A TECHNIQUE FOR OPTIMIZING GRID BLOCKS

John F. Dannenhoffer, III*

United Technologies Research Center
East Hartford, CT 06108

ABSTRACT

A new technique for automatically combining grid blocks of a given block-structured grid into logically-rectangular clusters which are "optimal" is presented. This technique uses the simulated annealing optimization method to reorganize the blocks into an optimum configuration, that is, one which minimizes a user-defined objective function such as the number of clusters or the differential in the sizes of all the clusters. The clusters which result from applying the technique to two different two-dimensional configurations are presented for a variety of objective function definitions. In all cases, the automatically-generated clusters are significantly better than the original clusters. While this new technique can be applied to block-structured grids generated from any source, it is particularly useful for operating on block-structured grids containing many blocks, such as those produced by the emerging automatic block-structured grid generators.

INTRODUCTION

Traditionally, the block-structured grid generation process has been called "complete" when a set of blocks and grid points which provide adequate resolution for a given configuration has been defined. While the location of the grid points is usually carefully controlled to provide the required spacings and grid quality, the location of the block boundaries is chosen arbitrarily to aid in the grid generation process.

Unfortunately, the location of the block boundaries can have a strong influence on the overall performance of the flow solver, especially when employing one of today's advanced computer architectures. For example, for a vector computer one typically wants as few blocks as possible, while simultaneously keeping the blocks as "long" as possible in order to maximize use of the vector pipelines. Alternatively, for a parallel computer (or a network of workstations) one typically wants the blocks to be as load-balanced as possible in order to minimize idle time. To compound the problem, the computer architecture which will be employed to obtain the flow solution is not always be known at grid generation time. Therefore, it is desirable to have a procedure which can recombine blocks (that is, create clusters) of any block-structured grid, regardless of the software used to generate it.

In addition, a new class of block-structured grid generators is emerging which "automatically" generates block-structured grids for a variety of configurations[1, 2]. A general property of the grids which these procedures generate is that they produce a very large number of grid blocks, sometimes running into the

*Senior Research Engineer

tens of thousands. This new clustering procedure could be used in such cases to significantly reduce the number of clusters (blocks) on which the flow solver would have to operate.

The objective of this work is to develop a general technique for automatically combining blocks of a given block-structured grid into logically-rectangular "clusters" so as to minimize a user-defined objective function.

TECHNICAL APPROACH

The Clustering Problem

Consider the block-structured grid configuration shown in Figure 1. It consists of seven grid blocks (the seven squares labeled A through G) which are arranged so as to fill an L-shaped domain; also shown are eight "removable" edges (which are labeled 1 through 8).

Simply stated, the objective in the clustering problem is to find the set of edges which, when removed, yields both a "valid" configuration and which minimizes some "score". For now, assume that the "score" is the number of clusters. A "valid" clustering is one which solely consists of (logical) rectangles; L-shaped clusters are invalid.

One could reduce the score of the clustering in Figure 1 by removing any edge; for example, when edge 7 is removed, the clusters shown in Figure 2 result, with the score now reduced to the better value of 6. To further reduce the score, either edge 1, 2, 3, 4, or 6 can be removed to yield a valid clustering and a reduced score of 5. But removal of either edge 5 or 8 yields a cluster (for example, cluster EFG in Figure 3) which is not logically-rectangular and thus invalid. This process can be continued, yielding the optimal score is 2 which is produced either by retaining only edge 2 or by retaining only edges 3 and 4.

Optimization Procedure

The optimization problem considered here is one which is characterized as a combinatorial minimization problem. That is to say, the design space cannot be characterized as an N-dimensional surface defined over N smoothly-varying design parameters. Rather it is a design space which is described by a set possible configurations. Other problems of this combinatorial nature include VLSI circuit layout[3] and the infamous traveling salesman problem[4].

There are two major difficulties associated with combinatorial optimization problems which do not arise in more traditional optimization problems. First, the design space can become extremely large because the configurations generally are describable by a sequence of events (the number of which, when taken together, grows factorially). Second, because the design variables can only take on discrete values (such as "on" and "off"), any procedure which exploits the concept of gradient is not relevant.

