

Working Paper Series

HEARIN CENTER
FOR
ENTERPRISE SCIENCE

HCES-03-00

**Reducing the Bandwidth of a Sparse Matrix with
Tabu Search**

by
**Rafael Martí
Vicente Campos
Manuel Laguna
Fred Glover**



The University of Mississippi

Director, Keith Womer
School of Business Administration
The University of Mississippi
Post Office Box 1848
University, MS 38677-1848
(662) 915-5820
<http://hces.bus.olemiss.edu>

Reducing the Bandwidth of a Sparse Matrix with Tabu Search

Rafael Martí^a, Manuel Laguna^b, Fred Glover^c and Vicente Campos^a

a Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot-Valencia, Spain. Vicente.Campos@uv.es and Rafael.Marti@uv.es

b Graduate School of Business and Administration, Campus Box 419, University of Colorado, Boulder, CO 80309-0419, USA. Manuel.Laguna@Colorado.edu

c Hearin Center for Enterprise Science, University of Mississippi, University, MS 38677, USA. fglover@bus.olemiss.edu

Latest Revision: February 21, 2000

Abstract — The bandwidth of a matrix $A = \{a_{ij}\}$ is defined as the maximum absolute difference between i and j for which $a_{ij} \neq 0$. The problem of reducing the bandwidth of a matrix consists of finding a permutation of the rows and columns that keeps the nonzero elements in a band that is as close as possible to the main diagonal of the matrix. This NP-complete problem can also be formulated as a labeling of vertices on a graph, where edges are the nonzero elements of the corresponding symmetrical matrix. Many bandwidth reduction algorithms have been developed since the 1960s and applied to structural engineering, fluid dynamics and network analysis. For the most part, these procedures do not incorporate metaheuristic elements, which is one of the main goals of our current development. Another equally important goal is to design and test a special type of candidate list strategy to be embedded in a tabu search procedure for the bandwidth reduction problem. This candidate list strategy accelerates the selection of a move in the neighborhood of the current solution in any given iteration. Our extensive experimentation shows that the proposed procedure outperforms the best-known algorithms in terms of solution quality consuming a reasonable computational effort.

Keywords: Metaheuristics, tabu search, matrix bandwidth

1. Introduction

Let $G(V, E)$ be a graph with vertex set V and edge set E . Assume $|V| = n$. A labeling f of G assigns the integers $\{1, 2, \dots, n\}$ to the vertices of G . Let $f(v)$ be the label of vertex v , where each vertex has a different label. The bandwidth of a vertex v , $B_f(v)$, is the maximum of the differences between $f(v)$ and the labels of its adjacent vertices. That is:

$$B_f(v) = \max\{|f(v) - f(u)| : u \in N(v)\}$$

where $N(v)$ is the set of vertices adjacent to v . The bandwidth of a graph G with respect to a labeling f is then:

$$B_f(G) = \max\{B_f(v) : v \in V\}$$

Then, the bandwidth of a graph is $B(G)$, the minimum $B_f(G)$ value over all possible labelings f . The bandwidth reduction problem consists of finding a labeling f that minimizes $B_f(G)$. Note that a labeling is simply a renumbering of the vertices.

If we let $A = \{a_{ij}\}$ be the incidence matrix of a graph V (i.e., $a_{ij} \neq 0$ if $(i, j) \in E$), then the bandwidth reduction problem consists of finding a permutation of the rows and the columns that keeps all the non-zero elements of A in a band that is as close as possible to the main diagonal. This interpretation of the problem arises in applications to solve nonsingular systems of linear algebraic equations of the form $Ax = b$. The preprocessing of A to reduce its bandwidth results in substantial savings on the computational effort associated with solving the system of equations. Other applications include problems in finite element methods for approximating solutions of partial differential equations, large-scale power transmissions systems, circuit design, hypertext layout, chemical kinetics and numerical geophysics.

We do not begin with a detailed review of the numerous algorithms that have been developed for the bandwidth reduction problem, because such procedures are well documented in the literature and our goal is to compare our tabu search implementation with the best available solution methods. In the next section, we briefly discuss the two algorithms that we will use for comparison purposes. We then provide details of our implementation, followed by our extensive computational experimentation. Finally, we outline our conclusions and directions for future research.

