High Fidelity Interval Assignment

Scott A. Mitchell[1]

Abstract. *Quadrilateral meshing algorithms impose certain constraints on the number of* intervals *or mesh edges of the curves bounding a surface. When constructing a conformal mesh of a collection of adjoining surfaces, the constraints for all of the surfaces must be simultaneously satisfied. These constraints can be formulated as an integer linear program. Not all solutions to this problem are equally desirable, however. The user typically indicates a goal (soft-set) or required (hard-set) number of intervals for each curve. The hard-sets constrain the problem further, while the soft-sets influence the objective function.*

This paper describes an algorithm for solving this interval assignment problem. The objective is to have a solution such that for each curve the positive or negative difference between its goal and assigned intervals is small relative to its goal intervals. The algorithm solves a series of linear programs, and comes close to minimizing the maximum vector of such differences. Then the algorithm solves a nearby mixed-integer linear program to satisfy certain "sum-even" constraints. The algorithm reliably produces intervals that are very close to the user's desires and is easily extendible to new constraints. Earlier versions of the algorithm were slower than alternative algorithms, but this is no longer a significant issue; in practice, the running time of the current version is a minor fraction of the time to mesh, even in models composed of thousands of curves.

keywords. interval assignment, mixed-integer linear programming, conformal meshing, mesh control

1. Introduction

CUBIT[1] is a quadrilateral and hexahedral mesh generation *toolkit*, meaning that the user has access to a variety of meshing algorithms and support tools. The usual steps for creating a mesh within CUBIT are the following: first, a model geometry is generated or imported; second, meshing schemes are assigned to curves, surfaces, and volumes of the model; third, the element size of the mesh is specified; fourth, for the more structured surface meshing algorithms, additional parameters called *corners*[2] are determined; fifth, the exact number of mesh edges (called *intervals*) for each curve is determined; sixth, curves are meshed, then the surfaces; and lastly the volumes are meshed. Many of these steps are automated or are in the process of being automated, and the user has the ability to specify parameters such as the meshing scheme and element size.

This paper deals with the fifth step, determining the number of intervals for each curve. This step also arises in other meshing toolkits besides CUBIT. The significance of this step is that once it is complete, the surfaces may be meshed independently (except for volume meshing restrictions such as sweep directions) and the mesh will be conformal. We assume that the meshing schemes and corners are given, and that the user has specified a *soft-set* goal or a fixed *hard-set* number of *intervals* for each curve. The key difficulty is a global one: for each surface, its meshing algorithm and associated corners requires a certain relationship between the number of intervals on each bounding curve. Volume meshing can impose additional constraints. Many CUBIT models are composed of many adjoining volumes; usually each curve is contained in two or more surfaces, and its intervals are constrained in some way by all of them. Hence the interval constraints are linked throughout the model.

These constraints are typically[3] assembled into a mixed-integer linear program (MILP)[4]. The various surface meshing algorithms create two types of constraints. First, the more structured algorithms create linear constraints of the form "intervals on opposite sides are equal" or "sum of intervals on two sides are greater than the third." Second, unstructured algorithms create constraints of the form "sum of intervals on all curves are even." These are formulated by constraining an interval sum to be equal to $2k$, where k is an integer. These k variables force the problem to be mixed-integer. (If a problem contains only constraints of the first type, then some previous interval assignment

1*samitch@sandia.gov.* Parallel Computing Sciences Dept., Sandia National Laboratories, Albuquerque, NM 87185. Scott Mitchell was supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research, and works at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AL04-94AL8500. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

algorithms obtain an integer solution to a linear program for free.) Note that for a given choice of meshing algorithms, corners, and hard-set intervals, interval assignment may be infeasible.

Most previous work[3] relies on minimizing the sum of weighted differences between assigned and goal intervals, whereas we essentially minimize the lexicographic vector of weighted differences. We get that the relative change in each curve is small, whereas previous approaches typically change as few curves as possible.

Our algorithm has two steps. The first step attempts to find an integer solution to the linear constraints. We divide the problem into independent subproblems. For each curve in a problem, the number of intervals assigned to it corresponds to a variable. We add two extra *delta* variables for each curve, that compute the positive and negative difference between the assigned-interval and goal-interval. We weight the deltas inversely proportional to the interval goal (to compute relative change). We add another variable M that computes the maximum of the weighted deltas. We relax the variables, allowing them to take on non-integer values. We solve this LP with objective function minimize M. We find a set of *tight* curves, i.e. curves that force M to be as large as it is. We fix the tight curves' intervals to nearby integer values, in effect removing the curves from the LP. We recursively solve the reduced LP until all curves are fixed or all un-fixed curves have intervals equal to their goals, i.e. until M is zero. At the end of the first step we have an integer solution that satisfies all of the constraints except perhaps the sum-even constraints.

A common complication is that fixing all tight curves simultaneously at the chosen integer values may make the problem infeasible; in that case we recursively try to fix only a fraction of these curves. A rare complication is that it is sometimes infeasible to round even a single curve's intervals to an integer given previous rounding; in that case we fix the curve at a non-integer value which is changed to integer in the second step.

In the second step of the algorithm we solve the true MILP, using branch and bound (B&B) to get an integer solution. We suspect a solution near the solution found in the first step. We un-fix the curve-interval variables, but bound them to ranges near their old fixed values. The sum-even k variables are similarly bounded. We minimize the weighted <u>sum</u> of k and curve-interval variables. Even with these small ranges and simple objective function, B&B may take too long; for a given set of bounds it may take exponential time. If this is the case, we try less tight bounds. We have four sets of bounds. If the first step had an integer solution and no curves are hard-set, then for one of these sets of bounds there is a solution. However, there is no guarantee that we can find such a solution within the allotted time.

