Improving the finite element ordering for the frontal solver

Stéphane Negre* — **Jean Paul Boufflet**,*** — **Jacques Carlier**,*** **Piotr Breitkopf**,****

* *INSSET*
48 rue du Raspail, 02100 Saint-Quentin
stephane.negre@insset.u-picardie.fr

** *Université de Technologie de Compiègne, Centre de Recherches*
BP 529, 60205 Compiègne Cedex

*** *UMR 6599 HEUDIASYC*
{jean-paul.boufflet,jacques.carlier}@utc.fr

**** *UMR Roberval*
piotr.breitkopf@utc.fr

ABSTRACT *We present the structure of greedy methods used for reordering the nodes or the finite elements of a mesh and we propose two improvements that can be easily implemented in existing software. We investigate a wave reordering method which adapts the reordering strategy during the reordering process. Then, we propose an adaptation of the Tabu Search optimization technique for finite element reordering. The results show the efficiency of the improvements The wave reordering method and the Tabu Search provide very good solutions offering a motivation to carry on research for optimizing their computing time.*

RESUME *Nous présentons la structure des méthodes gloutonnes utilisées pour réordonner les nœuds ou les éléments finis d'un maillage et nous proposons deux améliorations qui peuvent être facilement implémentées dans des codes existants. Nous explorons une méthode de renumérotation par vagues qui adapte la stratégie de renumérotation durant la renumérotation. Ensuite, nous proposons une adaptation de la technique d'optimisation de recherche taboue pour la renumérotation des éléments finis. Les résultats montrent l'efficacité des améliorations. Les méthodes de renumérotation par vagues et de recherche taboue fournissent de très bonnes solutions qui offrent une motivation pour poursuivre la recherche afin d'optimiser leurs temps de calcul.*

KEYWORDS: *reordering, frontal solver, greedy method, wave method, Tabu Search*

MOTS-CLES *renumérotation, solveur frontal, méthode gloutonne, méthode par vague, recherche taboue*

1. Introduction

The actual work concerns the solution of large scale systems of linear equations issued from the finite element method applied to mechanical problems. In particular, we are interested in problems of the size requiring the use of massively parallel computing. In our approach we distinguish three steps:

1. we decompose the domain into subdomains;
2. we search for an optimal numbering of elements in each subdomain;
3. we assign jobs to processors.

In the actual paper we focus the second stage of the process which is critical for the performance of the linear solver. The idea is to use optimization techniques issued from the operation research community and to analyze their impact in the domain of computational mechanics.

Optimizing the numbering of nodes (or elements) may have different goals depending on the algorithm chosen for the resolution of the linear system. The data structure of the global rigidity matrix has to be taken into account:

1. when a direct method (Choleskly, Gauss) is chosen along with skyline structure of the matrix, one has to minimize the banwitdh of the overall system.

2. a direct method with sparse data structure needs maximize the number of zero terms after triangulation. The sequence of nodes optimal for this pattern is usually different from that obtained for the skyline (band) storage.

3. when a frontal solver is used, the optimization concerning the sequence of elements is sought in the way to minimize the frontwidth.

The problems 1 and 2 concern the optimization of the numbering of the nodes and problem 3 the numbering of elements Despite this difference similar algorithms are used. In the finite element literature various algorithms have been proposed. E. Cuthill and J. McKee [CUT 69] have proposed an algorithm to minimize the bandwidth, Gibbs Poole and Stockmeyer [GIB 76] adress the problem of reducing the bandwidth and the profile. By applying an algorithm proposed by A. Razzaque [RAZ 80], one can obtain a finite element reordering from a node reordering. The algorithm proposed by S.W. Sloan for reducing the wavefront and profile is one of the most efficient [SLO 83][SLO 86][SLO 89]. An adapted version of this algorithm has been studied by I.S. Duff, J.K. Reid and J.A. Scott [DUF 89] for a frontal code. A Simulated Annealing approach was proposed by B. A. Armstrong [ARM 84] [ARM 85], considering the profile and the wavefront criteria.

Research in the domain is still active. Recently, the paper of G. Kumfert and A. Pothen [KUM 97] proposed a hybrid method combining the Sloan algorithm and the spectral method. They also discuss the importance of the tuning of the Sloan algorithm. J.K. Reid and J.A. Scott [REI 99] propose some improvements of the Sloan method ameliorating the orginal algorithm of [GIB 76] for computing the pseudo-

diameter of the graph. J.A. Scott [SCO 99] develops a set of reordering tools for a frontal solver.

The work we present concerns the reordering of finite elements for the frontal method [IRO 70] and in particular its parallel version: the multifrontal solver [ESC 92]. This paper presents two improvements that we propose for the Sloan reordering heuristics. Then, we propose two additional approaches:

– the first one spreads successive waves of reorderings in the mesh;

– the second one is based on the Tabu Search (TS) optimization technique [NEG 97] [AAR 97] [GLO 98], which visits the neighbourhood of a current solution like Simulated Annealing does.

In order to compare our methods to the literature a collection of finite element meshes equivalent to the Everstine [EVE 79] set of problems is used as a benchmark. The Everstine test examples have been widely used for testing reordering techniques but are small by today's standards. However, in the domain decomposition context, we do not aim at the optimal ordering for the whole system but we are working locally with subdomains of limited size.

The original Everstine [EVE 79] test suite is given in a matrix form. In appendix A, we show the method of construction of an 'equivalent' finite element mesh giving equivalent matrix topology.

The quality of the optimization process is measured by two criteria:

– the size of the profile of the equivalent global system which, in the case of the frontal method, is equivalent to the sum of frontwidths ;

– an estimator of the number of floating point operations of the frontal solver that we propose in the actual work.

It is clear, that the global time of optimization and of resolution has to be considered. An expensive method giving optimal reordering may not be useful in practical computations. From the research point of view however, it is interesting to start exploring new methods independently of the time constraint in order to find new optimal solutions. The time and the quality constraint may be then progressively relaxed in order to fit overall efficiency criteria.

2. The frontal solver

In order to introduce our notations, we recall here the basic idea of the frontal method. The frontal solver manages a frontal matrix $\mathcal{F}$ whose size varies during the solving process [IRO 70]. The execution time of the solver depends on the ordering of the elements. A reordering vector V_{num} is a vector of size m defining a permutation of the labels of the finite elements. The contributions of each finite element are assembled in turn according to V_{num}. The fontal process alternates between accumulation of element coefficients (assembly) and elimination. An equation is said fully summed

if all the contributions involved have been assembled. A variable becomes 'active' on its first appeareance (incorporation) and is eliminated after its last. We define an elementary step either as the incorporation or as the elimination of an unknown. For a problem with n degrees of freedom, there are n incorporations and n eliminations, corresponding to the assembling order V_{num} of the m finite elements. For simplicity, we assume that there is one degree of freedom per node and consequently, the number of node is n.

The list of active variables at an elementary step k is called the front and is denoted F_k. The number of these active unknowns is called the frontwidth and is denoted $\mid F_k \mid$ (i.e. the wavefront).

