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, which 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, although it runs more slowly than previous algorithms.

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 finally 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. Typically a curve is contained in two or more surfaces (volumes), 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 algorithms obtain an integer solution to a linear program {LP} for free.) Note that for a given choice of meshing algorithms, corners, and hard-set intervals, interval assignment may be infeasible.

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.

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.

The setup for our algorithm is the following: For each curve, the number of intervals assigned to it corresponds to a variable in the MILP. 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.

Our algorithm has two steps. In the first step we relax the variables, allowing them to take on non-integer values. We solve the LP with objective minimizing *M*. We find a curve that forced *M* to be as large as it is, and fix that curve's intervals to be the nearest integer value, and in effect remove it from the LP. We recursively solve the reduced LP until all curves are fixed or *M* is zero. That is, until all un-fixed curves have intervals equal to their goals. At the end of the first step we have an integer solution that satisfies all of the constraints except perhaps the sum-even constraints.

In the second step 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 sum 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 a 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 up to a thousand curves (or more if we're lucky and certain goals and meshing algorithms are chosen). To progress beyond that we conjecture that the problem should be divided up into subproblems, as is typically done with large LPs. Figure 4 shows some real-world models with about 500 curves each.

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 all 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].

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

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

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:

- Each curve's intervals are close to its specified goal.

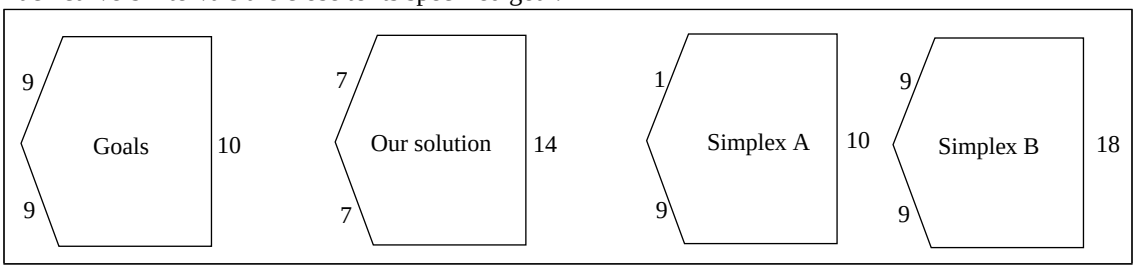

Figure 1. Given the goals on the left, we prefer our solution to those on the right. If a simplex method is used in conjunction with minimizing the weighted sum of interval changes, then, depending on the exact weighting, either Simplex A or Simplex B will be the solution.

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. Also, we chose a larger weight for D than for d, again reflecting the relative change idea. How much larger 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 that the linear programming problem already takes a large amount of time, and that quadratic programming is even slower, 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 the 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 perform badly.

2.4 Objective Function: 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 can't 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.

For speed, we don't solve the MILP at each step. Instead, we 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. In practice, it is critical for the running time that the old basis is used as a starting point for the new LP problem.

The *tightness* of a delta is verified by bounding it by a value less than its current value, resolving the relaxed LP, and testing if *M* has increased or the problem is infeasible. Curves with large deltas are tested first, and the first tight curve found is taken. (Also, if all of the positive deltas are less than 1, we don't test for tightness and instead take the curve with biggest delta.) This order is for speed: a better quality solution actually results from testing curves with small deltas first. An example that illustrates this is the heat sink in Figure 4 left.

One advantage of our method is that at the end of the pseudo-relaxed solution process, we have a solution in which 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.

2.5 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. Each curve and interval-sum variable is bounded above and below by its 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: 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

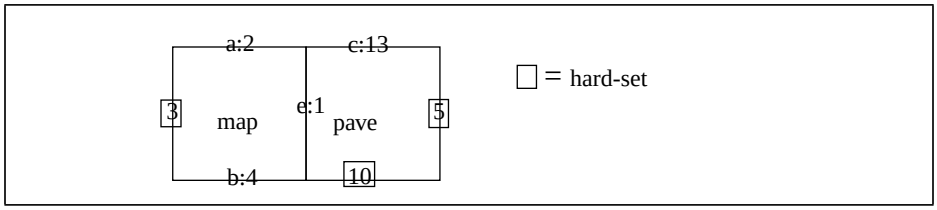

Figure 2. A small interval assignment example.

