
7.5	2.550	58.1	1.43×10^5	52.7	0.841
10	2.508	13.0	2.02×10^4	57.8	0.865
12.5	2.636	69.9	1.80×10^5	65.5	0.869
15	2.468	7.75	9.47×10^3	59.8	0.869

approximately constant for the block coordinate descent method. Even so, the inexact Newton method outperformed the block coordinate descent method for 100% improved solutions by a factor of 11 $\left(\text{for } 2/\sqrt{2}\alpha = 1\right)$ to 2.5 $\left(\text{for } \frac{2}{\sqrt{2}}\alpha = 15\right)$. In most cases, the block coordinate descent method was 30–80% faster for approximate solutions that were 50% improved, but in the case of $\frac{2}{\sqrt{2}}\alpha = 2.5$ the inexact Newton method was faster for both the highly improved solution and the approximate solution.(Table 7)

The top image in Figure 7 shows the ratio of times required by the inexact Newton method and block coordinate descent method to achieve certain levels of improvement. These results are considerably more interesting than the corresponding plots for increasing problem size and element size heterogeneity shown in Figs. 3 and 5. In particular, in a number of cases there appear to be several crossover points and if a very approximate solution is sought (less than 40% improved), the inexact Newton method is preferred to the block coordinate descent method. Interestingly if the desired level of improvement is between 40 and 90%, the block coordinate descent method is preferred. This behavior is very unlike what was seen in the earlier test cases. To help explain this more clearly, we include the time history of the objective function value and gradient norm for the perturbation $\frac{2}{\sqrt{2}}\alpha = 1$ in the bottom two images of Fig. 7. The block coordinate descent method is unable to make early progress toward the optimal solution because it does not have access to global information. The inexact Newton method is able to overtake it although its progress is sporadic. After approximately 20 iterations, the block coordinate descent method begins to make rapid progress and significantly improves the mesh in a few iterations, overtaking the inexact Newton method. As the mesh gets closer to the optimal solution, the quadratic convergence rates of the Newton method allow it to reach the optimal solution more quickly than the block coordinate descent method.

Conclusion 5: Sensitivity of optimization methods to large perturbations. Large amplitude perturbations from the optimal mesh result in significantly longer optimization times for both the block coordinate descent and inexact Newton methods. More importantly, *unlike many of the test cases discussed earlier, the tests in this subsection identified situations for which the inexact Newton method outperforms the block coordinate descent method, even when suboptimal solutions are acceptable.*

Table 7 Number of iterations and total time to achieve 100 and 50% of the optimal solution as the perturbation increases

$\frac{2}{\sqrt{2}}\alpha$	Newton			Coordinate descent		
	I	T_{100}	T_{50}	I	T_{100}	T_{50}
1	19	100	26.4	575	1,106	19.3
2.5	61	355	28.4	616	1,266	35.0
5	68	388	42.2	632	1,313	36.5
7.5	83	452	52.7	635	1,329	34.1
10	89	503	45.7	635	1,347	30.2
12.5	115	644	59.4	635	1,339	34.0
15	94	525	48.2	633	1,328	26.8

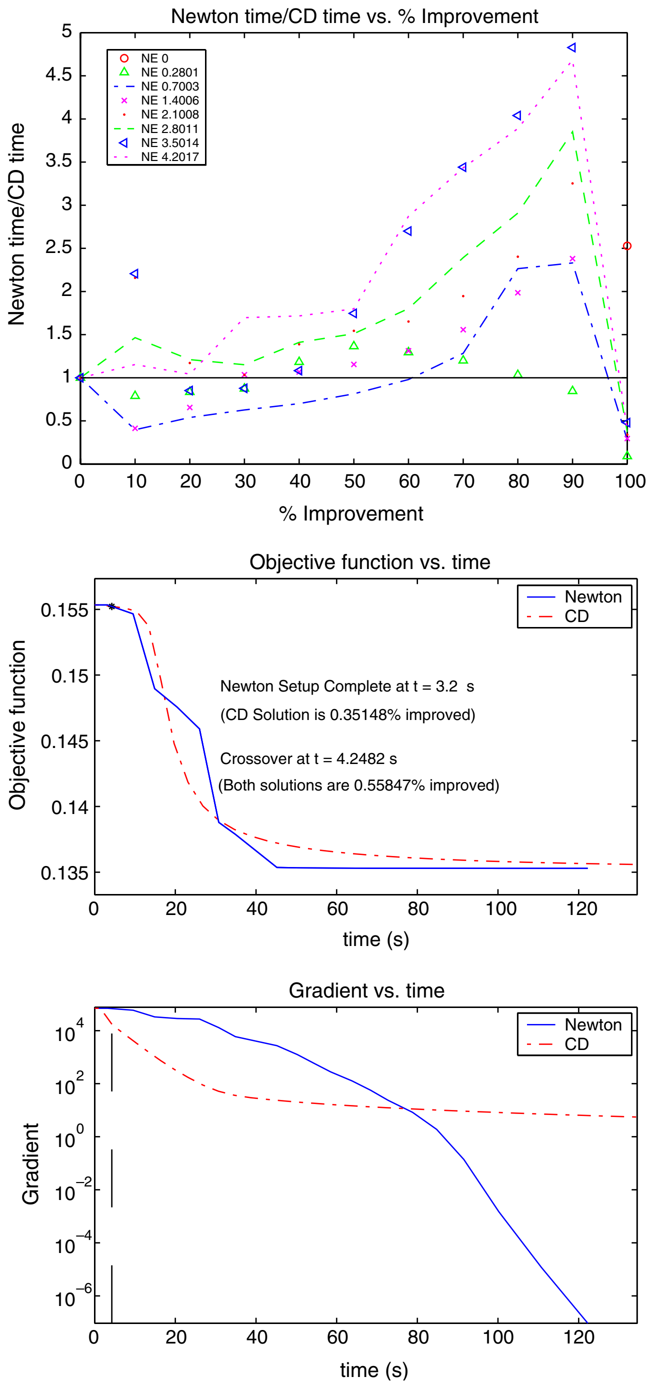

Fig. 7 The ratio of the times required by the inexact Newton and block coordinate descent solvers to achieve certain levels of improvement as the perturbation size increases. Note that the numbers in the key correspond to the maximal actual perturbation scaled by the average element length