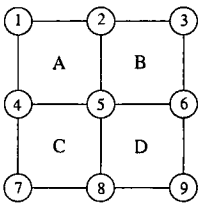

Figure 1. *An example of finite element mesh used to illustrate the frontal solver*

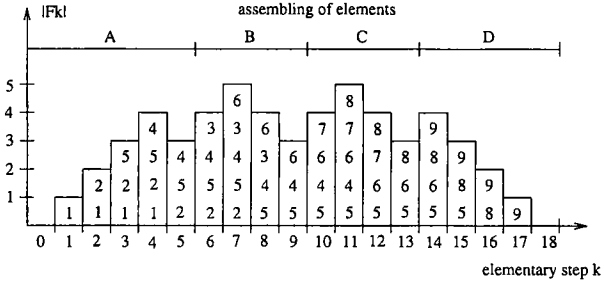

Figure 2. *Evolution of the frontwidth $|F_k|$ when incorporating and eliminating variables of successive finite elements of figure 1*

Figure 1 shows an example of a four finite elements mesh. The bar graph of figure 2 shows the evolution of F_k according to $V_{num} = [A, B, C, D]$. The list of variables in F_k are inside the bar k. The line above the graph indicates the intervals of elementary steps corresponding to the assembling of each finite element.

The incorporation at an elementary step k of one unknown has an O($\mid F_k \mid$) complexity. The computing time for eliminating one unknown at an elementary step k has an O($\mid F_k \mid^2$) complexity.

3. Estimators

To estimate the quality of a reordering we simulate the actual solution of the problem. According to a reordering V_{num}, the virtual elimination consists in managing a list of unknowns of the frontal matrix and counting the successive sizes of successive fronts $\mid F_k \mid$.

The maximum frontwidth $\mid F_{max} \mid$ permits to estimate the storage requirement and is defined by:

$$\mid F_{max} \mid (V_{num}) = \max_{1 \leq k \leq 2\ n} \mid F_k \mid \qquad [1]$$

The profile measures the total amount of the storage requirement:

$$P(V_{num}) = \sum_{k \in incorporation} \mid F_k \mid \qquad [2]$$

The computing time for the frontal solver is estimated [SLO 89] [DUF 89] by the root-mean-squared wavefront, given by:

$$C(V_{num}) = \sqrt{\frac{1}{n} \sum_{k \in incorporation} \mid F_k \mid^2} \qquad [3]$$

Here, we propose the modified estimator:

$$Q(V_{num}) = \alpha_1 \cdot \sum_{k \in incorporation} \mid F_k \mid + \alpha_2 \cdot \sum_{k \in elimination} \mid F_k \mid^2 \qquad [4]$$

where coefficients α_1 and α_2 depend on the mechanical problem considered and on the speed of the computer.

The cost of evaluating any of the above estimators may be estimated in the following way. Let us denote $\overline{N_e}$ the average number of nodes for each element, and $\overline{E_n}$ the average number of elements connected to each node. For each node of the finite element of label $V_{num}[i]$, we check whether it is already incorporated into F_k (giving F_{k+1}). For all finite elements, the complexity is $O(\mid E \mid \cdot \overline{N_e})$. When the finite element of label $V_{num}[i]$ has been assembled, we determine if nodes can be removed from the list F_k. The virtual elimination of all the nodes of an element takes $O(\overline{N_e} \cdot \overline{E_n})$ operations. The cost of a complete virtual elimination is $O(\mid E \mid \cdot \overline{N_e} + \mid E \mid \cdot (\overline{N_e} \cdot \overline{E_n}))$ operations. For a given mesh, the cost is linear with the number of element. A complete virtual elimination has to be performed for evaluating a reordering vector V_{num}.

4. Graph representation of the mesh

The finite element mesh is modelled by two graphs:

– the node graph, denoted $G(Y, V)$, where Y is the set of vertices corresponding to the nodes and V the set of edges. There is an edge between two vertices if the two corresponding nodes belong to the same finite element;

– the element graph is denoted $G(E, U)$, where E is the set of vertices corresponding to the finite elements of the mesh and U is the set of edges. There is an edge between two vertices if the two corresponding finite elements share at least one node.

Figure 3, 4 and 5 show respectively a mesh, the corresponding node graph $G(Y, V)$ and finite elements graph $G(E, U)$. A reordering of the nodes of the mesh corresponds

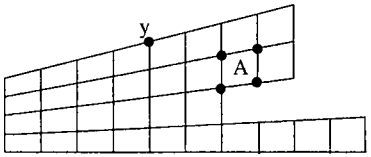

Figure 3. *An example of finite element mesh to illustrate the node graph $G(Y,V)$ and the element graph $G(E,U)$*

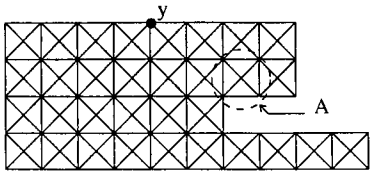

Figure 4. *The node graph $G(Y,V)$ corresponding to the mesh of figure 3*

to a reordering of the vertices of the node graph, while a reordering of the elements corresponds to a reordering of the vertices of the element graph.

The number of vertices of the graph $G(Y,V)$ is given by the number of nodes n and the number of vertices of the graph $G(E,U)$ is given by the number of finite elements m. The number of edges connected to a vertex y is called the degree of the vertex $d(y)$. For instance, the degree of the node y of figure 4 is $d(y) = 5$.

The maximum degree of a graph is noted $\Delta = max_{y \in G} d(y)$. For $G(Y,V)$ of figure 4 $\Delta = 8$. We denote K_i a subgraph of i vertices in which every pair of vertices is adjacent (K_i is a i-clique). The nodes of the finite element of label A of figure 3 correspond to a 4-clique in the $G(Y,V)$ graph of figure 4. Most of reordering al-

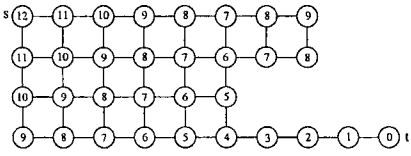

Figure 5. *The finite element graph $G(E,U)$ corresponding to the mesh of figure 3*

gorithms are based on the use of the information associated with the diameter of a graph. We denote $\delta(y,z)$ as the length of the shortest path between the vertices y and z. The diameter $D = \max_{y,z \in G, z \neq y} \delta(y,z)$ is the value of the maximal shortest path [BER 70]. Two vertices s and t, such that the length of the shortest path from s (the starting vertex) to t (the ending vertex) is equal to the value of the diameter, are called peripheral vertices. Figure 5 shows an example of peripheral vertices. Because the computation of the diameter is too time consuming, the authors generally compute a pseudo-diameter D_{pseudo}, which approaches the diameter [SLO 83]. Similarly, two vertices s and t, denoted the pseudo-peripheral vertices, are associated with the pseudo-diameter.

We define the eccentricity of the vertex y as $\delta_y = \max_{z \in G, z \neq y} \delta(y,z)$. A level structure rooted at y is a partition $L(G_y)$ of the graph G into $\delta_y + 1$ levels $L_0(y)$, $L_1(y)$, $L_2(y)$,$\cdots$, $L_{\delta_y)}$ such that:

- $L_0(y) = \{y\}$;
- $L_i(y)$ is the set of vertices having their distance to y equal to i.