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 relaxed LP and solution is:

Variables	a	b	c	e	k	D_a	d_a	D_b	d_b	D_c	d_c	D_e	d_e	M	
Bounds l,h	1	1	1	1	2										
Minimize														1	

Map, opp. sides	1	-1														=	0
Map, opp. sides				1												=	3
Pave, sum-even			-1	-1	2											=	15
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
c deltas			1							-1	1					=	13
c: $M \geq D, d$										0.08	0.10				-1	≤	0
e deltas				1								-1	1			=	1
e: $M \geq D, d$												1.0	4.0	-1		≤	0
Solution 1	4	4	13	3	15.5	2						2		2			

Here D_e determines M, so e is fixed at 3 by setting its lower and upper bound to 3, and M is de-coupled from D_e and d_e by reversing the inequality in the "e: $M \geq D, d$" row. The reduced LP is resolved, leading to:

Variables	a	b	c	e	k	D_a	d_a	D_b	d_b	D_c	d_c	D_e	d_e	M	
Bounds l,h	1	1	1	3,3	2										
Solution 2	2.89	2.89	13	3	15.5	0.89			1.11			2		0.44	

Note $M = W_a D_a = w_b d_b$. In this case both a and b are tight. However, in general, when two or more weighted deltas are equal to M, one or more of them might not be tight, and it would be a mistake to fix a curve that wasn't tight. So the LP is temporarily modified and resolved to verify that b is tight. Then b is fixed at 3 = round(2.89), M is de-coupled from b's deltas, and we proceed to the next iteration:

Variables	a	b	c	e	k	D_a	d_a	D_b	d_b	D_c	d_c	D_e	d_e	M	
Bounds l,h	1	3,3	1	3,3	2										
Solution 3	3	3	13	3	15.5	1		0.11	1.11			2		2	

Note that both D_b and d_b are now positive, a by-product of reversing the inequality in the "e: $M \geq D, d$" row, and the fact that the objective function doesn't strictly decrease because curves are fixed at their rounded values. In practice, however, reversing the inequality leads to a better running time than eliminating the constraint altogether. Now a uniquely determines M, so a is fixed at 3 by setting its lower and upper bounds, its deltas are de-coupled from M, and we proceed to the next iteration:

Variables	a	b	c	e	k	D_a	d_a	D_b	d_b	D_c	d_c	D_e	d_e	M	
Bounds l,h	3,3	3,3	1	3,3	2										
Solution 4	3	3	13	3	15.5	1			1			2		0	

Since $M = 0$, we stop. Note that c was not fixed, and in practice often intervals are exactly equal to their goals and the algorithm stops with fewer iterations than variables. We now proceed to the integer phase, where the "sum-even" constraints are satisfied. The delta constraints and variables are removed and the first set of bounds are tried:

Variables	a	b	c	e	k		
Variable type	Int	Int	Int	Int	Int		
Low bounds	3	3	13	3	16		
High bounds	4	4	14	4	17		
Minimize	0.03	0.03	0.01	0.03	1		
Map, opp. sides	1	-1				=	0
Map, opp. sides				1		=	3
Pave, sum-even			-1	-1	2	=	15
Solution A	3	3	14	3	16		

A feasible solution was found (in fact 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.

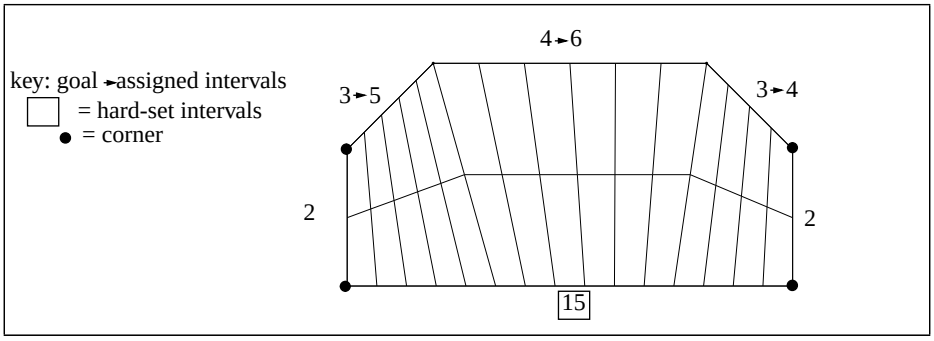

Figure 3. This surface was rectangle-primitive meshed using our automatic corner picking[2] and interval assignment algorithms.