One technique which has proven very successful at solving the combinatorial optimization problem is the simulated annealing algorithm, which was first proposed by Metropolis[5] and which is described extensively by Aarts and Korst[6].

Stated very briefly, the simulated annealing algorithm is: starting from a known valid configuration, randomly select a change of state; any change which results in lower objective function value or which results in an objective function value which is not "too much worse" than that of the previous configuration is accepted. This process is repeated with the definition of "too much worse" tightening as it proceeds, until no proposed changes to the configuration yield any reduction in the objective function.

This procedure gets its name from an analogy with the thermodynamic process of annealing, or freezing, of liquids into solids. In the beginning stages of the solidification process, molecules move freely in an attempt to form crystalline structures which have low internal energies (the objective function). For annealing, the temperature is slowly reduced and the mobility of the molecules is gradually restricted, resulting in minimum energy crystals. Alternatively, for quenching, the temperature is quickly reduced, resulting in a crystalline structure which has significantly higher energy.

These same concepts can be applied to optimization. The initial steps of annealing (when the control parameter is high) is similar to a random-walk procedure, which is known to be an effective strategy for getting into the neighborhood of a global optimum. The final steps of annealing (when the control parameter is low) is similar to a hill-climbing procedure, which is known to be effective in the vicinity of an optimum. The annealing schedule provides an orderly mechanism of transitioning from one to the other. It should be noted that quenching is analogous to hill-climbing alone, which is notorious for getting stuck in local extrema (as are all greedy algorithms).

In terms of pseudo-code, the algorithm can be stated as follows:

* initialize the configuration and compute its objective function $\mathcal{O}$

* set an initial value for the control parameter T

* do { (outer loop of generations)

  * do { (inner loop of attempts)

    * propose a random change to the configuration

    * compute the change in objective function $\Delta\mathcal{O}$ of the proposed change

    * if ($\Delta\mathcal{O} < 0$ or random$(0 \rightarrow 1) < e^{-\Delta\mathcal{O}/T}$) then

      - take the step and adjust $\mathcal{O}$

    } until (stop criterion is met)

  * decrease the control parameter T

  } until (there were no successful proposals at the previous T)

In order to use this algorithm for optimizing block clusters, one needs to provide the following:

Configuration (a method of describing the current "state" of a configuration). For the current application, the current configuration can be described as a table that says which edges currently exist (or conversely, which edges have been removed);

Rearrangements (a method for proposing and carrying out changes to the configuration). Here the procedure is to pick an edge at random, and if it exists, remove it; alternatively if it has already been removed, then re-insert it. In this way, the effects of any previous rearrangements can be undone (which, by the way, is generally an important property when using simulated annealing);

Objective function computation (an efficient procedure for computing the change in objective function given any proposed change to the configuration). One of the nice features of using the simulated annealer is that, except for the initial configuration, one only has to calculate the change in objective function $\Delta\mathcal{O}$ that results from a proposed configuration change. This is indeed fortunate, because for the clustering problem, the change in objective function can be computed in a small, fixed amount of work (that is, independent of the problem size), whereas the work to compute the actual objective function scales with N (the number of removable edges in the configurations); and

Annealing schedule (a recipe for controlling the decrease of the control parameter T). This is the least rigorous part of the simulated annealing algorithm. In general, a suitable annealing schedule can only be found by experimenting.

Objective Function Formulation

In addition to the selection of suitable "design variables" (discussed above), the application of the simulated annealing algorithm to the clustering problem requires the definition of the "objective function" and "constraints".

First consider the objective function, $\mathcal{O}$. The most obvious choices include:

Minimize number of clusters This objective function is appropriate when one wants to minimize the number of concurrent processors which are needed to perform some calculation. A suitable form might be

$$\mathcal{O} = N_{\text{clus}}$$

where N_{clus} is the number of clusters.

Minimize size of largest cluster When running a calculation on a parallel processor or on a group of workstations, one typically wants to distribute the work as evenly as possible across the processors. A suitable objective function might be

$$\mathcal{O} = N_{\text{size}}$$

where N_{size} is the size of (number of grid points in) the largest cluster. (Note that by itself, this objective function might arrive at the solution that each original cluster should remain unchanged.)