The number of vertices $|L_i(y)|$ of each level $L_i(y)$ defines a width. The width of the level structure $L(G_y)$ is defined by: $|L_{max}(y)| = \max_{0 \leq i \leq \delta_y} |L_i(y)|$. The root-mean-squared width of the level structure $L(G_y)$ is defined by:

$$R(y) = \sqrt{\frac{1}{\delta_y} \sum_{i=0}^{\delta_y} |L_i(y)|^2} \qquad [5]$$

Figure 5 shows the partition $L(G_t)$ of the finite element graph $G(E, U)$. The number inside each vertex indicates the level number it belongs to. The pseudo-peripheral vertices have a maximum eccentricity and we compute level structures to select them [CUT 69] [GIB 76][SLO 86]. When selecting a pair (s, t) of pseudo-peripheral vertices, we obtain a kind of axis of the mesh from a graph point of view. Then we may reorder the vertices in the direction of the axis. This reduces the width of the associated level structure and consequently the half bandwidth of the matrix.

5. The Sloan, DRS and S.A. methods

In order to introduce the improvements we propose, we present the outline of a greedy method for reordering and the detail of the original Sloan and DRS methods. We illustrate this presentation with numerical results obtained with our implementation. We also present the adaptation of the Similated Annealing (S.A.) optimization technique carried out by B.A. Armstrong for minimizing the profile and we report the results obtained. This technique is a metaheuristics, and we propose in section 7 the adaptation of the Tabu Search optimization technique which is an another metaheuristics widely used by the operation research community for finding high quality solutions for hard combinatorial problems. We use the standard classification introduced by Duff *et al.* [DUF 89]: an indirect method reorders the node graph while a direct one reorders the element graph.

5.1. *Outline of a greedy method for reordering*

The outline of most greedy algorithms can be presented as in table 1.

At the end of the algorithm the vector V_{num} contains the reordering. The quantity $number\_of\_vertices\_of\_G$ equals n or m depending on the choice of the $G(Y, V)$ or $G(E, U)$ graphs. The methods in the literature differ in the choices of the starting vertex s and of the ending vertex t (line 4), and in the management of C_j the priority list of candidate vertices (lines 7 and 11), defining the greedy strategy applied.

1	compute a pseudo-diameter D_{pseudo} in the graph G;
2	compute L_s the list of starting vertices;
3	compute L_e the list of ending vertices;
4	select $s \in L_s$ and $t \in L_e$;
5	for all vertex y of G compute $\delta(t,y)$;
6	$V_{num}[1] \leftarrow s$;
7	compute C_1 the priority list of candidate vertices;
8	for $j \leftarrow 2$ to $number\_of\_vertices\_of\_G$ do
9	selects $y \in C_{j-1}$ with the highest priority;
10	$V_{num}[j] \leftarrow y$;
11	compute C_j by updating C_{j-1}.

Table 1. *The outline of a greedy algorithm for reordering problems*

5.1.1. *The Sloan method*

The Sloan method is an indirect one. The starting vertex s of algorithm of table 1 is the vertex having the minimum degree and t is arbitrarily chosen. Let us now consider an iteration j. We denote H_j the list of the already reordered nodes. The list C_j of candidate vertices is composed of the vertices $k \notin H_j$ having a distance to the nodes of H_j smaller than or equal to 2. This distance is used to update the priorities of the vertices in the list C_j. Let us consider a vertex $y \in C_j$. The vertex y has an initial degree $d(y)$. We define:

- ε_1 as the set of neighbours of y that are already reordered;
- ε_2 as the set of neighbours of y that are also adjacent to H_j vertices;
- $deg(y) = d(y) - |\varepsilon_1 \cup \varepsilon_2|$.

A priority p_y is computed as follows:

$$p_y = -w_1 \cdot deg(y) + w_2 \cdot \delta(t,y) \qquad [6]$$

The list C_j and the associated priorities are updated at each iteration j. The first part of the priority expression (6) corresponds to a local consideration. It avoids an excessive growth of the frontwidth. The second part of the expression (6) corresponds to a global view of the finite element mesh. The objective is to build a level by level reordering orthogonal to a path from s to t.

The weights w_1 and w_2 are set experimentally. Sloan suggests the weights $w_1 = 2$ and $w_2 = 1$.

We denote SLO our implementation of the Sloan method. Figures 6 and 7 show respectively the profile $P(V_{num})$ and the computing time estimator $Q(V_{num})$ for each mesh of the test set using the Sloan method.

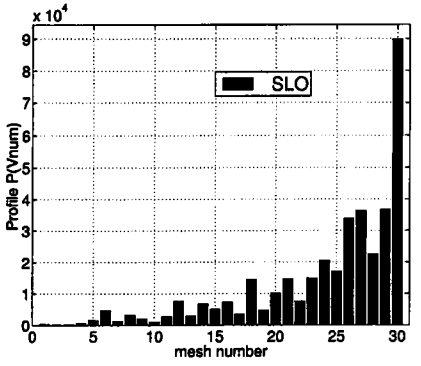

Figure 6. *Global storage requirement as measured by the profile $P(V_{num})$ with Sloan reorderings for the set of test problems*

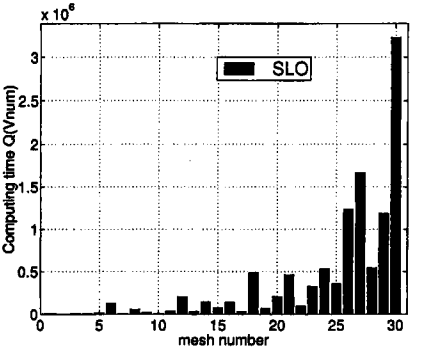

Figure 7. *The computing time estimator $Q(V_{num})$ with Sloan reorderings for the set of test problems*

5.1.2. *The DRS methods*

The two DRS methods are direct ones. The strategy is adapted from Sloan to take into account the finite element graph $G(E,U)$. In both algorithms the starting vertex s is chosen using the same criterion as used by Sloan. Instead of choosing t arbitrarily, the vertex t is chosen in the set of nodes $L_{\delta_s}(s)$ of the last level according to the following criterion. For each vertex $t_i \in L_{\delta_s}(s)$, a level structure is computed and the ending vertex that gives the minimum width $|L_{max}(t_i)|$ is selected. Considering a reordering iteration j, we denote H_j as the list of the already reordered vertices (finite elements). A list C_j of candidate vertices is composed of the vertices adjacent to vertices of H_j. Let us consider a vertex $y \in C_j$. The priority of the first DRS method is computed as:

$$p_y = -w_1 \cdot deg(y) + w_2 \cdot \delta(t,y) + w_3 \cdot var(y) \qquad [7]$$

The value $deg(y)$ is the same computed as in the Sloan method (6), except that here y corresponds to a vertex of $G(E,U)$. The value $var(y)$ is the number of nodes (nodal points) of the considered finite element y. This term permits selecting between two elements having the same priority according to the first two terms of (7). [DUF 89] argues that it is better to select the element having more nodal points because it permits the elimination of more unknowns during future steps without increasing the front. However, the results of this strategy are not as good as expected.