3.3 Larger Examples

Submapping Heat Sink: See Figure 4 left. The heat sink is composed of 190 surfaces and 504 curves. The interval size is about twice the length of the shortest curve. In the first test case, we mesh all surfaces with submapping. The pseudo-relaxed problem takes 148 iterations, fixing about 1/3 of the 504 curves. The running times for this and subsequent examples are for an HP 735/125. The first iteration takes 3.13 seconds. The time for subsequent iterations decreases, for a running time of 92.64 seconds, or 0.63 iterations/second. Verifying that curves are tight takes 0.53 seconds total, and in most problems is not a significant contribution to the running time. Getting the final integer solution takes only 0.92 seconds: Since there are no sum-even variables, the solver just has to verify that the pseudo-relaxed solution is integer, and feasible, to find an optimal solution for the most aggressive bounds. Interval assignment takes 96 seconds total.

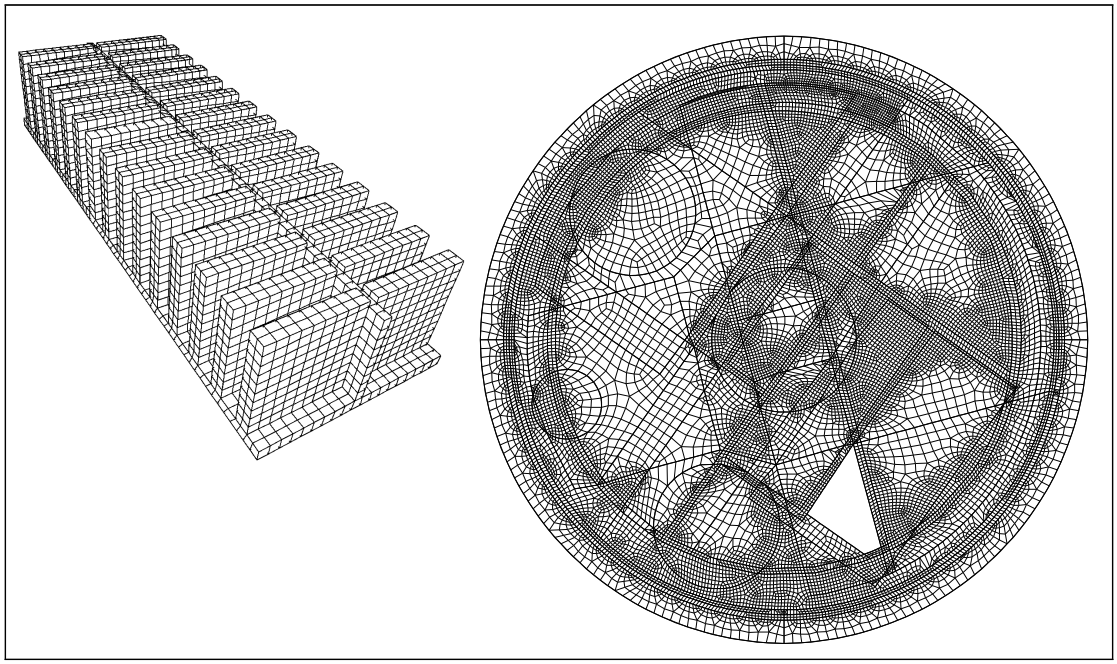

Figure 4. Left, submapped heat sink. Right, the mostly-paved CMDS problem with varying interval size.

Interval assignment changes 2 curves from 54 to 60 intervals, 60 curves from 9 to 10 intervals, and 12 curves from 2 to 1 intervals. Each initially-54 interval curve is opposite a composite side of 31 initially-2 interval curves. The sides don't match due to rounding: the curve length (3) divided by the user-specified size (2) gets rounded to an integer interval value (3/2 -> 2). Interval assignment sets the long curve to 60 intervals and two of the opposite curves to 1 interval. The alternate strategy of testing the smallest delta would have lead to all 2-interval curves remaining at 2-intervals, and the 54-interval curves changing to 62.