2. Relevant Existing Procedures

Gibbs, Poole and Stockmeyer (1976) developed a heuristic for the bandwidth reduction problem, referred to as GPS. Their computational testing was performed on 19 cases with the order of the matrices ranging from 68 to 918. These matrices were taken from the solution of various differential equations and variational problems in structural engineering applications of the finite element method. The context of these applications included aircraft structures, liquid nitrogen gas tanks, propel blades and submarines. Their comparisons were made against the reverse Cuthill-McKee algorithm (Cuthill and

McKee, 1969). The experiments show that on average GPS gives a bandwidth that is slightly smaller than that of the reverse Cuthill-McKee procedure, although in only 9 out of the 19 cases GPS is actually better than Cuthill-McKee. It is reasonable to conclude that these two methods yield similar results in terms of solution quality. However, GPS is considerably faster, averaging speed that is about 8 times faster than the reverse Cuthill-McKee procedure.

GPS is a procedure that operates on a level structure L . A level structure is a partition of V into sets L_1, L_2, \dots, L_k , called levels, with the following characteristics:

- Vertices adjacent to a vertex in level L_1 are in either L_1 or L_2
- Vertices adjacent to a vertex in L_k are in either L_k or L_{k-1}
- Vertices adjacent to a vertex in L_i (for $1 < i < k$) are in either L_{i-1}, L_i or L_{i+1}

The procedure consists of the following three phases:

1. *Finding endpoints of a pseudo-diameter*: This procedure attempts to find a pair of vertices that are at nearly maximal distance apart. (Note that the diameter of G is the shortest path connecting two vertices of maximal distance apart.) The resulting endpoints tend to generate level structures of small width and therefore maximal depth.
2. *Minimizing level width*: In phase 1, the algorithm constructs level structures rooted at both endpoints. Phase 2 combines these two level structures into a new structure whose width is usually less than that of either of the original ones.
3. *Numbering*: The numbering procedure assigns, level-by-level, consecutive positive integers to the vertices V of G . The numbering starts at the endpoint with lesser degree. The procedure uses forward numbering but the numbering is reversed under special circumstances.

The reported times for running this procedure range from 0.6 to 19.32 seconds on an IBM 360 model 50. This compares favorably to the times ranging from 3.08 to 183.68 seconds reported for the reverse Cuthill-McKee method.

Regarding metaheuristic methods for the bandwidth reduction problem, Dueck and Jeffs (1996) describe an implementation of simulated annealing. The search starts from a labeling f generated at random. The search then moves from one labeling to another by exchanging the labels of two vertices. That is, this SA implementation is based on swapping a pair of labels or also known as swap moves. In any given iteration, a move is generated by randomly choosing two vertices to exchange labels and the move value is calculated. Let f be the current labeling and f' the labeling after applying a swap move. Then Dueck and Jeffs define the move value as:

$$move_value = B_{f'}(G) - B_f(G)$$

If $move_value$ is less than or equal to zero then the move is accepted and immediately executed. If the $move_value$ is strictly positive (raising the bandwidth of the current labeling) then the move is accepted with a probability that depends on the current temperature and is calculated with the negative exponential function used in standard simulated annealing implementations. There are 5 parameters in Dueck and Jeffs' implementation: temperature ($temp = 1.0$), cooling rate ($cool_rate = 0.95$), maximum

number of accepted moves at each temperature ($max_moves = 4 * |E|$), maximum number of attempted moves at each temperature ($max_attempted_moves = 80 * max_moves$), and the maximum number of consecutive iterations in which the procedure accepts less than max_moves moves ($max_frozen = 50$). The authors justify their choice for the parameter values shown above based on general simulated annealing guidelines, their own experiences and experimental tuning in the current context.

Dueck and Jeffs tested their SA implementation on 18 graphs with number of edges ranging from 13 to 255. This set includes 8 different types of graphs. The experimentation shows that this SA implementation is inferior to GPS on “grid”, “path”, “circle”, “windmill” and “st” graphs. However, SA outperforms GPS in terms of solution quality on ternary trees, binary trees and random graphs. It is important to point out that even though SA finds better labelings than GPS in 11 out of 18 graphs, it does so by employing up to 2000 times longer. The authors argue that although the computational effort of their implementation limits its applicability, it can be used as a tool to evaluate the performance of other procedures. It is precisely in this way in which we make use of their solution method.