For this reason, the second greedy algorithm differs by a change in the local strategy. Its objective is to reduce both the number of active elements and the number of active unknowns. The priority is now computed as follows:

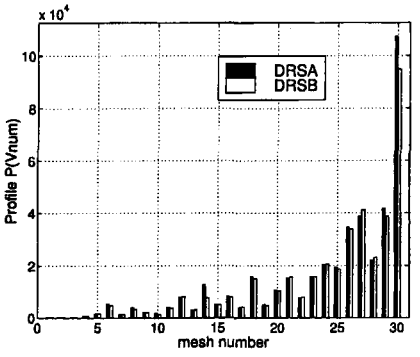

Figure 8. *Global storage requirement as measured by the profile $P(V_{num})$ with DRSA and DRSB for the set of test problems*

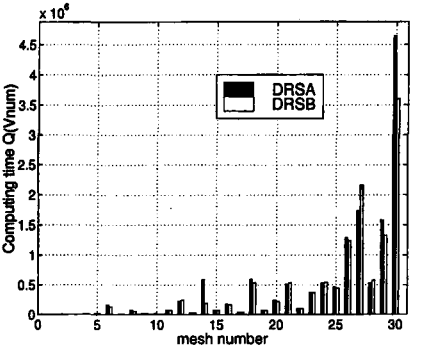

Figure 9. *Computing time estimator $Q(V_{num})$ with DRSA and DRSB for the set of test problems*

$$p_y = -w_1 \cdot diff(y) + w_2 \cdot \delta(t, y) - w_3 \cdot \widehat{deg}(y) \qquad [8]$$

The quantity $diff(y)$ corresponds to the difference between the number of new unknowns assembled into the frontal matrix and the number of unknowns that can be eliminated after assembling the finite element y. The value $\widehat{deg}(y)$ represents the degree of vertex y minus the number of its adjacent vertices that have so far been reordered. The values suggested for the weights w_1, w_2, and w_3 for both strategies are respectively 12, 6 and 1. The third criterion is useful for breaking ties. We have implemented both greedy algorithms in order to compare with our methods.

In the figures, we denote $DRSA$ the first method and $DRSB$ the second one. Figures 8 and 9 show respectively the $P(V_{num})$ and the $Q(V_{num})$ estimators for each mesh of the test set problems. The $DRSB$ method provides better results.

5.2. *The Simulated Annealing method*

A Simulated Annealing method has been proposed by B. A. Armstrong to reorder the nodes for minimizing the profile. It can be used as an indirect method for the frontal problem. Simulated Annealing is an iterative improvement technique exploring successively the neighbourhood of a current solution.

A parameter, analogous to the temperature of a physical system to be cooled, controls which perturbations to the system will be allowed visiting the feasible solutions. Using it requires defining and tuning of several parameters and functions. The neighbourhood function used is the classical 2-OPT neighbourhood that exchanges two

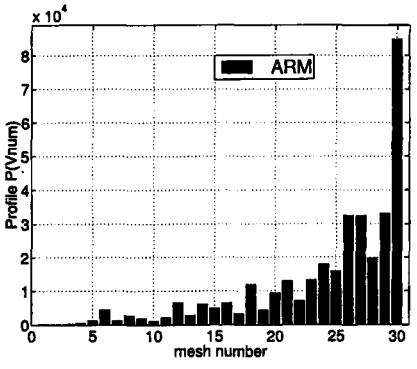

Figure 10. *Global storage requirement as measured by the profile $P(V_{num})$ with ARM for the set of test problems [ARM 85]*

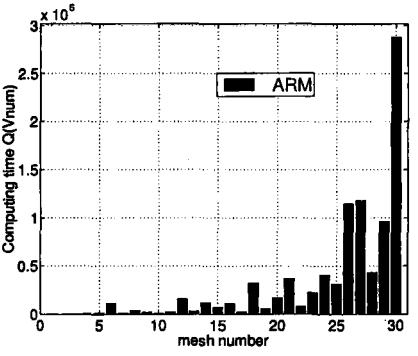

Figure 11. *Computing time estimator $Q(V_{num})$ with ARM for the set of test problems [ARM 85]*

nodes. The initial temperature is defined in [ARM 85], but some other parameters are insufficiently specified and the experiment is not reproducible.

This method produces very good solutions and [ARM 85] argues they are optimal or near optimal. However it is too slow for a practical use. Figures 10 and 11 show the original results obtained by [ARM 85] for the $P(V_{num})$ and the $Q(V_{num})$ estimators for each mesh of the test set. We refer to this method as ARM.

6. Two improvements of greedy algorithms

The weights suggested in Sloan and DRS methods favour the local criterion over the global one, the first term in (6), (7) and (8) is significatively larger than the second. However, the choice of the ending vertex t among the list of ending vertices influences the quality of the results because it represents the basis of the global criterion in the greedy strategies. [SLO 89] chooses it arbitrarily, but [DUF 89] improves the previous results by choosing a vertex $t \in L_{\delta_s}(s)$ minimizing the width $|L_{max}(t)|$.

We propose to choose the vertex t that gives the level structure having the smallest root-mean-squared width (see formula (5) in section 4). Thus we take into account the sizes of each level (not only the largest one). We denote respectively $SLO1$ and $DRSB1$ the modified versions of SLO and $DRSB$.

The second improvement consists in computing the $\delta(t, y)$ in a specific way. Figure 12 shows a node graph and illustrates our idea. We have four nodes for each square finite element. Let us consider the set of ending vertices $\{t_1, t_2, t_3, t_4, t_5\}$. Figure 12 shows the behavior of the computation of $\delta(t_1, y)$. The vertices that are linked by

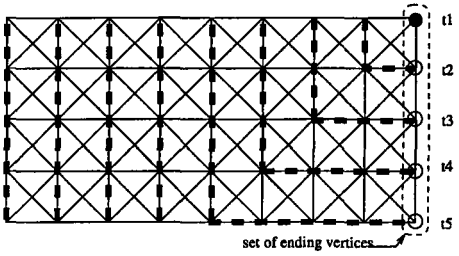

Figure 12. *Node graph $G(Y,V)$ with successive layers of $\delta(t_1,y)$ when considering a single root vertex t_1*

a dotted bold line represent a set of vertices having the same value of $\delta(t_1,y)$ that corresponds to an isovalue layer.

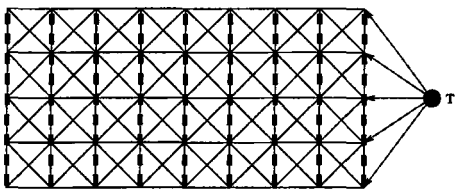

Figure 13. *Node graph with successive layers when considering a virtual supervertex*

If we only consider the criterion $\delta(t,y)$ in the expressions of priority (6), (7) and (8), the reordering shall have a layer by layer behavior. Each layer corresponds to a level $L_i(t_1)$ of the level structure considering the vertex t_1 as the root. Now consider figure 13. We artificially create the virtual super-vertex T and we compute $\delta(T,y)$. It is clearly seen that $|L_{max}(T)|$ is lower than $|L_{max}(t_1)|$.

We denote respectively $SLO2$ and $DRSB2$ the modified version of SLO and $DRSB$.

Figure 14 and 15 show respectively the percentage of improvement of the profile $P(V_{num})$ and the computing time estimator $Q(V_{num})$.