This technique gives interval assignments that have very high fidelity to the user-desired goals, spreading out necessary changes to reduce mesh distortion. Our techniques appear to be more robust and general than previous approaches. However, our techniques are slower because we iteratively solve the relaxed LP, taking time $O(n^3)$ rather than $O(n^2)$.

Our techniques are practical for models with thousands of curves, depending on the meshing algorithms. Figure 4 shows some real-world models, with about 500 curves each, for which our current algorithm solves the interval assignment problem quickly. To progress beyond thousands of curves, we conjecture that the problem should be iteratively divided into dependent subproblems, solved independently, and stitched together. This is the typical strategy for huge LPs, e.g. airline crew scheduling problems.

The remainder of the paper is organized as follows. Section 2 describes the algorithm in detail. Section 2.1 describes the constraints for the different surface meshing algorithms. Section 2.2 contains some practical remarks on implementing the constraints and the MILP. Section 2.3 motivates our approach. Section 2.4 describes iteratively solving the relaxed LP while Section 2.5 describes solving the bounded MILP. Section 3 gives some examples. Section 3.1 goes through the algorithm steps in detail for a small problem. Section 3.2 illustrates how interval changes typically get distributed. Section 3.3 shows some larger, real-world examples. Section 4 discusses the running time of the algorithm in practice, and discusses possible improvements for large problems. Section 5 concludes with a summary of the results and Section 5.1 discusses future directions.

2. The Interval Assignment Algorithm

2.1 Mesh Scheme Constraints

We consider four representative schemes, called *Paving, Submapping, Mapping,* and *Tri-Mapping.* In CUBIT there are additional schemes that e.g. cut off triangular corners or insert boundary layers that we do not consider here. A *loop* is a connected sequence of curves bounding a surface. A surface may have more than one loop. A curve may appear more than once in a loop. Except for paving, the constraints also depend on which vertices are *corners.* The sequence of curves between successive corners is called a *side*. Let *I(e)* denote the number of intervals subdividing a curve or collection of curves *e*.

Paving is an unstructured quadrilateral meshing algorithm for general surfaces. It simply requires that for each loop the sum of intervals is even. (Note that for *any* quadrilateral meshing scheme, the sum of intervals over <u>all</u> loops must be even.) For each loop *a*,

$$I(a) = 2k \qquad k \in integers \geq 2.$$

Mapping places a structured, rectangular *m* x *n* mesh onto a geometric surface with a single loop. The first step is to pick corners, see Mitchell[2]. This partitions the loop into 4 connected subsequences of curves: *up, right, down* and *left*. For surfaces such as a cylinder, there may be only two opposite sides. Opposite sides must have equal intervals:

$$I(up) = I(down)$$
$$I(right) = I(left)$$

Submapping is a generalization of mapping. It subdivides a surface into regions that can be mapped. The four sides, *up, right down* and *left,* need not be subsequences, and the surface may have more than one loop. For each loop, opposite sides must have equal intervals; see Tam[3], White[5] and Whiteley[6]. There are additional constraints to prevent a loop from overlapping itself in interval-parameter space which we will discuss in future work.

Tri-mapping meshes a surface with a three-sided *m* x *n* x *p* primitive. The single loop bounding a surface is partitioned into three logical sides, *a, b,* and *c*. The constraints are

$$I(a) + I(b) \geq I(c) + 2$$
$$I(a) + I(c) \geq I(b) + 2$$
$$I(b) + I(c) \geq I(a) + 2$$
$$I(a) + I(b) + I(c) = 2k \qquad k \in integers \geq 3.$$

If there are only two bounding curves, then constraints are based on the assumption that the curve with the larger number of goal intervals will be subdivided into two sides after curve meshing. If there is only one curve, e.g. the surface is a disk, then we only have the last constraint $I(a) = 2k$, $k \in integers \geq 3$.

2.2 From Constraints to MILP

The collection of constraints in Section 2.1 for all surfaces of the model are collected in a mixed-integer linear program (MILP). The general form of a MILP is the following

$$\begin{array}{ll} \text{Minimize} & c^T x \\ \text{such that} & Ax = b \\ & Dx \geq e \\ & u \geq x \geq l \\ & x_j \in integers, \quad j \in J \text{, index set} \end{array}$$

Soft-set curves correspond to variables x_i. Hard-set curves contribute to the value of the right hand side of the constraints. Additional "sum-even" k variables (x_j) are used to enforce that certain sums are even. Since a curve must have at least one interval, a lower bound l of 1 is set for each curve variable, and a lower bound of 2 or 3 is set for each "sum-even" variable. Constraints correspond to rows of A or D. The objective function is c. See Section 3.1 for a small example.

We use the library LP Solve[7] to represent and solve MILPs. LP Solve uses a sparse matrix implementation, which is essential for efficiency because the interval assignment problem is sparse; each curve bounds only a few surfaces.

2.3 Objective Function Goals and Prior Approaches

Setting up the constraints presented in Section 2.2 and solving the MILP is technically sufficient to "solve" the interval assignment problem, in that it would be possible to mesh all of the surfaces according to their schemes and corners, and hard-set intervals would be respected. However, the assigned intervals for soft-set curves would be arbitrarily far from the goals. Also, depending on the objective function, without good bounds on the integer variables the MILP might take exponential time to solve. Our objective is to craft an objective function that leads to a solution where

- the intervals for each curve is close to its specified goal.