3. Tabu Search Implementation

One of the main characteristics of the bandwidth reduction problem is that there may be many labelings f with the same $B_f(G)$ value. This means that for a given $B_f(G)$ value there may be multiple *critical* vertices v for which $B_f(v) = B_f(G)$. Consequently, changing a labeling in order to reduce the $B_f(v)$ value of a critical vertex does not imply that the $B_f(G)$ value will also decrease. We take into consideration this important feature of the problem to develop mechanisms (including an unconventional definition of the move value) that are effective in searching for good labelings.

As in the case of the SA implementation described above, we define a move as the swap of labels of a pair of vertices. That is, the operator $move(v, u)$ assigns the label $f(u)$ to vertex v and the label $f(v)$ to vertex u . Since our focus is to change the labels in order to reduce the current value of $B_f(G)$, we construct a *candidate list* of moves based on a set $C(f)$ of critical and near-critical vertices. A near-critical vertex v is one for which $B_f(v) \geq \alpha * B_f(G)$ and $1 > \alpha > 0$. Near-critical vertices do not determine the value of the objective function $B_f(G)$ in the current labeling, but they are considered likely to do so in subsequent iterations. In other words, if a move is able to eliminate a critical vertex, it is likely that a near-critical vertex will become critical in subsequent iterations. The set that consists of both critical and near-critical vertices can be defined as:

$$C(f) = \{v : B_f(v) \geq \alpha * B_f(G)\}.$$

When this set is constructed, the updated value of $B_f(G)$ is used. This value, however, is not updated after the execution of a move during the examination of the vertices in $C(f)$, because the updating is a computationally expensive calculation. In particular, the calculation of $B_f(G)$ requires the examination of all the vertices in the graph. The notion of not updating key values (e.g., move values) after every iteration is based on

the *elite candidate list* suggested in Glover and Laguna (1997). The design considers that it is not absolutely necessary to update the value of the moves in a candidate list after an iteration is completed (i.e., the selected move is executed) because most of these move values either remain the same or their relative merit remains almost unchanged. The application of this strategy is particularly useful when the updating of the move values is computationally expensive, as in our context.

In order to construct a candidate list of moves based on the vertices in $C(f)$, we must find a suitable swapping vertex u in $N(v)$. We define the following two quantities for a vertex v and a labeling f :

$$\begin{aligned} \max(v) &= \max\{f(u) : u \in N(v)\} \\ \min(v) &= \min\{f(u) : u \in N(v)\} \end{aligned}$$

Since the best label for v in the current labeling f is:

$$\text{mid}(v) = \frac{\max(v) + \min(v)}{2}$$

then the set of suitable swapping vertices for v is defined as:

$$N'(v) = \{u : |\text{mid}(v) - f(u)| < |\text{mid}(v) - f(v)|\},$$

which considers all vertices u with labels $f(u)$ that are “closer” to $\text{mid}(v)$ than $f(v)$. If $N'(v) = \emptyset$, then $B_f(v)$ cannot be reduced by simply changing the current label of v . The candidate list of moves associated with a vertex $v \in C(f)$ is given by:

$$CL(v) = \{\text{move}(v, u) : u \in N'(v)\}.$$

Note that when the label for vertex v changes, one or more of the $B_f(u)$ values for $u \in N(v)$ could change. However, the bandwidth of the adjacent vertices will remain the same as long as the new label for vertex v , $f'(v)$, is within the range $[\min(u), \max(u)]$ for all $u \in N(v)$. Hence, when $\min(u) \leq f'(v) \leq \max(u)$, it is possible to reduce the bandwidth of v without increasing the bandwidth of its adjacent vertices. In practice, this criterion can be relaxed. For example, consider the case when $f'(v) > \max(u)$. $B_f(u)$ increases only if $f'(v) - f(u) > f(u) - \min(u)$. Similarly, if $f'(v) < \min(u)$, then $B_f(u)$ increases only if $f(u) - f'(v) > \max(u) - f(u)$.

Enforcing a criterion that forbids moves, associated with critical vertices, when either of aforementioned conditions is violated makes the search unnecessarily inflexible. Note that if the bandwidth of a vertex u that is adjacent to a vertex $v \in C(f)$ increases marginally with respect to the bandwidth of the graph, then we can consider that the move does not affect the bandwidth of u . In fact, we only consider that the bandwidth of vertex u increases when

$$|f'(v) - f(u)| > B_f(u) \quad | \quad \text{and} \quad |f'(v) - f(u)| > \beta * B_f(G).$$

This criterion is applied to both vertices involved in a swap. That is, if the $move(v,u)$ is being considered, then the criterion is applied to determine the change in the bandwidth of the vertices adjacent to v and those adjacent to u . Note that in such exchange $f'(v) = f(u)$ and $f'(u) = f(v)$.