Hybrid Clearly a hybrid objective function could be formed which combines the above, with suitable coefficients, as in

$$\mathcal{O} = \lambda_{\text{clus}} \min\left(N_{\text{clus}}, L_{\text{clus}}\right) + \lambda_{\text{size}} \min\left(N_{\text{size}}, L_{\text{size}}\right)$$

where λ_{clus} and λ_{size} are weighting coefficients and L_{clus} and L_{size} are user-specified limits. These limits are particularly useful for tuning the objective function in a variety of ways. For example, one could minimize the number of clusters with the restriction that no cluster to be larger than some specified size (as might be imposed by a memory restriction in some processor). Similarly, given the number of clusters, one could minimize the size of the largest cluster (in essence, this amounts to load-balancing).

Notice that in the hybrid formulation of the objective function, the "weak" constraints imposed by L_{size} and L_{clus} are actually written as part of the objective function. There is in addition a "strong" constraint on this problem, namely the requirement that all clusters be logically rectangular. This can be accounted for by adding another "penalty" term to the objective function, or

$$\mathcal{O} = \lambda_{\text{clus}} \min\left(N_{\text{clus}}, L_{\text{clus}}\right) + \lambda_{\text{size}} \min\left(N_{\text{size}}, L_{\text{size}}\right) + \lambda_{\text{brak}} N_{\text{brak}}$$

where N_{brak} is the number of edges which would have to be re-inserted to yield all logically-rectangular clusters. In general, $\lambda_{\text{brak}} > \lambda_{\text{clus}}$ to ensure that the final clusters are all well shaped.

COMPUTED RESULTS

As demonstration cases, results were obtained on several two-dimensional configurations; two of them are shown here.

The first test case is for an internal-flow passage containing eight circular posts (hence the test case is called "posts"). The original grid for this case, which contains 5543 nodes in 89 clusters (each block was initially assigned to its own cluster), is shown in Figure 4a.

For the first experiment on the "posts" test case, the simulated annealer was applied to minimize the number of clusters (the user-supplied constants are given in Table I). Intermediate results, that is the clustering at the end of each setting of the control parameter T, are shown in Figures 4b–e. Notice that most of these intermediate results contain many badly-shaped clusters (that is, they are not logically rectangular). The final clustering, which is shown in Figure 4f, contains 19 clusters, where the largest cluster contains 969 nodes. Table I summarizes these intermediate and final results.

For this case, as well as all others shown here, the control parameter was initially set to $T = 5.0$ and was reduced at the end of each generation (the outer loop) by a factor of $T_{\text{new}}/T_{\text{old}} = 0.75$; these values, which were determined through numerical experimentation, have been found to be sufficient for all test cases executed to date. Also, the stopping criterion at the end of the attempts (inner) loop was taken to be the first of: $10N$ successful attempts or $100N$ total attempts, whichever comes first (where N is the number of removable edges). Again, these values were determined by numerical experiments and were found to work well for all cases which have been tried to date.

In order to better understand how the simulated annealer works, a convergence history for this case was plotted in Figure 5. The abscissa represents the successful attempt number while the ordinate shows the current value of the objective function $\mathcal{O}$. The circles indicate those times when the control parameter T is decreased. There are a number of interesting features evident in the Figure:

- the objective function $\mathcal{O}$ does not strictly decrease, but rather tends toward an optimum;
- there is a "noise band" corresponding to the concept of "not too much worse". As expected, the width of the band decreases as the control parameter decreases; and
- the number of successes between control parameter reductions decreases, corresponding to the fact that as T gets smaller, it is more difficult to find a success.

It should be noted that the simulated annealing algorithm is stochastic because of the random number generator used both to select a proposed change and in the acceptance check. In theory then, the results which it produces cannot necessarily be repeated. In order to determine the sensitivity of the present results to the random number seed, a few of the test cases have been re-executed many times. In all cases, the "optimum" solutions were very close to each other and were very far from the initial solution.