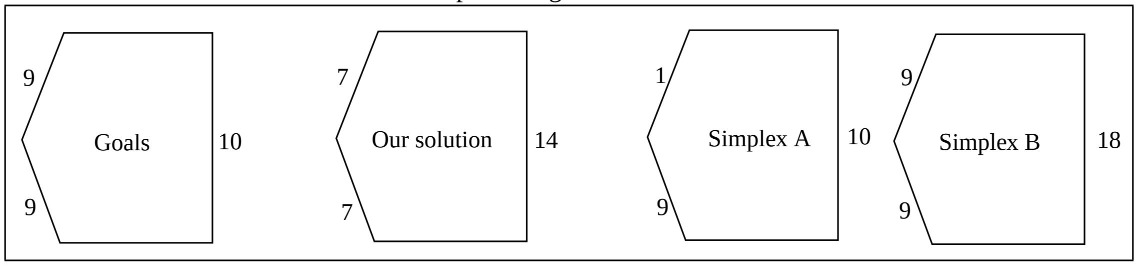

Figure 1. Given the goals on the left, we prefer our solution to those on the right. If a simplex method is used in conjunction with minimizing the weighted sum of interval changes, then, depending on the exact weighting, either Simplex A or Simplex B will be the solution.

The objective function in Tam and Armstrong[3] is to minimize the weighted sum of the *deltas*, where a *delta* is the absolute value of an interval deviation from the goal G. Note $|x\text{-}G|$ is a non-linear function, but it is a standard trick of linear programming to compute it using a sum of two variables, $|x\text{-}G| = D + d$, by using the constraints $D \geq x\text{-}G$ and $d \geq -x+G$, where $D, d \geq 0$. More succinctly we constrain

$$D - d = x - G \qquad D, d \geq 0$$

and in effect minimize $D + d$ so that only one of D or d will be non-zero.

In Tam and Armstrong[3] and our work, the delta weights are chosen so that curves with smaller goals have larger weight. This reflects the fact that a one-interval change in a small curve is a larger relative change than in a long curve. Also, we chose a smaller weight W for D, increasing intervals, than w for d, decreasing intervals. This again reflects relative change. How much smaller W is than w depends on the initial goal. In particular, we use weights W and w approximately $1/G$ and $1.2/(G - 1)$.

The strategy of minimizing the sum of the weighted deltas fails to meet our bulleted objective if a simplex-based[4] linear program solver is used and large deltas occur. Since the simplex method chooses among vertices of the feasible polyhedra, all of the change usually gets assigned to one curve of a side; see Figure 1. Large deltas might be avoided in some cases by adjusting the goals before setting up the MILP, but doing this reliably is a global problem equivalent to interval assignment.

Quadratic programming might meet the bulleted objective by minimizing the sum of the weighted deltas squared. (Actually, one would like to minimize something like the sum over all edges e of the relative change in interval size,

X_e/G_e for $X_e>G_e$ and G_e/X_e for $X_e<G_e$.) Given the slow running time of quadratic programming, it seems unlikely that this could be made into a practical system.

Minimizing the sum of deltas minimizes the total change, but can lead to large individual changes. Our solution is to minimize *M*, the maximum value of the weighted deltas. This maximum *M* can be computed by adding a constraint

$$M \geq W_i D_i + w_i d_i$$

for each curve *i*. Recall that *D* and *d* are the vector of positive and negative deltas, or deviations from the goal intervals; *W* and *w* their weights, which are approximately inversely proportional to the goal; and only one of D_i or d_i will be non-zero.

Minimizing *M* meets the bulleted objective when applied to one pair of opposite sides on a rectangle-primitive surface, or to one paved surface. However it fails to push a curve close to its goal if it is already better than the worst curve, so it could fail miserably in a large model.

We might suppose to get around this defect by considering a combination of the maximum of deltas and the sum of deltas, or by breaking up the problem into subregions in some way. For all such strategies we have explored, we have been able to find a plausible example where that strategy would give a poor solution.

2.4 Our Objective Function & Step 1: The Pseudo-Relaxed Problem

Our highest fidelity solution is to minimize the lexicographic vector of weighted deltas. This is accomplished by iteratively solving a shrinking MILP, fixing one curve at each iteration. At each iteration, we find an optimal solution of the MILP that minimizes *M*, the maximum weighted delta. We choose a curve with weighted delta equal to *M*, such that we cannot reduce its delta without increasing *M*. Such a curve is called *tight*. We fix the intervals of the tight curve at its current (integer) solution value, and remove it from the MILP. We iterate until *M* is zero, that is until all curves are fixed or have zero deltas. This algorithm takes an unreasonable amount of time. Our fast algorithm is a modification of this based on four speed improvements.

The first speed improvement is to relax the integer variables and solve an LP at each step, and round the number of intervals for a tight curve to the nearest integer value. We call this the *pseudo-relaxed* problem. On rare occasions we paint ourselves into a corner. For example, we may round a tight curve to an odd number of intervals and fix it. Suppose there are two curves on a side opposite it, and the rest of the model constrains those two curves to have equal intervals. Then at some future iteration when those two curves are tight, we will be unable to round either one and still have a feasible solution. In that case, we fix the curves to be a non-integer value, and rely on the second phase, Section 2.5, to find a nearby integer solution.