These concepts are best understood if we illustrate them with an example. Consider the partial graph in Figure 1. The graph consists of 7 vertices with the current labeling given by the numbers shown next to each vertex. Consider the following values associated with vertices u , v and w :

$$\begin{array}{lll}
 f(u) = 11 & f(v) = 9 & f(w) = 8 \\
 B_f(u) = 6 & B_f(v) = 7 & B_f(w) = 1 \\
 \min(u) = 9 & \min(v) = 2 & \min(w) = 7 \\
 \max(u) = 17 & \max(v) = 10 & \max(w) = 9 \\
 & \text{mid}(v) = \frac{10+2}{2} = 6 &
 \end{array}$$

Suppose that it is possible to swap the current level for v from 9 to 6, where the value of 6 corresponds to the current label of a vertex that is not shown in Figure 1. According to our previous notation, we have that while $f(v) = 9$, $f'(v) = 6$. This means that although $f'(v) < \min(u)$, $B_f(u)$ does not change because $f(u) - f'(v) < \max(u) - f(u)$, that is $11 - 6 < 17 - 11$ or $5 < 6$. On the other hand, $B_f(w)$ increases as a result of executing the move. The increase, however, is of one unit since the new bandwidth value for w is $|f'(v) - f(w)| = 2$, which cannot cause an increase in $B_f(G)$ because $B_f(v)$ is now 6 and this value represents a lower bound for $B_f(G)$. This partial analysis (which disregards the effect in the swapping vertex u) shows that the move is favorable. The bandwidth associated with vertex v is reduced from 7 to 6, while the bandwidth for vertex u remains the same and the bandwidth for vertex w increases from 1 to 2.

One of the key elements in heuristic search is the definition of the value of a move. The most common practice is to define the move value as the change in the objective function value, which is the way it was defined in the simulated annealing implementation described in the previous section. Nevertheless, in the context of the bandwidth reduction problem, the change in the objective function value provides little or no information during the search whenever the current labeling has more than one critical vertex. Additionally, the calculation of $B_f(G)$ is computationally expensive. As a result, we have defined the value of a $move(v,u)$ as the number of vertices adjacent to v or u whose bandwidth increases due to the move. (Recall that the bandwidth increase for adjacent vertices is controlled by the parameter β as specified above.)

Our basic tabu search implementation consists of a short-term memory design in which the identity of a vertex whose label has been changed is the attribute used to impose a tabu restriction. Specifically, after a $move(v,u)$ is executed, the labels of vertices v and u are not allowed to change until the tabu tenure expires. We employ a one-dimensional array $tabu(v)$, initially set to zero, to store the iteration number when vertex v loses its tabu status. That is, if vertex v changes labels at iteration $iter$, then

$tabu(v) = iter + tenure$, where $tenure$ is the number of iterations that vertex v is not allowed to change labels. Then, $move(v, u)$ is tabu if:

$$tabu(v) > iter \text{ or } tabu(u) > iter$$

Note that although we use the same $tenure$ value for both vertices, an interesting variant is to use a different $tenure$ value for vertex v than for vertex u . In such design, the $tenure$ value for vertex v should be larger than the $tenure$ value for vertex u , because v is a critical or near-critical vertex and u is a vertex that simply happens to have a label that makes the move attractive. In our experiments, however, we determined that the effect of using different $tenure$ values does not justify the increase in complexity related to calibrating an additional search parameter.

Figure 2 shows a pseudo-code of our implementation. The basic implementation starts from a random initial labeling or from a labeling constructed using the restarting mechanism explained below, which is based on longer-term diversification. In step 2, we initialize the counter of iterations without improvement ($iterw$). The procedure runs until $maxiter$ iterations without improvement. Step 3 builds $C^*(f)$ which consists of the near-critical vertices whose tabu status is inactive. The inner while-loop (steps 5-9) searches for and executes moves until the updated list of critical or near-critical vertices is empty. In step 6, the best vertex u to swap labels with vertex v is found as follows. Since the best label for v is $mid(v)$, then the search for the best swapping vertex attempts to find a vertex u in $N'(v)$ whose tabu status is inactive. The search order is $mid(v)$, $mid(v)+1$, $mid(v)-1$, $mid(v)+2$, ..., $mid(v)-f(v)$, as long as the range of labels stay within the allowed range from 1 to n . The search uses a first-improving strategy that selects and executes the first non-tabu $move(v, u)$ with a move value of zero. If a move with a value of zero is not found, then the one with the minimum value is selected.