One further point should be made about the algorithm. The work required to reach "convergence" is bounded by $N \times M$, where N is the number of design variables (the number of removable edges) and M is the number of times which the control parameter T needs to be decreased. Although in the worst case $M \to \infty$, test cases of varying sizes seem to indicate the M is approximately constant, with a value of about $M \approx 10$. In fact, even though in the current implementation a limit of $M = 20$ has been hard-coded (to protect against infinite looping), it has never been exercised.

For the second experiment on the "posts" test case, the number of clusters were again minimized, but with an upper limit $L_{\text{size}} = 500$ on the size of any one cluster. The clustering which results for this case, which is shown in Figure 6a and is summarized in Table I, is very different from that discussed above.

A third experiment on the "posts" test case was conducted, wherein the size of the largest cluster was to be minimized with the constraint the the number of clusters be bounded by $L_{\text{clus}} = 25$. The results of

this test are shown in Figure 6b and again in Table I. Again the clusters which are formed are different from the above two cases, and in general not intuitively obvious.

A second test case, corresponding to a gas-turbine combustor cross-section (and hence call "combustor") is also shown here. The Figures 7 and 8 and Table II summarize its results. The results, which are very similar to those obtained for the "posts" test case, serve to demonstrate the broad applicability of the new clustering technique. As in the "posts" test case, all cases executed in less than two minutes of CPU time.

CONCLUSIONS

A new procedure for optimizing "clusters" of grid blocks has been described. The procedure, which uses the simulated annealing optimization algorithm, has been executed on many configurations; it has been demonstrated here on two different two-dimensional configurations. In all cases, the new algorithm has yielded clusterings which are "good" in the sense that they have either significantly reduced the total number of clusters (with and without a limit on the maximum cluster size) or have balanced the cluster sizes (with a given number of clusters). Even though the simulated annealer does not guarantee that an actual global extremum has been achieved, the results of the automatic algorithm were at least as good as the clusterings which were manually generated. All cases executed required less than two CPU minutes on an Silicon Graphics Indigo2-R4400 for problem sizes of about 100 removable edges; numerical experiments indicates that the time should scale linearly with problem size. While this procedure is applicable to block-structured grids generated by any software, it is particularly well-suited to those generated by the emerging "automatic" block-structured grid generators.

ACKNOWLEDGEMENTS

This work was funded under the UTC Corporate Sponsored Research Program. The author would like to thank David Sirag for his helpful insights and comments about the simulated annealing algorithm as well as many of his other colleagues who reviewed this manuscript.

REFERENCES

[1] Dannenhoffer, JF, "Automatic Blocking for Complex 3-D Configurations," NASA CP-3291, May 1995.

[2] Shaw, JA, and Weatherill, NP, "Automatic Topology Generation for Multiblock Grids," *Applied Math. and Comp.*, vol 53, pp 355–388, 1992.

[3] Vecchi, MP, and Kirkpatrick, S, "Global wiring by simulated annealing," *IEEE Trans. on Computer-Aided Design*, vol 2, pp 215-222, 1983.

[4] Aarts, EHL, Korst. JHM, and VanLaarhoven, PJM, "A quantitative analysis of the simulated annealing algorithm: a case study for the traveling salesman problem," *J. of Statistical Physics*, vol 50, pp 189-206, 1988.

[5] Metropolis, N, Rosenbluth, A, Teller, A, and Teller, E, *J. Chem Phys.*, vol 21, p. 1087, 1953.

[6] Aarts, E and Korst, J, *Simulated Annealing and Boltzmann Machines*, Wiley & Sons, 1989.

Figure	λ_{clus}	L_{clus}	λ_{brak}	λ_{size}	L_{size}	N_{clus}	N_{brak}	N_{size}
4a	1.00	1	2.00	0.001	0	89	0	425
4b						23	28	1666
4c						22	26	1543
4d						28	16	1027
4e						31	13	850
⋮								
4f						19	0	969
6a	1.00	1	2.00	0.100	500	23	0	455
6b	1.00	25	2.00	0.001	0	25	0	441

Table I: Summary of results for test case "posts".

Figure	λ_{clus}	L_{clus}	λ_{brak}	λ_{size}	L_{size}	N_{clus}	N_{brak}	N_{size}
7a	1.00	1	2.00	0.001	0	80	0	273
7b						30	19	633
7c						31	17	978
7d						21	14	1356
7e						36	4	533
⋮								
7f						16	0	828
8a	1.00	1	2.00	0.100	500	22	0	468
8b	1.00	25	2.00	0.001	0	25	0	468

Table II: Summary of results for test case "combustor".