The second speed improvement is to fix many curves at once. The main difficulty is determining the *tightness* of many deltas at once. First, we gather all of the curves whose weighted deltas are equal to the maximum *M*. It may be that some of these deltas could be reduced further, but the LP has no incentive to do so since it wouldn’t affect *M*. To tell the difference, we set up and solve a modified LP: we constrain *M* not to increase from its current value, and constrain each of the large deltas to be at least the ceiling of its current value minus one. For the objective function, we minimize the sum of the large deltas. We solve the LP. Any weighted delta in the new solution that is less than *M* was not tight. We recursively remove such non-tight curves from consideration and resolve the LP, until only tight curves remain.

Another difficulty in fixing many curves at once is determining which way to round and fixing as many as possible while still remaining feasible. We round all curves’ intervals the same direction, either towards the next bigger integer or the next smaller integer. We choose rounding up or down depending on which we predict would result in the smaller maximum weighted delta amongst the tight curves, assuming the problem remains feasible. We sort the list of tight curves by descending weight (basically increasing goal). The reason is the highest weight deltas are most affected by improper rounding. We fix all of the tight curves, and resolve the LP. If the LP is infeasible, or the objective function increases more than we predicted, then we recursively try to only fix the first 3/4 of the tight curves.

The third speed improvement is to explicitly divide the problem into independent subproblems. Many problems are composed of independent subproblems. E.g. in a submapped volume there are three independent subproblems, one for each dimension of the volume. Solving the entire problem will fix the same sets of curves at the same values as does solving the subproblems independently; solving each problem independently is faster because a larger fraction of the curves of the relevant LP is fixed at each step.

The fourth speed improvement is to check for variables being completely determined by the fixed curves. E.g. in a mapped cube, once one of the four curves in an independent subproblem is fixed, all of the other three curves must be equal to that fixed value as well. In our implementation, whenever a curve is fixed, all of the LP rows containing that curve are checked to see if there is only one free variable left. If so, then that free variable is recursively fixed as well.

One advantage of our method is that at the end of the pseudo-relaxed solution process, we have a solution in which (almost) all of the curve interval variables are integer, and that satisfies all the constraints except the sum-even constraints. That is, the *k*s are not necessarily integer in the “sum of intervals = 2*k*” constraints. Intuitively, sum-even constraints are unlikely to cause large deltas, as an odd sum can be changed to even by simply increasing one of the summands by one. Similarly the non-integer curve intervals need only be changed by a small amount to become integer.

2.5 Step 2: Getting a Sum-Even Solution

We now construct a mixed integer linear program (MILP) whose solution will solve the interval assignment problem. We take the previous LP and throw away the constraints having to do with the deltas and *M*. This leaves the curve and sum-even variables, which are now all constrained to be integer. For speed we divide the problem into independent subproblems as in Section 2.4. In each subproblem, each curve and interval-sum variable is bounded above and below by its pseudo-relaxed solution value plus or minus a small value, depending on the four cases below. The objective function is the **sum** of the intervals-sum variables, plus a small constant times the sum of the weighted curve-interval variables. The weights V_x are chosen as they were for the deltas, so that large-interval rather than small-interval curves x are changed (increased). I.e.

$$\text{minimize} \quad V_a a + V_b b + \ldots V_e e + k_1 + k_2 + \ldots k_n$$

Four different sets of bounds on the integer variables are tried in sequence. If a feasible solution is found for one set of bounds, we don’t consider subsequent bounds. The more aggressive bounds are tried first. By “more aggressive” we mean the bounds more likely to result in small changes to the intervals and less likely to be feasible.

A given set of bounds may not be feasible, and the branch and bound (B&B) procedure may take exponential time to verify this. Also, even if a feasible solution is found, B&B may take exponential time to reach an optimal solution. To circumvent these problems, we explicitly limit the running time of the search for an initial feasible solution and, once a feasible solution is found, the time spent improving it.

In practice, if there is a feasible solution, it is found quickly. This is because the objective is a simple sum, and not the maximum, of weighted deltas. Usually we avoid generating an exponential number of sub-problems. Also, usually an optimal solution is not found, but one that is close to optimal.

Bounds 1. Curve intervals can increase by at most one, the interval-sum variables are between the ceiling of the pseudo-relaxed solution and ceiling + 1. This may be infeasible, especially in the presence of mapping faces with a composite side opposite a single side. This strategy is similar to the Ford solution[8], which iteratively increases a largest non-even interval curve on an odd-sum surface by one.

Bounds 2. Curve intervals can increase or decrease by one. Interval-sum variables are bounded between the floor and the floor plus one of the pseudo-relaxed solution.

Bounds 3. Curve intervals can double, but can't decrease. Interval-sum variables are bounded between the ceiling and twice the ceiling of the pseudo-relaxed solution. If there are no hard set curves, then there is always a solution: doubling all curve intervals still satisfies the (sub)mapping-face constraints, and since all curves now have an even number of intervals, the sum-even constraints are also satisfied.

Bounds 4. Curve intervals can double, and can decrease by at most one. Interval-sum variables are bounded between the floor and twice the ceiling of the pseudo-relaxed solution.

An alternative to B&B is to cast the problem as a matching problem on the dual graph. In the dual graph, odd interval-sum loop surfaces are nodes, and edges are (shortest) paths in the model that go from one surface to another surface via shared curves. A solution to the matching problem implies a solution to the interval assignment problem: increase the intervals by one on each of the curves dual to an edge path in the matching. Special care is needed for paths through two curves of the same side of a mapped surface, perhaps by alternating increasing and decreasing intervals on curves. This alternating idea is the reason behind allowing intervals to decrease in the above sets of bounds. Möhring et. al[9] casts the entire problem of interval assignment plus choosing the meshing primitive for each surface as a bidirectional flow problem.