Paving Heat Sink: In this example surfaces are meshed with paving. Solving the single iteration of the pseudo-relaxed problem takes 1.91 seconds. Finding a feasible solution to the most aggressive integer bounds takes 1.7 seconds and we stop at a sub-optimal solution after 7.35 more seconds. 60 surfaces initially had odd intervals-sums. Solving changes 15 curves from 1 to 2 intervals, and 44 curves from 9 to 10 intervals.

Mixed Head Sink: In this example, every other surface is paved, selected at random, and the remaining surfaces are submapped. The pseudo-relaxed problem takes 113 iterations for 41 seconds. Finding the optimal integer solution takes 8.7 seconds. Solving changes 1 curve from 54 to 65, 65 curves from 9 to 10, 9 curves from 2 to 3, 2 curves from 2 to 1, and 1 curve from 1 to 2 intervals.

CMDS problem. 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 pseudo-relaxed solution takes 2.5 seconds for three iterations. 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.

Figure 5, and Figure 6 illustrate some interval compromises, and points out a problem when assigned-intervals must vary wildly from the goals.

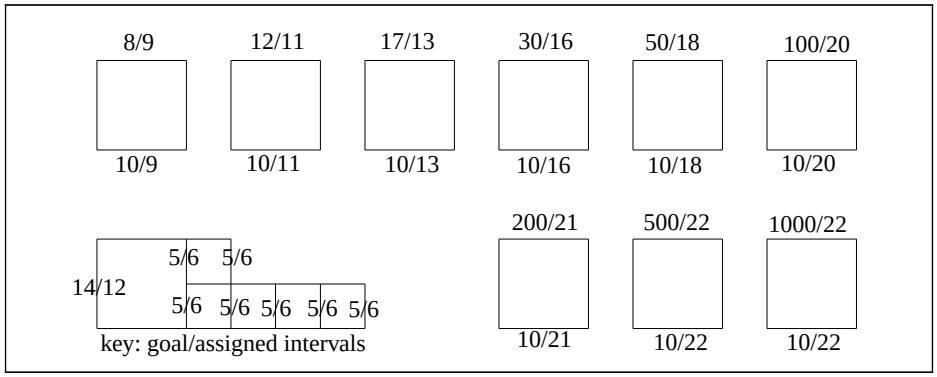

Figure 5. All squares are rectangle-primitive mapped. These examples illustrate some interval trade-offs arising from our choice of solution strategy and objective function. Given our strategy, for any set of constant weights, for large enough goal differences, we will see behavior like that in the lower right squares.

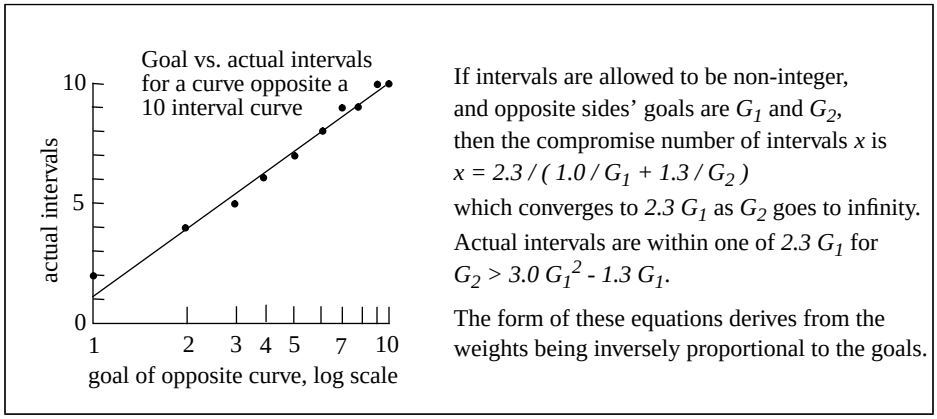

Figure 6. Goal versus actual intervals.

4. Running Times and Scalability

Experiments suggest that the running time of our algorithm for a collection of surfaces is $O(n^3+nm^2)$, where n is the number of non-paved surfaces and m the number of paved surfaces. We assume surfaces contain a constant number of curves.