The results obtained with $SLO1$ and $SLO2$ are expressed in percentage of improvement comparing with the result of SLO. Results of $DRSB1$ and $DRSB2$ are also relatively compared with results of $DRSB$. We observe on figures 14 to 17 that the first improvement provides better or equivalent solutions for the majority of the meshes of the test set. In few cases we obtain a slight degradation. Considering the computing time estimator $Q(V_{num})$, figure 15 shows for $SLO1$ that one improvement reaches 12.5% (mesh number 24) and four other are larger than 3%. Figure 17 shows for $DRSB1$ an improvement of up to 19% (mesh number 11). This first improvement

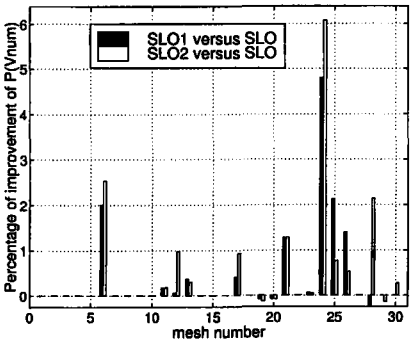

Figure 14. *Percentage of improvement for global storage requirement as measured by the profile $P(V_{num})$ with $SLO1$ and $SLO2$ relative to SLO*

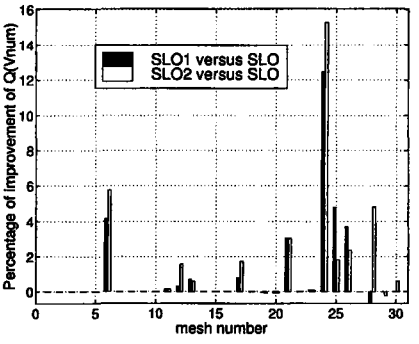

Figure 15. *Percentage of improvement for the computing time estimator $Q(V_{num})$ with $SLO1$ and $SLO2$ relative to SLO*

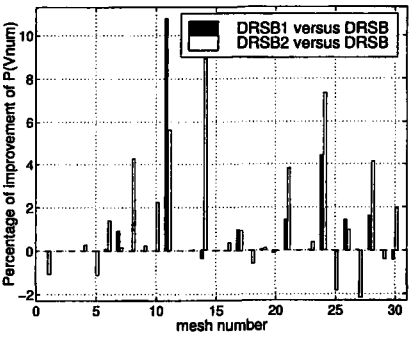

Figure 16. *Percentage of improvement for the global storage requirement as measured by the profile $P(V_{num})$ with $DRSB1$ and $DRSB2$ relative to $DRSB$*

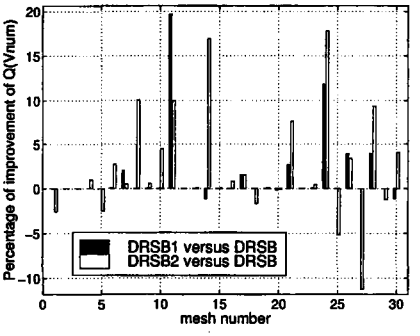

Figure 17. *Percentage of improvement for the computing time estimator $Q(V_{num})$ with $DRSB1$ and $DRSB2$ relative to $DRSB$*

can be applied with benefit because it rarely degrades, it generally provides equivalent solutions and allows to compute better solutions with a similar computing time.

In figure 15 we observe roughly the same behaviour for $SLO2$ as for $SLO1$, except for a case where $SLO2$ improves (mesh number 28). Figure 17 shows important degradations using $DRSB2$ for two cases (meshes number 25 and 27), although one can remark that the number of meshes where the reorderings are improved increases. The second improvement applied with $DRSB$ method also provides good results but, on few cases, the degradation can be significative.

7. Wave Reordering Methods

[KUM 97], [REI 99] and [SCO 99] studied the effect of adjusting the weights in the priority functions of the SLO or DRS algorithms. The authors observe that the set of weights giving minimum wavefronts differs with the problem and the method. Our feeling is that the set of weights has to be adapted during the reordering process itself. Weights efficient at one part of the mesh may not be optimal for another part. Consecutively, we propose building a wave reordering method that spreads waves of reorderings on the mesh. We apply the same heuristics with nbm sets of weights from the same starting vertex s_0, until a number siz of vertices has been reordered. Then we evaluate the nbm partial reorderings obtained and we choose the best one according to our estimation $Q(V_{num})$. Hence, the $V_{num}[1 : siz]$ vector is computed. This partial solution has a frontier with the vertices that are not yet reordered. We have to choose another starting vertex s_1 on this frontier and to repeat the process (see figure 18). Then, using the same process, we compute the V_{num} vector between indices $siz + 1$ to $2 \cdot siz$. By repeatedly applying this principle, we finally obtain an element reordering vector V_{num}.

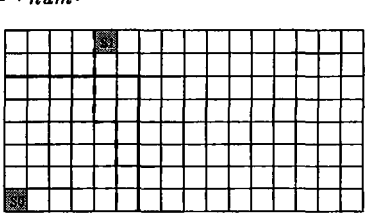

Figure 18. *Two reordered areas of* $siz = 36$ *elements on a grid mesh*

This method can be viewed as a succession of waves of reorderings on the graph. Figure 18 illustrates our idea on a simple grid mesh: two frontiers are obtained by applying a reordering method with two sets of weights (the value of siz is 36). The gray patterns inside the finite elements s_0 and s_1 show the equivalent starting vertices in $G(E, U)$. We assume that the dotted line is the frontier obtained by the best local reordering. The vertex s_1 is chosen on this frontier and the process is applied again. Indeed, the subsets of reordered elements are determined by the number siz of elements to reorder and by the set of weights.

The behavior of this method depends on the value of the parameter siz. This value can be constant or it can be modified between two consecutive waves, leading to an adaptative method. The principle can be applied on the node graph $G(Y, V)$ or on the element graph $G(E, U)$.

We focus here on constant sizes of packets. The variable siz denotes the number of elements to be reordered at each step, and $nb = \lceil \frac{|E|}{siz} \rceil$ the number of packets. The last packet has $(\mid E \mid -(nb - 1).siz)$ elements. This particular case does not appear in the outline of the algorithm we present below, however it is easy to take it into account.

1	compute a pseudo-diameter D_{pseudo};
2	compute L_s the list of starting vertices;
3	compute L_e the list of ending vertices;
4	$\forall t \in L_e$ compute $R(t)$;
5	select $\tilde{t} \in L_e$ minimizing $R(t)$;
6	$\forall$ vertex y of G compute $\delta(\tilde{t}, y)$;
7	$nb \leftarrow \lceil \frac{\lvert E \rvert}{siz} \rceil$;
8	for $i \leftarrow 0$ to $nb - 1$ do
9	for $j \leftarrow 1$ to nbm do
10	reorder siz elements using the set of weigths j;
11	estimate the partial reordering number j obtained;
12	if the partial reordering is the best found so far memorise;
13	enfor
14	insert the best partial reordering in $V_{num}[i \cdot siz + 1 : (i+1) \cdot siz]$;
15	endfor
16	return V_{num}

Table 2. *The outline of the wave reordering algorithm*

In order to study the behavior of our wave method, we have tried in turn all the possible values of the variable siz from 1 to $|E|$. Obviously, choosing $siz = |E|$ is equivalent to reorder the mesh nbm times using a given set of weights at a time. We rarely obtain the best result with this particular value of siz that confirms the interest of our approach. The outline of our algorithm is reported in table 2. Note that we use the first improvement (see section 6) to select the ending vertex $\tilde{t}$. This outlined algorithm has to be adapted if we design an indirect method that reorders the node graph.

Two wave reordering algorithms have been designed and tested. The first one is based on SLO algorithm with four sets of weights ($nbm = 4$). The second one is based on the DRSB algorithm and seven sets of weights are tried for each wave ($nbm = 7$). The subscripts used to present the two wave methods are respectively $WSLO$ for Wave SLOan method and $WDRSB$ for Wave Duff Reid Scott method.