3. Examples

3.1 A Simple, Detailed Example

Figure 2 shows an example of two adjoining surfaces, one with mesh scheme pave and the other with map, with various hard- and soft-set intervals. Note that D_i and d_i are the deltas referred to in the last section, the positive and

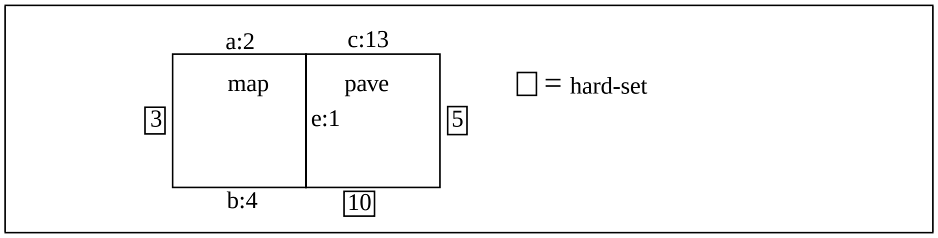

Figure 2. A simple interval assignment example.

negative difference between the goal intervals and the computed intervals x_i for curve i. Recall that M is the computed maximum of all of the deltas. Let k be the variable that computes the sum of intervals divided by two for the paved surface. The following illustrates the interval assignment algorithm run for this problem. The initial constraint matrix is the following:

Variables	a	b	c	e	k		
Bounds l,h	1	1	1	1	2		
Map, opp. sides	1	-1				=	0
Map, opp. sides				1		=	3
Pave, sum-even			-1	-1	2	=	15

This divides into two pseudo-relaxed problems, A and B. First A is solved; the first iteration solution follows:

Variables	a	b	D_a	d_a	D_b	d_b	M		
Bounds l,h	1	1							
Minimize							1		
Map, opp. sides	1	-1						=	0
a deltas	1		-1	1				=	2
a: $M \geq D, d$			0.50	1.20			-1	≤	0
b deltas		1			-1	1		=	4
b: $M \geq D, d$					0.25	0.40	-1	≤	0
Solution A.1	2.89	2.89	0.88			1.11	0.44		

The LP determines that both *a* and *b* are tight. Based on *a* and *b*, we predict rounding up would result in an *M* of 0.5, and rounding down an *M* of 0.8, so we choose to round intervals up: *a* is fixed at 3, and *b* is fixed at 3. Resolving we note that the problem is feasible, with *M* within the expected range, so *a* and *b* are both left fixed. Indeed, *M* is zero; recall the fixed curves have no contribution to *M*. So we move on to subproblem *B*. Its first iteration solution follows:

Variables	e	D_e	d_e	M		
Bounds l,h	1					
Minimize				1		
Map, opp. sides	1				=	3
e deltas	1	-1	1		=	1
e: $M \geq D, d$		1.0	4.0	-1	≤	0
Solution B.2	3	2		2		

Since this is the first iteration, and the problem is feasible, we check for dependent variables before looking for tight curves. We find that *e* is completely determined, so we fix it at 3. There are no remaining variables so we are done with subproblem B and move onto the sum-even constraints. In this case, there is just one subproblem; curves *a* and *b* are already integer and need not be considered again. The first set of bounds are tried:

Variables	c	e	k		
Variable type	Int	Int	Int		
Low bounds	13	3	16		
High bounds	14	4	17		
Minimize	0.01	0.03	1		
Map, opp. sides		1		=	3
Pave, sum-even	-1	-1	2	=	15
Solution 1	14	3	16		

We find a feasible solution, indeed an optimal solution, so there is no need to try the other, less aggressive bounds. Matching intervals was successful and the meshing process can proceed with meshing each surface independently.

3.2 An Illustration of Distributing Deltas

Figure 3 is a half-octagon that shows how minimizing the maximum weighted lexicographic vector of deltas distributes interval changes.

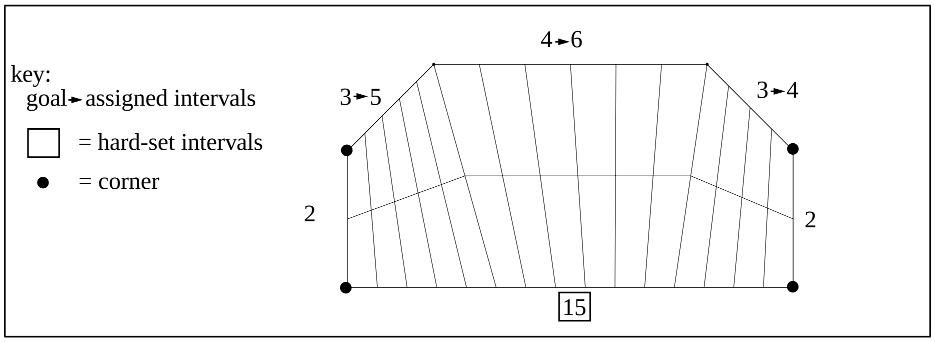

Figure 3. This surface was rectangle-primitive meshed using our automatic corner picking[2] and interval assignment algorithms.

3.3 Larger Examples

Running times are in cpu seconds for a Sun Ultra 2 workstation with one 300 MHz processor.