The basic procedure shown in Figure 2 does not incorporate sophisticated elements of the tabu search framework. This basic procedure is meant to yield reasonably good solutions at a competitive speed. In order to improve solution quality, we added a longer-term search strategy to the basic implementation.

3.1 Longer-term Diversification

The goal of incorporating a longer-term diversification strategy to our basic tabu search implementation is to find labelings of higher quality even if this means investing additional computational effort. Diversification is the notion of expanding the search to unexplored regions in the solution space. This expansion consists of visiting solutions that have not been previously examined. Diversification strategies are generally based on either encouraging the incorporation of new elements or discouraging often examined elements. In particular, we use a frequency count $freq(v, f(v))$ to record the number of times vertex v is labeled with label $f(v)$.

We use this frequency count to re-start the search with a new labeling f . The new labeling is built considering that the best label for vertex v should be “close” to the average label of the vertices in $N(v)$. Since we want to diversify, we select the label $f(v)$

in such a way that the frequency value $freq(v, f(v))$ is small relative to alternative labels. With this in mind, we implemented a function that given a vertex v and a label $f(v)$ it returns the label $leastfreq(v, f(v))$, in the interval $[f(v) - 5, f(v) + 5]$, that has been assigned to v the least number of times during the search.

The construction procedure based on frequency information starts with the random selection of a vertex v among those of minimal degree. We then assign the label $leastfreq(v, r)$ to v , where r is a random integer number between 1 and n . At each construction step after the first, the set of candidate vertices consists of those unlabeled vertices that are adjacent to vertices that have been already labeled. A vertex v in the set of candidates is randomly selected. This random selection was experimentally shown to outperform other pseudo-random criteria such as one based on the degree of the vertices in the candidate set. The selected vertex v is assigned the label $leastfreq(v, r)$, where r is the average label of the vertices in $N(v)$ that have already been labeled.

This restarting procedure is used to experiment with two versions of our implementation. The first version is a simple run of the basic TS implementation of Figure 2. In the second version, we execute the basic TS procedure $maxstart$ times using the diversification strategy discussed in this section.

4. Computational Experiments

For our computational testing, we implemented, in C, the GPS procedure of Gibbs, Poole and Stockmeyer (1976), the simulated annealing (SA) of Dueck and Jeffs (1995) and our two variants of the TS procedure: 1) TS($maxiter$), the basic TS implementation with a stopping criterion of $maxiter$ iterations and 2) TS($maxiter, maxstart$), the TS implementation with the diversification mechanism for restarting $maxiter$ times. The codes were compiled with Microsoft Visual C++ 5.0, optimizing for maximum speed. The experiments with 126 instances were run on a Pentium II at 350 MHz. We performed three sets of experiments with the following goals:

1. A preliminary experimentation with 10 instances to find the best values for the key search parameters α , β and $tenure$.
2. An experiment with the entire set of 126 instances to compare the performance of GPS, SA, TS($maxiter$) and TS($maxiter, maxstart$).
3. An experiment to find the best-known solutions to all the instances by using GPS to generate the initial solution for a TS($maxiter, maxstart$) run.
4. An experiment to compare the performance of TS($maxiter, maxstart$) when $maxiter \gg maxstart$ and when $maxiter \ll maxstart$.

The preliminary experimentation was performed on 10 representative problem instances with the goal of finding appropriate values for the three key search parameters. We tested values for α and β in the range $[0.2, 0.8]$, and for $tenure$ in the range $[3, 15]$. The results of this test are shown in Table 1.

Table 1. Preliminary experimentation.

α	β	<i>tenure</i>	$B_f(G)$	<i>CPU seconds</i>
0.5	0.5	5	43.89	15.02
0.8	0.8	5	46.00	5.93
0.8	0.2	5	75.44	2.33
0.2	0.2	5	71.89	22.69
0.2	0.8	5	35.00	7.79
0.8	0.8	10	49.00	4.38
0.8	0.8	15	82.22	4.10
0.2	0.8	3	35.67	7.24
0.2	0.8	7	37.00	7.15

For each combination of parameter values, Table 1 shows the average bandwidth and the average CPU seconds over 10 problem instances. These results show that the best average bandwidth value at a competitive speed is obtained when $\alpha = 0.2$, $\beta = 0.8$ and *tenure* = 5. Hence, we use these values to perform the rest of our experimentation.