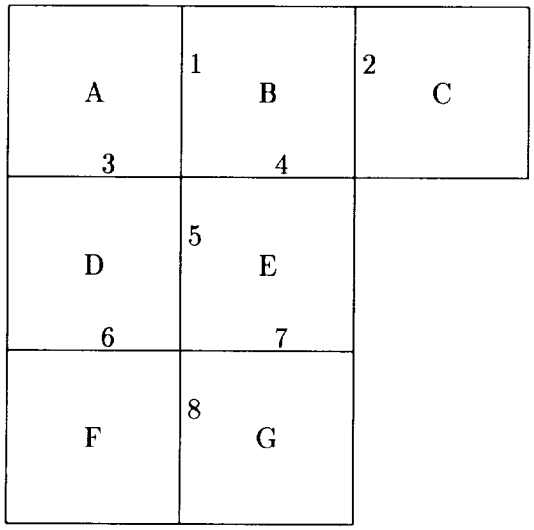


Figure 1: Original configuration. Score=7.

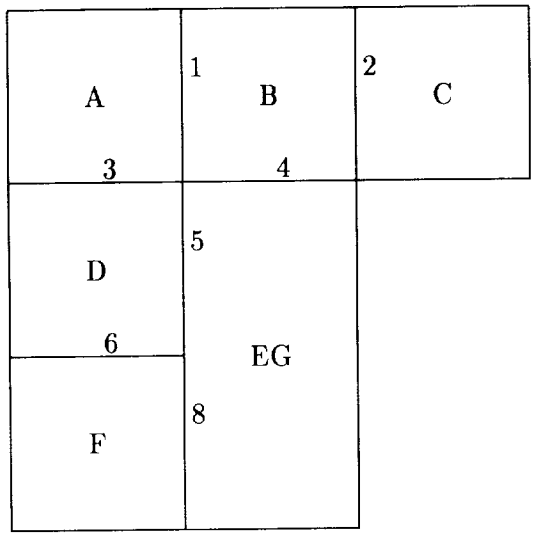


Figure 2: (Valid) configuration with edge 7 removed. Score=6.

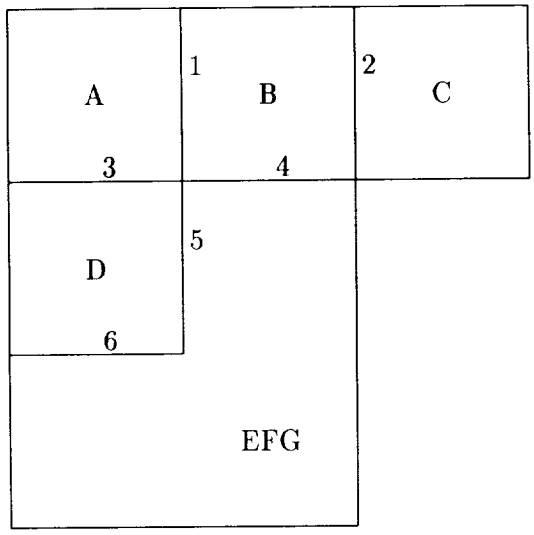


Figure 3: (Invalid) configuration with edges 7 and 8 removed.