Submapping Heat Sink: See Figure 4 left. The heat sink is composed of 190 surfaces and 504 curves. The interval

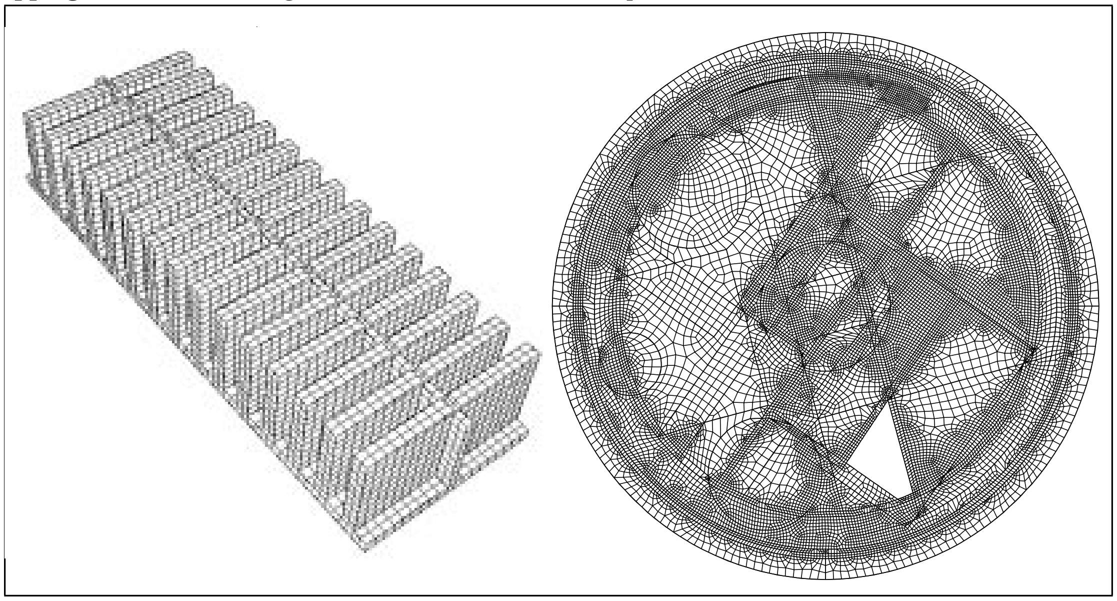

Figure 4. Left, submapped heat sink. Right, the mostly-paved CMDS problem with varying interval size.

size is about twice the length of the shortest curve. We mesh all surfaces with submapping. The running time is negligible, 0.7 seconds; the conference version of the algorithm[10], without the four speedups, took 96 seconds. The current version's solution is also of higher quality: the two long curves of the base change from 54 to 62 intervals, and sixty curves change from 9 to 10 intervals. All of the 1- and 2-interval curves remain at their user-set intervals.

There are three pseudo-relaxed subproblems, one for each of the principle axis. The first subproblem involves the short curves of the base. The first iteration finds 143 large-delta curves, of which 124 are tight, and all of which can be fixed. The remaining 60 curves are all the only remaining free curve in a constraint row, and are so fixed without a second iteration required. The second subproblem involves the long curves of the base. The first iteration finds 188 tight curves: all curves are tight! The two long curves of the base and many smaller curves opposite these long sides are also tight. Rounding all of these curves up is infeasible: the sum of the roundings for the small curves is much more than the roundings for the two long curves. Hence only the first 3/4 (141/188) small curves are fixed. But 39 other curves are completely determined by these fixed curves, and so are fixed as well. In the second iteration, the remaining 8 curves are tight and fixed. The third subprogram is similar. There are no sum-even constraints.

CMDS problem. This flat set of surfaces will be swept into a volume mesh; see Figure 4 right. This example consists of 209 surfaces and 529 curves. All the surfaces are paved, except one surface is triangle-primitive mapped. The solver finds an integer solution satisfying the most aggressive bounds, but times out after 11.5 seconds at a sub-optimal solution. Initially there are 80 surfaces with an odd loop-interval count. Solving modifies intervals on 78 curves.

ECA Deck. See Figure 5.This is a representative example of models that are hex-meshed at Sandia. The model consists of ten volumes with about 141 curves. The volumes are interlocked and will be swept (i.e. meshed with a 2.5 dimensional scheme). Our independent subproblem decomposition strategy is particularly effective for this type of model; because the paved source and target surfaces of the sweeping are interlaced with mapped side surfaces, there are many independent subproblems. Solving interval assignment takes 0.31 seconds, and overall meshing takes about 3 minutes. Interval assignment increases seven curves' intervals by one.

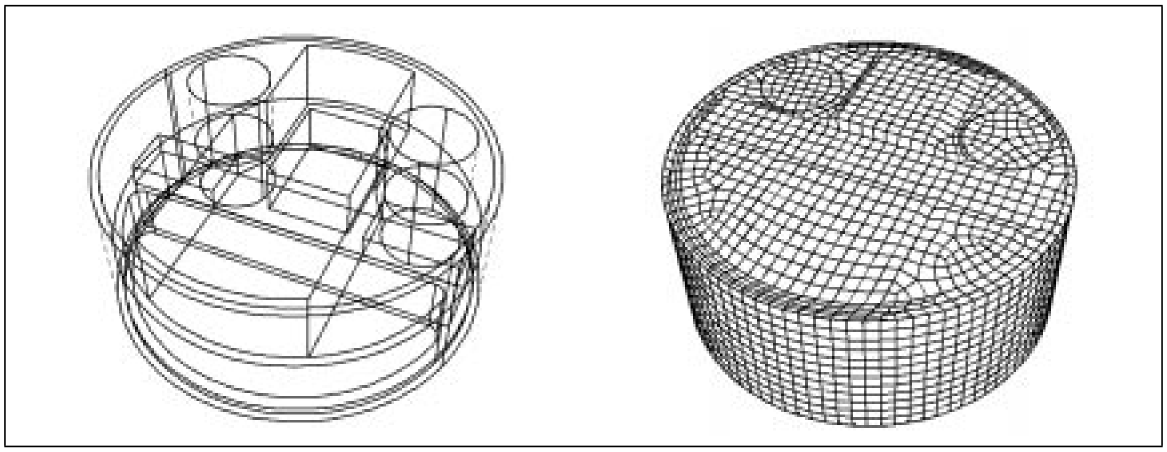
Figure 5. The ECA Deck: left, a wireframe view of the model interior; right, a hidden-line view of the meshed volumes.