Figures 19 to 22 show the best results we obtain for each mesh of the test set by scanning the different value of siz, the size of waves. Figure 19 shows the percentages of improvement obtained with $WSLO$ for $P(V_{num})$ comparing with the results of SLO while figure 20 concerns the $Q(V_{num})$ criterion. Similarly, figure 21 shows the percentages of improvement obtained with $WDRSB$ for $P(V_{num})$ comparing with the results of $WDRSB$ while figure 22 concerns the $Q(V_{num})$ criterion. We observe that figure 19 follows the same trend as figure 20 while figure 21 follows the same trend as figure 22.

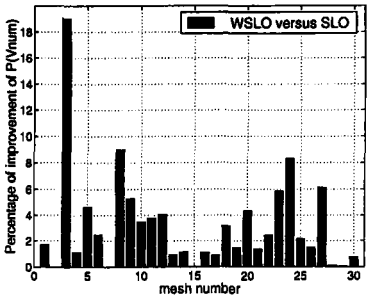

Figure 19. *Percentage of improvement for the global storage requirement measured as the profile $P(V_{num})$ with $WSLO$ relative to SLO*

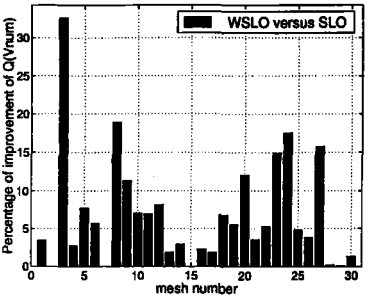

Figure 20. *Percentage of improvement for the computing time estimator $Q(V_{num})$ with $WSLO$ relative to SLO*

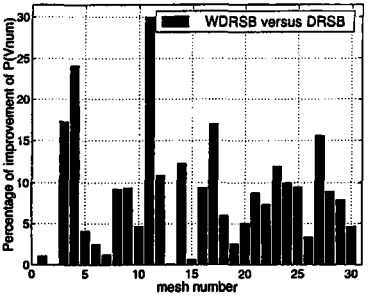

Figure 21. *Percentage of improvement for the global storage requirement measured as the profile $P(V_{num})$ with $WDRSB$ relative to $DRSB$*

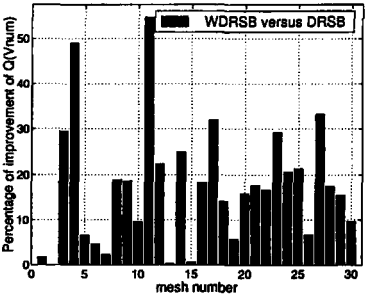

Figure 22. *Percentage of improvement for the computing time estimator $Q(V_{num})$ with $WDRSB$ relative to $DRSB$*

Considering the computing time estimator $Q(V_{num})$, the gain is larger than 10% for seven meshes with $WSLO$ (see figure 20) and the gain is larger than 20% for ten meshes with $WDRSB$ (see figure 22). The sets of weights have been chosen to favour, in turn, a part of the priority expression (6) or (8) over the others.

For a given value of the parameter siz, we experimentally observe that the successive best partial reorderings are not obtained with a unique set of weights. The wave reordering method chooses the best local reordering during the process. However, we don't know a priori the optimal value of the size of waves. For some value of the parameter siz, the solution we found is worst than those obtained with the original Sloan or DRS methods using the same sets of weights. We think that it is due to the choice of the starting vertex at the beginning of each wave. A bad choice can brake the continuity of the reordering. Further investigations are also in progress in order to determine good values for the parameter siz.

8. Applying Tabu Search for finite element reordering

We first present an overview of the Tabu Search (TS) optimization technique. Then we propose a TS version for minimizing $Q(V_{num})$, and discuss our results.

8.1. *Overview of the Tabu Search metaheuristics*

TS has been applied successfully to numerous combinatorial optimization problems [AAR 97]. It is a local search algorithm which starts with an initial solution σ, and then iteratively replaces it by another solution $\hat{\sigma}$ in the neighbourhood $\mathcal{N}(\sigma)$ of σ. Any neighbour of σ is obtained by a licit move μ: $\hat{\sigma} = \sigma \oplus \mu$ ($\mu \in M(\sigma)$, where $M(\sigma)$ is the set of licit moves from σ). Generally, moves consist in exchanging some elements of a sequence associated with σ.

1	Choose an initial solution $\sigma \in S$;
2	$\sigma^* \leftarrow \sigma$ and $l \leftarrow 0$;
3	Set $l \leftarrow l + 1$ and generate a subset $\mathcal{V}^*$ of solutions in $\mathcal{N}(\sigma)$ such that the tabu conditions are not violated ($\hat{\sigma} \in \mathcal{V}^*$ and $\mu \notin \mathcal{T}$);
4	Choose the best $\hat{\sigma} = \sigma \oplus \mu \in \mathcal{V}^*$ with respect to f, or some solution $\hat{\sigma}$ satisfying the aspiration condition;
5	$\sigma \leftarrow \hat{\sigma}$;
6	If $f(\sigma) < f(\sigma^*)$, then $\sigma^* \leftarrow \sigma$;
7	Update the tabu list $\mathcal{T}$ and aspiration conditions;
8	If a stopping condition is met, then stop. Else return Step 3.

Table 3. *Overview of a Tabu Search method*

In order to escape from a local minimum, it is necessary to prevent cycles. TS keeps a tabu list $\mathcal{T}$ of recent moves: if a move belongs to the list $\mathcal{T}$, it is considered as illicit. It is also costly to explore the full neighbourhood and to evaluate for each move the objective function f. So, we often only generate a subset $\mathcal{V}^*$ of solutions in $\mathcal{N}(\sigma)$ such that the tabu conditions are not violated.

Because we only manage forbidden moves in the tabu list $\mathcal{T}$ instead of full solutions, it may happen that one of illicit moves improves the search process. If a tabu move has an aspiration level higher than a threshold value, then its tabu status is dropped. Usually this aspiration threshold corresponds to the value of the objective function $f(\sigma^*)$ of the best solution visited so far.

The stopping conditions may be:

– the neighbourhood of $\mathcal{N}(\sigma)$ is empty;

– the number of iterations is larger than the maximum number of iterations allowed;

– the number of iterations since the last improvement of σ^* is larger than a threshold;

– evidence can be given that an optimum has been obtained.

An overview of a Tabu Search (TS) metaheuristics is presented in table 3.

8.2. *Adaptation of Tabu Search for the frontal solver*

We have used two basic neighbourhoods for defining our TS methods:

– the classical $2-OPT$ neighbourhood introduced for the Traveling Salesman Problem [AAR 97]. The idea is to exchange the labels of the finite elements at indices i and j in V_{num} (see the left part of figure 23).

– the second neighbourhood consists in removing the label of finite element $V_{num}[i]$, shifting left the content of V_{num} from i to $j-1$ and inserting the stored label before the index j (see the right part of figure 23).

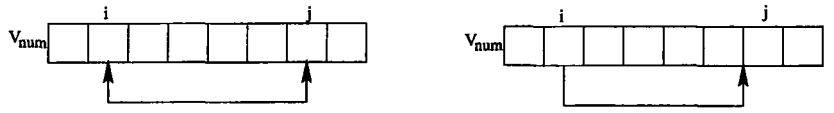

Figure 23. *Transposition and insertion neighbourhood examples*

The preliminary results [NEG 97] show that the insertion neighbourhood leads to better results and we here present this version.