This running time makes sense intuitively: the time for a single call to the LP solver is about $O((n+m)^2)$, and we make about $O(n)$ calls to solve the pseudo-relaxed problem. For each successive call, n is one less, but this only affects the constants not the asymptotic behavior of the algorithm. For problems with all surfaces paved, the running time for solving the first iteration of the trivial pseudo-relaxed problem is about $O(m^2)$, and only one iteration is needed. The time for getting a feasible solution to the integer problem (for the most aggressive bounds) is also roughly $O(m^2)$. LP Solve is allowed to try to improve the solution for $O(n+m)$ time.

Running time seems to be the most severe drawback of our method. There are a number of improvements possible. First, after each pseudo-relaxed iteration, if a curve has a large change, the goals could be explicitly changed on all curves sharing a composite side with the curve. This might be based on a "min sum" rather than a "min max" strategy, where goals were changed according to a geometric progression to simulate a "min delta^2" strategy. Second, currently many iterations are spent where a curve gets changed by only one interval or less, so the algorithm might switch to a "min sum" strategy sooner. These two improvements might reduce the running time for mapped problems to $O(n^2 \log r)$, where r is the maximum ratio of a curve's assigned intervals to its goal, and its reciprocal. Third, the sum-

even constraints shouldn't be added to the LP until after the pseudo-relaxed problem is solved. Fourth, something other than branch and bound, such as Lagrangian relaxation[10][11] or matching techniques[9], might be used to satisfy the sum-even constraints, although this might not improve the running time.

Currently our algorithm is practical for problems with a thousand curves. With these improvements at best it would be practical for problems with ten thousand curves. For very large models, we conclude that the problem must be broken down in some way.

5. Conclusions

We have described an implementation of a practical and robust way to solve the problem of globally assigning intervals to 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 large models our algorithm is unable to find a solution in a reasonable amount of time. The algorithm is practical for models of up to a thousand curves. In practice the running time is $O(n^3)$, which is slower than some previous $O(n^2)$ methods.

5.1 Future Directions

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

Currently the interval assignment problem may be locally feasible but globally infeasible due to hard-set intervals, and corner and scheme selection. Figure 7 illustrates how choosing corners locally may lead to the global MILP being infeasible.

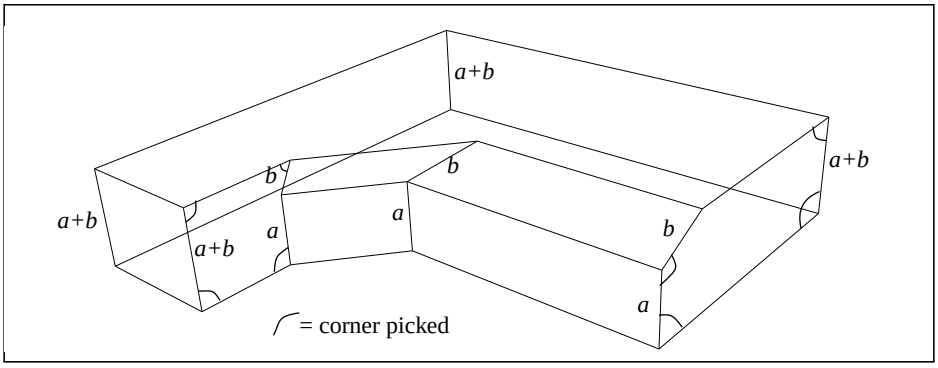

Figure 7. Local corner picking makes global interval assignment impossible on this real-world geological-fault geometry. We get a system of equations that reduces to $a+b = a$, whose only solution has $b=0$, but a curve must have at least one interval!

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. The constraints arise from geometric data. However, many variations are in general NP-complete, and its not clear how to correlate a feasible/infeasible sub-problem to the scheme or corner set that needs to be changed.
- It might be easy to determine if infeasibility is due to hard-set intervals, by relaxing the hard-set constraints.
- Some global corner-picking infeasibilities might also be recognized by relaxing the constraint that intervals are non-zero.

- Combining the above two bullets, a solution of all zeros is then 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 5 and 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. This might also become a strategy for improving the running time.

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, in proc. *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.html

[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 Michael Berkelaar. m.r.c.m.berkelaar@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] Marshall L. Fisher. The Lagrangian relaxation method for solving integer programming problems. Management Science **27**:2, January 1981, pp. 1-18.

[11] Marshall L. Fisher. An applications oriented guide to Lagrangian relaxation. Interfaces **15**:2, March/April 1985, pp. 10-21.