Tradeoffs. Figure 6 illustrates some interval compromises, and points out a problem when assigned-intervals must vary wildly from the goals: for a mapped surface whose opposite sides' goals are G_1 and G_2, the compromise number of intervals x before rounding is $x = 2.3 / (1.0 / G_1 + 1.3 / G_2)$ which converges to 2.3 G_1 as G_2 approaches infinity. Actual intervals are within one of 2.3 G_1 for $G_2 > 3.0\ G_1^2 - 1.3\ G_1$. This problem is fundamental and arises from the weights being inversely proportional to the goals. A possible solution is mentioned in Section 5.

4. Running Times and Scalability

Experiments suggest that on realistic models composed of swept volumes with paved source and target surfaces and mapped sides, the running time of the algorithm is insignificant. On models with a large number of adjacent paved surfaces, the running time can be noticeable, e.g. a minute for a model with 200 surfaces, due to the **integer** linear program.

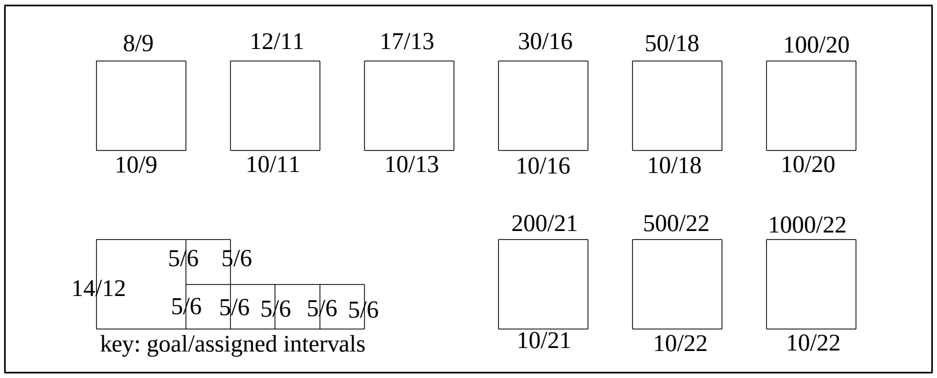

Figure 6. All squares are rectangle-primitive mapped. These examples illustrate some interval trade-offs arising from our choice of solution strategy and objective function. Given our strategy, for any set of constant weights, for large enough goal differences, we will see undesirable behavior like that in the lower right squares.

Good running time mainly depends on explicitly dividing the problem into independent subproblems, and fixing many curves at once. These features are highly complementary. Consider a single pseudo-relaxed subproblem. Since the variables of the problem are all dependent, there are typically only a few degrees of freedom. Consider a problem containing a large number of curves on a side of a mapped surface, whose opposite side is hard-set. In this case, a one-interval increase in one of these curves allows a one-interval decrease in another curve. Hence all of these curves are in tension with one another, and they are typically all tight at the same iteration. In general, the more dependent the curves are, the more they will be in tension with one another. Curves of independent subproblems will never be in tension with one another, so including them together in the same subproblem is not beneficial.

We explicitly limit the running time of the integer linear program, ILP, used to solve the sum-even constraints. First, the pseudo-relaxed solution and the tight bounds usually help the running time because we are near a feasible solution. In practice, if there is a feasible solution for a set of bounds, it is usually found quickly. The main problem is that the B&B process will take a large amount of time trying to improve the solution. Because of this, we proceed to the next set of bounds if a feasible solution is not found in O(n) time, and only allow B&B to improve the feasible solution for O(n) time. The less aggressive bounds are given more time as well.

In our problems the number of columns (variables) an the number of rows (constraints) are typically within an order of magnitude of each other, since surfaces usually contain a bounded number of curves. The time for a single call to the LP solver is about O(n^2), where n is the size of the (roughly square) LP.

In the worst case, there is a single subproblem, and only a constant number of curves gets fixed at each iteration. This might occur with a chain of mapped surfaces, where each shared side is composed of two curves with deviously chosen goals. Then we require O(n) calls to the LP solver, each of which takes O(n^2) time. Hence the pseudo-relaxed step may require O(n^3) time. The worst case running time of the B&B process is exponential.

Running time seems to be the main drawback of our method. Perhaps something other than branch and bound, such as matching techniques[9], might be used to satisfy the sum-even constraints, although it is not clear that this would improve running time in practice. Currently our algorithm is practical for typically structured models with thousands of curves. For very large models, we conclude that each subproblem must be further subdivided in some way, and the partial solutions stitched together.

5. Conclusions

We have described an implementation of a practical and robust way to solve the problem of globally assigning the number of mesh edges, *intervals*, to each curve in a complex of surfaces, so that each surface may be meshed according to pre-set meshing schemes, hard-set intervals, and corners. Unlike previous methods, our algorithm is good locally:

each curve has its assigned intervals close to its goal. We come close to minimizing the lexicographic vector of weighted interval differences. Our implementation is robust, although sometimes the problem is infeasible and no algorithm could work, and in some huge models our algorithm is unable to find a solution in a reasonable amount of time. The algorithm is practical for models of up to thousands of curves, given typical meshing schemes. In practice the algorithm is fast enough that it takes a very small faction of the total meshing time.