5 Future work

The numerical results show that the block coordinate descent method is often best for making fast improvements in the shape quality of tetrahedral meshes with an equilateral reference element. The inexact Newton method is best when a very accurate solution to the optimization problem is necessary. We note that a 50% optimization level may be somewhat misleading in that it does not indicate the degree of the accuracy in the solution. For most of the duct meshes, the iterate at a 50% optimization level has four or five digits of accuracy in the objective function value. If the initial quality of the mesh is very poor, however, then the iterate at a 50% optimization level may not have any digits of accuracy in the objective function value. Alternative definitions of 'solution accuracy' will be explored in future work.

This study was limited to ideal shape improvement of tetrahedral meshes using an equilateral reference element; triangular, quadrilateral, and hexahedral meshes were not included in the study. Preliminary experience with planar quadrilateral meshes shows that the crossover point for our two codes generally occurs earlier, but further study is warranted. In particular, the efficiency of the block coordinate descent method is related to the structure of the reference element. Some of the efficiency in our implementation for equilateral elements is lost when using a different reference element. The storage required when applying a fast block coordinate descent method to an anisotropic mesh in which the reference element is different for each element is significant. While the storage can be reduced by applying a permutation and factorization as needed, the performance of the block coordinate descent method degrades.

Furthermore, because the global objective function is not convex, a trust-region method for the inexact Newton code may perform better since it can handle directions of negative curvature more rigorously. A limited-memory quasi-Newton method may also outperform the block coordinate descent method when obtaining an approximate solution in a small amount of time. Efficient implementations of these methods can be based on our existing infrastructure developed for the inexact Newton and block coordinate descent methods.

In conclusion, there are many factors which can affect whether or not one should use a block coordinate descent method or an inexact Newton method for mesh optimization. The present work identifies some of the potentially important factors and develops a methodology for further investigations on this topic. Future work will consider remaining open questions including consideration of other mesh element types, quality metrics, objective function templates, movement of nodes along the boundary of the domain, and different applications of mesh optimization such as r-adaptivity in which the reference element will not be constant as it was in this study.

Acknowledgments The initial version of the analytic gradient for the inverse mean-ratio metric for tetrahedral elements was provided by Paul Hovland (Argonne National Laboratory). The clipped cube mesh image was provided by Carl Ollivier-Gooch (University of British Columbia). The work of the first, second, and third authors was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contracts W-7405-Eng-48 (UCRL-CONF-205150), DE-AC-94AL85000, and W-31-109-Eng-38, respectively. Part of the work of the fourth author was performed while a member of the Center for Applied Mathematics at Cornell University, supported by Sandia National Laboratories, Cornell University, the National Physical Science Consortium, and NSF grant ACI-0085969.

References

1. Bank R, Smith B (1997) Mesh smoothing using a posteriori error estimates. SIAM J Num Anal 34:979–997
2. Canann SA, Stephenson MB, Blacker T (1993) Optismoothing: an optimization-driven approach to mesh smoothing. Fin Elem Anal Des 13:185–190
3. Parthasarathy VN, Kodiyalam S (1991) A constrained optimization approach to finite element mesh smoothing. Fin Elem Anal Des 9:309–320
4. Zavattieri P, Dari E, Buscaglia G (1996) Optimization strategies in unstructured mesh generation. Int J Num Methods Eng 39:2055–2071
5. Castillo J (1991) A discrete variational grid generation method. SIAM J Sci Stat Comp 12:454–468
6. Freitag L, Knupp P (2002) Tetrahedral mesh improvement via optimization of the element condition number. Int J Numer Methods Eng 53:1377–1391
7. Anderson D (1990) Grid cell volume control with an adaptive grid generator. Appl Math Comp 35:209–217
8. Knupp P (2001) Algebraic mesh quality metrics. SIAM J Sci Comp 23:193–218
9. Knupp P (1999) Matrix norms and the condition number. Proceedings of 8th International Meshing Roundtable, pp 13–22
10. Bertsekas DP (1999) Nonlinear programming, 2nd edn. Athena Scientific, Belmont
11. Nocedal J, Wright SJ (1999) Numerical optimization. Springer, Berlin Heidelberg New York
12. Munson TS (2004) Mesh shape-quality optimization using the inverse mean-ratio metric. Preprint ANL/MCS-P1136-0304, Argonne National Laboratory, Argonne
13. Munson TS (2004) Mesh shape-quality optimization using the inverse mean-ratio metric: tetrahedral proofs. Technical memorandum ANL/MCS-TM-275, Argonne National Laboratory, Argonne
14. Armijo L (1966) Minimization of functions having lipschitz-continuous first partial derivatives. Pac J Math 16:1–3
15. Saad Y (2003) Iterative methods for sparse linear systems, 2nd edn. SIAM, Philadelphia
16. Griewank A (2000) Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, Philadelphia
17. Sandia National Laboratories (2003) Albuquerque, New Mexico. CUBIT 80.1 Mesh Generation Toolkit
18. Ollivier-Gooch CF (1998–2002) GRUMMP—Generation and refinement of unstructured mixed-element meshes in parallel. http://www.tetra.mech.ubc.ca/GRUMMP