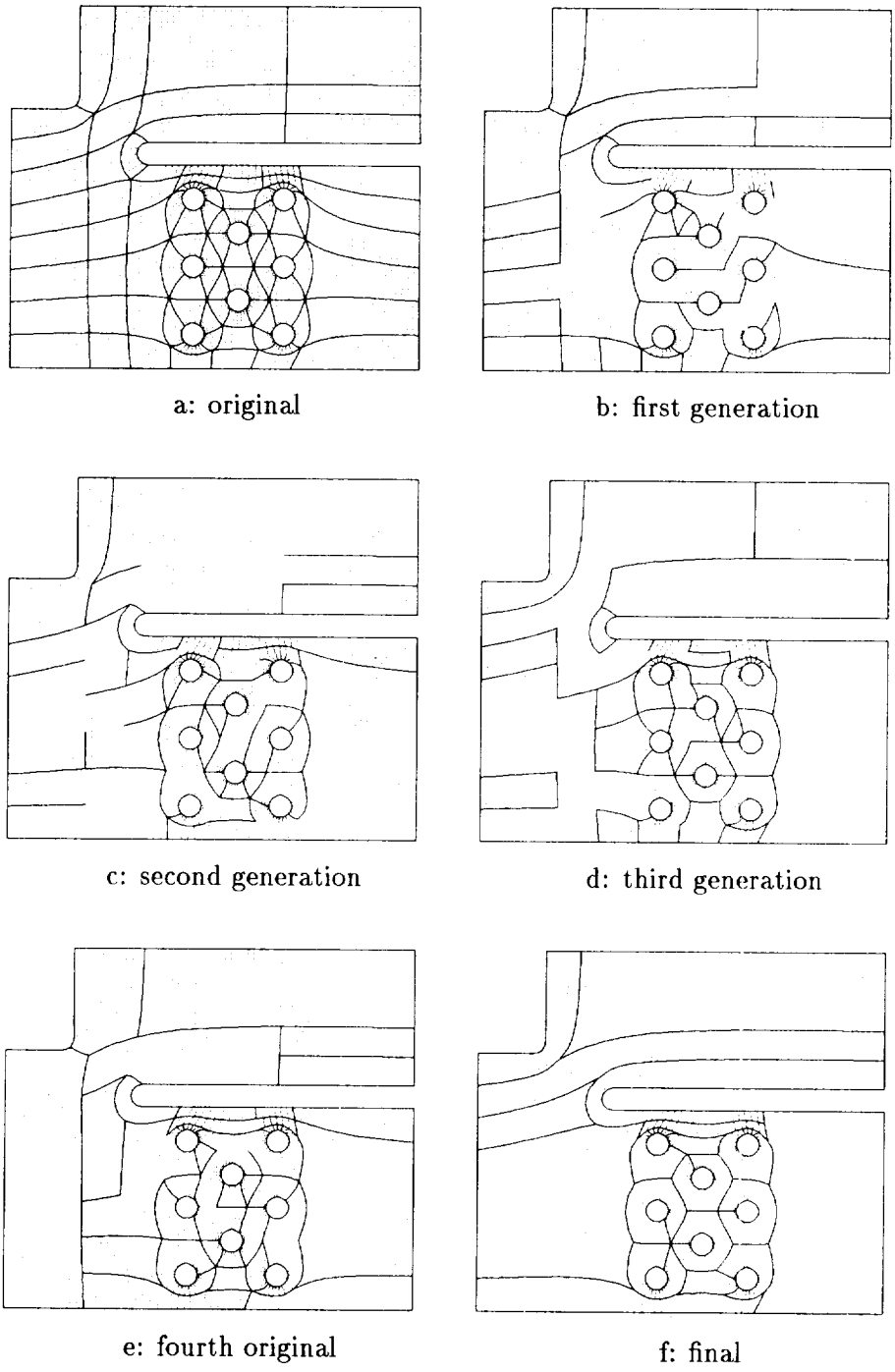


Figure 4: Minimization of the number of clusters for test case "posts".