Using the basic insertion neighbourhood, the total number of neighbours of a solution is $O(\mid E \mid^2)$. We propose to reduce the size of neighbourhood to explore. We have experimentally observed that it is rarely interesting to develop simultaneously disconnected fronts. Hence, we introduce the interval $[i,j]$ associated with the finite element of label e that defines its life span. This finite element has a list of variables L_e where:

– the index i is the minimum index of V_{num} where a variable $x \in L_e$ appears at the first time in the front;

– the index j is the maximum index of V_{num} where all the variables of L_e have been eliminated.

Now, we consider the insertion moves of this element only in the interval $[i,j]$. In this way we avoid creating disconnected fronts. If we denote $\overline{D_E}$ the average life span of the elements, the number of insertion moves is now O($\mid E \mid \cdot \overline{D_E}$) instead of $O(\mid E \mid^2)$. We have defined the $\mathcal{N}_{ins}(\sigma)$ neighbourhood function.

The computation of $Q(V_{num})$ is bounded by $O(\mid E \mid \cdot(\overline{N_e}\cdot\overline{E_n}))$, but it is too costly to compute this evaluation for each move. We can reduce this time if we compute only the modifications of the evolution of the front from the current solution. For each index i of V_{num}, we manage:

– the list F_k^i of frontal nodes after the assembling of the finite element of label $V_{num}[i]$;

– the successive sizes $|F_k^i|$ of the frontal matrix $\mathcal{F}$ corresponding to the assembling of the finite element of label $V_{num}[i]$;

– the local value of the estimation $Q(V_{num[1:i]})$ corresponding to the evaluation of Q from $V_{num}[1]$ to $V_{num}[i]$.

Let us assume now that we insert the finite element of label $V_{num}[i]$ before the finite element of label $V_{num}[j]$ (with $i < j$). Using $Q(V_{num[1:(i-1)]})$ we need only to compute a partial virtual elimination between index $i-1$ and index j of V_{num}. Then, using the stored sizes of lists F_k of frontal variables, we simply sum $|F_k|$ when a new variable is added and $|F_k|^2$ when a variable is eliminated from index j to index m. Experimentally, this refinement halves the computing times of the Tabu Search methods.

The tabu list $\mathcal{T}$ for this initial version of $TSRE_{ins}$ is managed using the FIFO rule and its sizes have been set to 7.

We use two stopping conditions:

– a maximum number of iterations;

– a maximum number of iterations since the last improvement of σ^*.

The first one has been experimentally set to 1000 and the second one to 15.

The tabu status of a move can be dropped, if an aspiration condition is satisfied. This is done as follows: if there is no improving move, and if there is a tabu move that improves the best objective function found so far, then the tabu status of this move is dropped. This defines our aspiration condition.

Experimentally, the algorithms stops before 1000 iterations. But we have noticed that, if we use an initial solution with low quality, the number of iterations increases. For this reason, the $WSLO$ and $WDRSB$ algorithms are used to generate initial solutions.

8.3. *Further improvements of the Tabu Search*

We observe large computing times using the $TRSE_{ins}$ method. The neighbourhood is poorly exploited compared with the cost of its generation and evaluation. Let us consider a neighbourhood $\mathcal{N}_{ins}(\sigma)$ and the subset of improving solutions $\mathcal{N}_{imp}(\sigma) \subset \mathcal{N}_{ins}(\sigma)$. Since even the best improving move may have a negligible effect, it makes sense to apply several improving moves of $\mathcal{N}_{imp}(\sigma)$ simultaneously at iteration l.

However, we experimentally observe that mutually exclusive moves sometimes lead to an important increase of the objective function. We have to take into account the order to carry out the moves. They are sorted in decreasing order of improvements of the objective function. Then, considering this order, the moves are evaluated in

turn. If the evaluated move still improves the objective function after the previous moves have been performed, it is accepted, otherwise it is rejected.

The management of the tabu list $\mathcal{T}$ is now different because many moves are performed at each TS iteration. We do not store moves but we use a vector Θ of size $|E|$. When an element is moved to $V_{num}[i]$, the $\Theta[i]$ is set to a number that corresponds to a tabu-active status during this number of iterations. It is called the tabu tenure of the move. These values are updated at each iteration. After several experimental tries, the tabu tenure has been fixed to 5 because the neighbourhood is often empty for larger values.

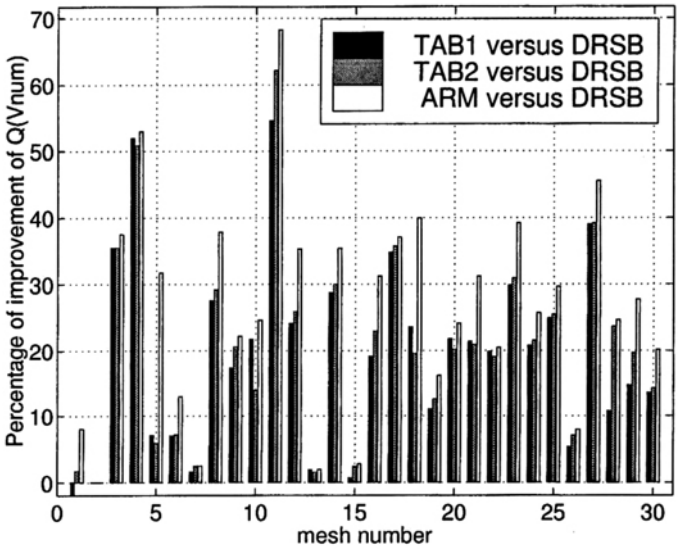

Figure 24. *Tabu Search: Percentage of improvement for the computing time estimator $Q(V_{num})$ with $TAB1$, $TAB2$ and ARM relative to $DRSB$*

However, it is still difficult to escape from a local optimum because the metaheuristics still ambles along a flat valley. When this happens, we perform all the moves that have the same evaluation as the current solution σ in order to escape from the local optimum more quickly.

The subscripts are $TAB1$ for the TS metaheuristics initialized with a solution provided by $WSLO$, and $TAB2$ if the initial solution is provided by $WDRSB$.

Figure 24 shows the percentage of improvement for the computing time estimator $Q(V_{num})$ relative to $DRSB$ for each mesh of the test set using $TAB1$, $TAB2$ and ARM.

The gains we obtain are substantial. We observe that we haven't an important difference between $TAB1$ and $TAB2$ because the initial solutions provided by $WSLO$

or $WDRSB$ are similar and the improvements are carried out from these initial solutions. We have set the maximum number of iterations to 1000 in order to study the behaviour of the TS, but the optimization process stops well before 1000 iterations because the threshold of maximum number of iterations since the last improvement of σ^* is reached. The gain is near or larger than 20% for 19 meshes and can reach 50% that potentially halves the condensation time. Figure 24 also shows the gain of the Simulated Annealing ARM and one can see that the solutions we obtain are near. The Simulated Annealing has a stochastic behaviour but Tabu Search doesn't, so if a Simulated Annealing runs a very long time it can find better solutions. We don't obtain equivalent solutions because the exploring process stops because the threshold of iterations is overtaken before an improving solution has been found. We have significantly reduced the cost of the first version of our Tabu Search by reducing the neighbouring, by performing many moves in turn and by evaluating the criterion by difference to the previous solution. However, the optimization process is too time-consuming for a practical use. But we observe that the major part of the gains is obtained during the first ten iterations of the TS. This is a promising observation for testing more efficient search techniques that can lead to efficient practical methods.