5.1 Future Directions

There are four broad categories of proposed improvements: finding and fixing global infeasibilities, see below; implementing volume scheme constraints, in progress; reducing the running time for huge problems, see Section 4; and improving the quality when intervals must vary wildly, see below.

Currently the interval assignment problem may be feasible for each surface individually, but globally infeasible due to hard-set intervals, or corner and scheme selection. Figure 7 illustrates how a locally optimal corner choice may lead to the global MILP being infeasible.

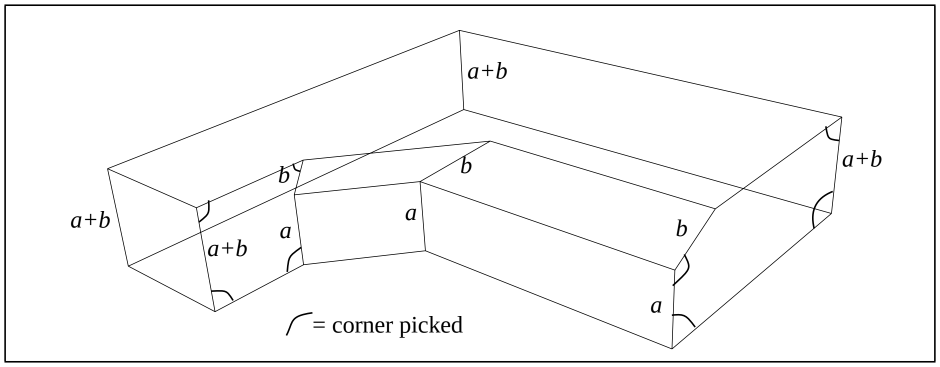

Figure 7. Local corner picking makes global interval assignment impossible on this real-world geological-fault geometry. We get a system of equations that reduces to $a+b = a$, whose only solution has $b=0$. But a curve must have at least one interval!

Identifying the cause of these infeasibilities, and fixing them automatically or at least reporting them to the user, is essential for large problems. There are a few obvious options to explore:

- find a minimal sub-problem that is infeasible, or a maximal sub-problem that is feasible. Many variations of this problem are in general NP-complete, but there is some hope since the constraints arise from geometric data. Also, it is not clear how to correlate a feasible/infeasible sub-problem to the schemes and corners that need to be changed.
- determine if infeasibility is due to hard-set intervals by relaxing the hard-set constraints.
- recognize global corner-picking infeasibilities by relaxing the constraint that intervals are non-zero.
- Combining the above two bullets, a solution of all zeros is feasible. Allowing a solution to have zeros, and then exploring where the zeros actually occur, might give insight into the problem.

Currently, quality is poor when interval goals vary wildly. In some cases large changes in intervals are necessary in order to satisfy mapping constraints. For **any** set of **constant** weights, if the changes are large enough then some curves will have intervals decreased by too much; see Figure 6. One possible fix is whenever a tight curve’s pseudo-relaxed solution is less than half its goal, instead of fixing it at its solution value, set an upper bound on it of half its goal, and reset its weight to a multiple greater than one of the weight based on half its goal.

References

[1] T. D. Blacker, W. J. Bohnhoff, T. L. Edwards, J. R. Hipp, R. R. Lober, S. A. Mitchell, G. D. Sjaardema, T. J. Tautges, T. J. Wilson, W. R. Oakes, S. Benzley, J. C. Clements, L. Lopez-Buriek, S. Parker, M. Whitely, D. White, and E. Trimble, CUBIT mesh generation environment volume 1: users manual. SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994.

[2] Scott A. Mitchell, Choosing corners of rectangles for mapped meshing. ACM Press, *Proceedings, 13th Annual Symposium on Computational Geometry*, June 4-6 1997, pp 87-93. URL:http://sass577.endo.sandia.gov/9225/Personnel/samitch/choosing-corners.html

[3] T. K. H. Tam and C. G. Armstrong, Finite element mesh control by integer programming, International Journal for Numerical Methods in Engineering, vol 36, 2581-2605, 1993.

[4] Linear Programming FAQ List. URL:http://www.mcs.anl.gov/home/otc/Guide/faq/linear-programming-faq.ht-ml

[5] David White, Automatic, quadrilateral and hexahedral meshing of pseudo-cartesian geometries using virtual subdivision. Published Masters thesis of Brigham Young University, August 1996.

[6] M. Whiteley, D. White, S. Benzley and T. Blacker, Two and three-quarter dimensional meshing facilitators, Engineering with Computers (1996) 12:144-154.

[7] LP Solve ftp site, ftp://ftp.es.ele.tue.nl/pub/lp_solve, 131.155.20.126. Author Michel Berkelaar. m.r.c.m.berke-laar@ele.tue.nl.

[8] Paul Kinney, Ford Motor Company, pkinney@ford.com or paul.kinney@sdrc.com. Personal communication.

[9] Rolf H. Möhring, Matthias Müller-Hannemann, and Karsten Weihe, Mesh refinement via bidirected flows: modeling, complexity, and computational results. Technische Universität Berlin, Department of Mathematics, Report No. 520 / 1996. URL: http://www.math.TU-Berlin.DE/~mhannema/projecteng.html

[10] Scott A. Mitchell, High fidelity interval assignment, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 33-44, October 1997.