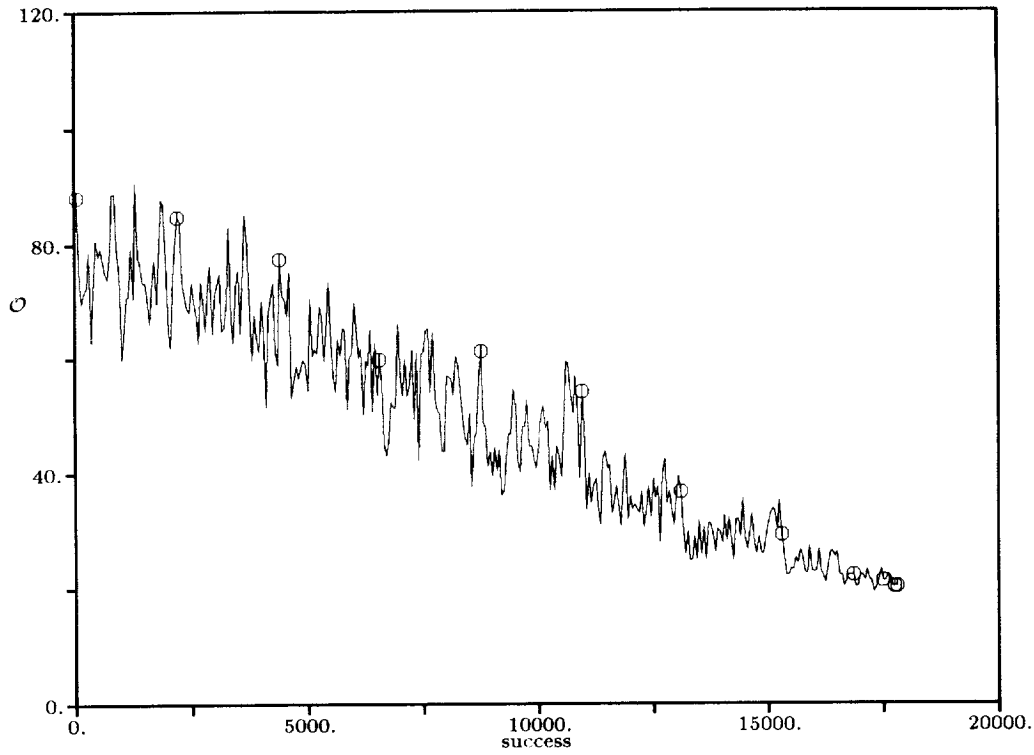
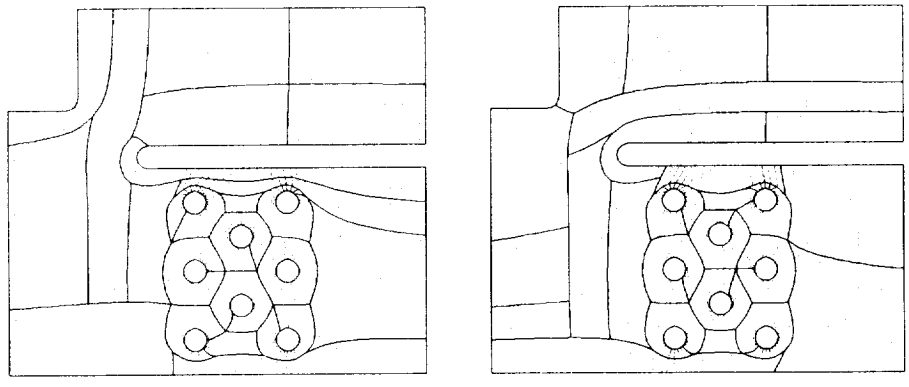
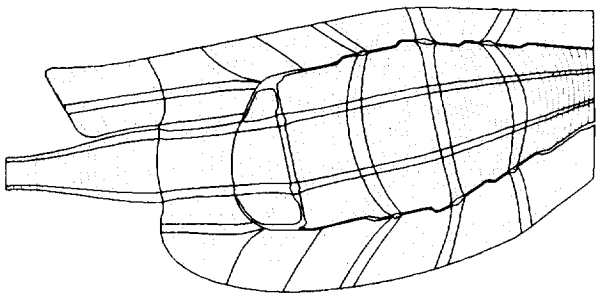


Figure 5: Convergence history for test case "posts". Only every 50th successful change is plotted. The circles show the times when the control parameter T was decreased.