9. Conclusion and outlines

The Everstine collection has been used for our tests. The collection includes very different types of structures from simple regular two-dimensional meshes to complex three-dimensional ones. Their relatively small size is well suited for exploring new approaches which is the subject of the present paper. One has to be careful when applying the conclusions to real cases. The large scale examples were not treated in the present work and will be dealt with in forthcoming publications. The Everstine test examples are not available in element form. In order to perform our test on the element graph, we have used an algorithm to build it from the node graph (see appendix A).

The improvements proposed in this paper may be easily implemented in existing software. The gains are significative on relatively small finite element meshes of the explored collection of benchmark tests. Due to the nature of the proposed improvement one may expect that the relative gains will decrease with the problem size. For this reason the developped approaches are better suited for parallel solvers with decomposition into large number of subdomains. At the actual stage, standard domain decomposition tools may be used. One of the aspects of our future work concerns a mixed approach of simultaneous domain decomposition and optimal numbering.

The presented results are based on estimations of CPU time. However, when comparing with real computation, our modified CPU estimator involving a quadratic term provides a precise estimation. The alternative approches by wave reordering and Tabu Search are still too costly for practical use, but in our opinion they merit further investigation.

10. References

[AAR 97] AARTS E., LENSTRA J. K., *Local search in combinatorial optimization*, John Wiley and Sons, 1997, Series in Discrete Mathematics and Optimization.

[ARM 84] ARMSTRONG B. A., "A Hybrid Algorithm for Reducing Matrix Bandwidth", *Int. J. for Num. Meth. in Eng.*, vol. 20, 1984, p. 1929-1940.

[ARM 85] ARMSTRONG B. A., "Near-Minimal Matrix Profiles and Wavefronts for Testing Nodal Resequencing Algorithms", *Int. J. for Num. Meth. in Eng.*, vol. 21, 1985, p. 1785-1790.

[BER 70] BERGE C., *Graphes et Hypergraphes*, Dunod, 1970.

[CUT 69] CUTHILL E., MCKEE J., "Reducing the bandwidth of sparse symmetric matrices", *Proc. 24th National Conference of the Association for Computing Machinery*, 1969, p. 157-172, New Jersey.

[DUF 89] DUFF I. S., REID J. K., SCOTT J. A., "The Use of Profile Reduction Algorithms with a Frontal Code", *Int. J. for Num. Meth. in Eng.*, vol. 28, 1989, p. 2555-2568.

[ESC 92] ESCAIG Y., "Décomposition de domaines multiniveaux et traitement distribué pour la résolution de problèmes de grande tailles", PhD thesis, Université de Technologie de Compiègne, 1992.

[EVE 79] EVERSTINE G. C., "A Comparison of Three Resequencing Algorithms for the Reduction of Matrix Profile and Wavefront", *Int. J. for Num. Meth. in Eng.*, vol. 14, 1979, p. 837-853.

[GIB 76] GIBBS N. E., JR W. P., STOCKMEYER P. K., "An Algorithm for Reducing the Bandwidth and the Profile of a Sparse Matrix", *SIAM J. Numer. Anal.*, vol. 13, num. 2, 1976, p. 236-250.

[GLO 98] GLOVER F., LAGUNA M., *Tabu Search*, Kluwer Academic Publishers, 1998.

[IRO 70] IRONS B. M., "A Frontal Solution Program for Finite Element Analysis", *Int. J. for Num. Meth. in Eng.*, vol. 2, 1970, p. 5-32.

[KUM 97] KUMFERT G., POTHEN A., "Two Improved Algorithms for Envelope and Wavefront Reduction", *BIT nordisk tidskrift for informationsbehandling*, vol. 37, 1997, p. 559-590.

[NEG 97] NEGRE S., BOUFFLET J. P., CARLIER J., "Reordering finite elements for the frontal method", *Second Metaheuristics International Conference, MIC'97*, July 1997, Sophia Antipolis, France.

[RAZ 80] RAZZAQUE A., "Automatic Reduction of Frontwidth for Finite Element Analysis", *Int. J. for Num. Meth. in Eng.*, vol. 15, 1980, p. 1315-1324.

[REI 99] REID J. K., SCOTT J., "Ordering Symmetric Sparse Matrices for Small Profile and Wavefront", *Int. J. for Num. Meth. in Eng.*, vol. 45, 1999, p. 1737-1755.

[SCO 99] SCOTT J., "On Ordering Elements for a Frontal Solver", *Commun. Numer. Meth. Engng.*, vol. 15, 1999, p. 309-323.

[SLO 83] SLOAN S. W., "Automatic Element Reordering for Finite Element Analysis with Frontal Solution Schemes", *Int. J. for Num. Meth. in Eng.*, vol. 19, 1983, p. 1153-1181.

[SLO 86] SLOAN S. W., "An Algorithm for Profile and Wavefront Reduction of Sparse Matrices", *Int. J. for Num. Meth. in Eng.*, vol. 23, 1986, p. 239-251.

[SLO 89] SLOAN S. W., "A FORTRAN Program for Profile and Wavefront Reduction", *Int. J. for Num. Meth. in Eng.*, vol. 28, 1989, p. 2651-2679.

Appendix A

The Everstine test examples are not available in element form. Only the matrix form are given from which the node graph may be constructed in a unique manner. We compute the $G(E,U)$ graph from the $G(Y,V)$ graph by building and filtering sets of cliques. Let us denote C_i the list of cliques of size i. Initially each edge of $G(Y,V)$ corresponds to a 2-clique that gives C_2.

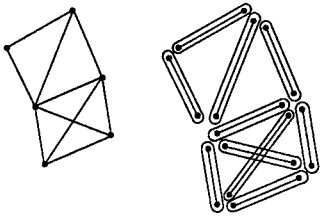

Figure 25. *A $G(Y,V)$ graph and the corresponding 2-cliques*

Figure 25 shows on the left part the node graph $G(Y,V)$ of a mesh composed of two triangular and one quadrilateral finite elements. On the right part we have the 2-cliques corresponding to the edges.

Consider $K_2^j \in C_2$ the clique number j. For each vertex $y \notin K_2^j$ but adjacent to a vertex of K_2^j, we check if it is adjacent to every vertex of K_2^j. If it is, we add the corresponding K_3 in the C_3 list. Then we scan C_2: if a K_2^j belongs to an element of C_3, it is removed from C_2. By repeatedly applying this process, we compute an 'equivalent' finite element scheme. The process is stopped, either when we cannot build a C_{i+1} list or when C_Δ has been computed. Each clique corresponds to an 'equivalent' finite element.

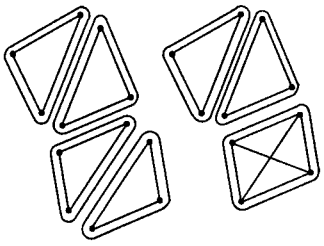

Figure 26. *The 3-cliques and the 'equivalent' finite elements*

On the left part of figure 26 we show the 3-cliques obtained by applying an iteration of the algorithm and we obtain in our example the initial finite elements on the right part. However, in some cases, the 'equivalent' finite element doesn't fit the original finite element model but corresponds to an 'equivalent' scheme from a computing point of view. For instance triangular finite elements are obtained in our example but the original model would be composed of 'edge' finite elements. We also can't obtain the original model when some equations have been eliminated by taking into account boundary conditions. However, this method allows to split examples given in global matrix form into sequence of elementary matrices convenient for evaluating reordering techniques for an element-by-element frontal solver.