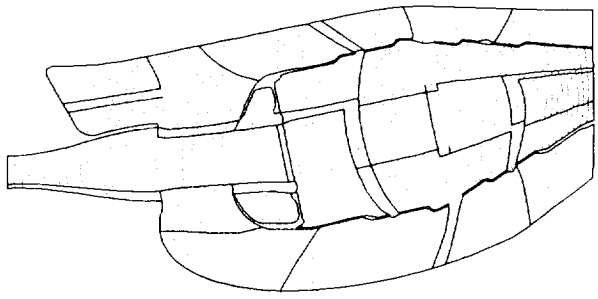


a: $L_{\text{size}} = 500$ b: $L_{\text{clus}} = 25$

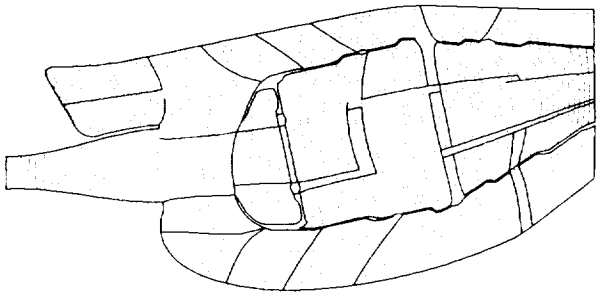
Figure 6: Final clusters for alternative optimizations for test case "posts".



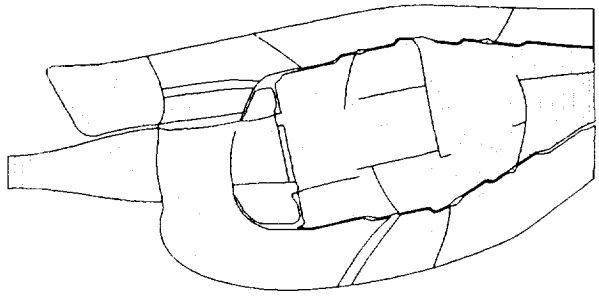
a: original



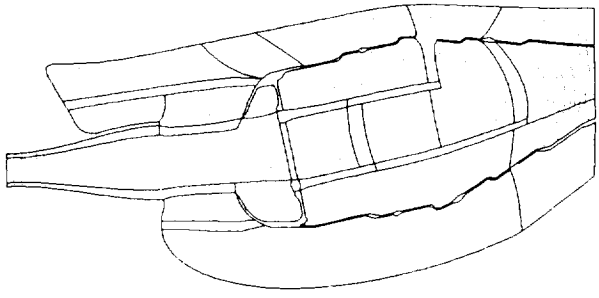
b: first generation



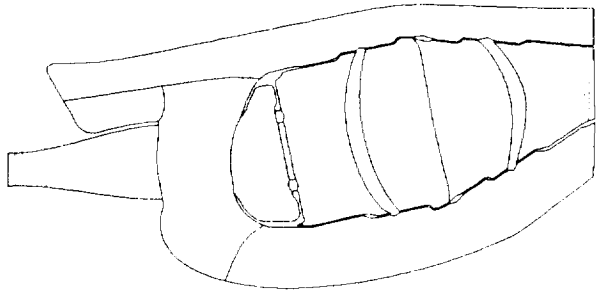
c: second generation



d: third generation

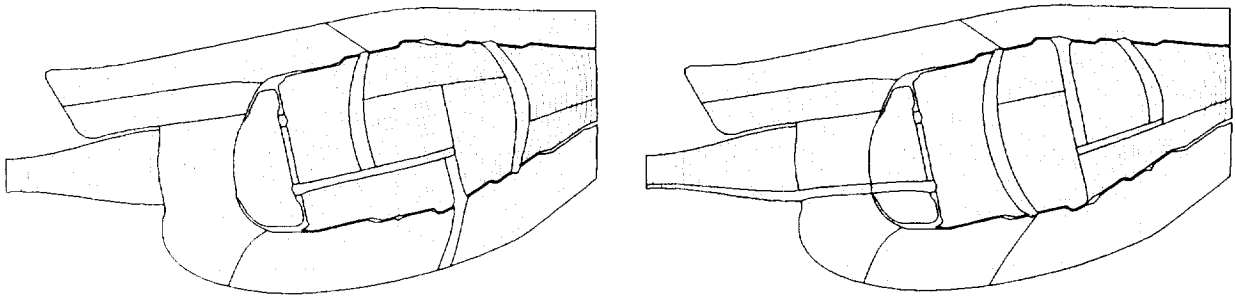


e: fourth original



f: final

Figure 7: Minimization of the number of clusters for test case "combustor".



a: $L_{\text{size}} = 500$ b: $L_{\text{clus}} = 25$

Figure 8: Final clusters for alternative optimizations for test case "combustor".