In our second set of experiments, we use 126 problem instances to compare the performance of our proposed procedure with the best heuristic reported in the literature (GPS) and a metaheuristic based on simulated annealing (SA). Instead of limiting our experimentation to less than 20 problem instances (as done in the articles referenced in section 2), we have compiled 126 instances from the *Harwell-Boeing Sparse Matrix Collection* (<http://math.nist.gov/MatrixMarket/data/Harwell-Boeing>). This collection consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large test cases arising in applications. Iain Duff, Roger Grimes, and John Lewis (1992) originally developed this collection. Table 2 shows, for each method, the average bandwidth over the instances in each set along with the average CPU seconds. This table also shows the average deviation from the best-known solutions. The best-known solutions are the best solutions found by applying all the procedures to the same problem instance.

Table 2. Performance comparison according to problem size.

37 instances with $n = 30, \dots, 199$							
	GPS	SA	SA2	TS(100)	TS(200)	TS(100,10)	TS(200,20)
$B_f(G)$	37.73	36.59	57.54	30.49	30.35	27.68	27.57
Deviation	33%	37%	220%	19%	19%	6%	5%
CPU sec.	0.02	1434.97	22.27	0.98	1.69	24.82	74.32
89 instances with $n = 200, \dots, 1000$							
	GPS	SA	SA2	TS(100)	TS(200)	TS(100,10)	TS(200,20)
$B_f(G)$	161.07	439.45	384.98	111.04	109.64	101.79	100.31
Deviation	43%	942%	866%	32%	30%	8%	5%
CPU sec.	0.71	1800.00	581.53	32.76	54.89	711.42	3151.3

Note that there are two versions of simulated annealing (SA and SA2) in Table 2. The version denoted with SA is the one originally proposed by Dieck and Jeffs (1995). This version employs a set of parameter values that result in a cooling profile that is so slow that for large instances the method does not finish. SA2 uses a set of parameter values

that speeds the cooling and allows the search to finish before the cutoff time of 30 minutes. (Note that for the large instances, SA reaches the cutoff limit every time.) Regardless of the parameter settings, SA yields inferior solutions compared to TS. The basic implementation of TS is superior to GPS in terms of solution quality and competitive in terms of speed in the small instances. In the large instances, TS cannot compete in terms of time with GPS. The TS version with restarting is robust in terms of solution quality, with an average deviation from the best-known solutions of 5% for the longer runs. These experiments show that GPS is capable of generating good solutions at a speed that is hard to match by a metaheuristic. Nonetheless, tabu search keeps finding high-quality solutions when the search is allowed to go beyond a handful of CPU seconds. It should also be mentioned that GPS performs best on instances with structured graphs. That is, the more structure in the graph the better the performance of GPS. While GPS is designed to exploit the underlying structure of a graph, our TS implementation does not take advantage of the particular characteristics of a graph.

Since GPS obtains good solutions fast, a viable strategy for finding solutions of the highest quality is to first run GPS and then use the resulting solution as the starting point for tabu search. We use this strategy and report our findings in Table 3. In this experiment, we are concerned only with the quality of the resulting solutions and not with the time to find them, which is determined by the values of *maxiter* and *maxstart* because GPS generates the initial solutions almost instantaneously. Table 3 shows the average bandwidth found with GPS, TS and the GPS-TS combination along with the relative deviation of their corresponding solutions. The results in Table 3 do not categorically determine a winning strategy for finding the best solution to a given problem instance. For instances with less than 200 vertices, the GPS-TS combination yields a larger average bandwidth than the TS procedure that starts from a random labeling. For the larger instances, the GPS-TS yields the best solutions in terms of both the average bandwidth and the average deviation from the best solutions found.

This experiment indicates that while GPS provides good starting points for TS, the effect of starting from a good solution is diluted by the restarting mechanism implemented in TS. We also attempted to run TS(*maxiter*) with larger values of *maxiter* in combination with GPS and were not able to improve upon the solutions found when the restarting procedure is used.

Table 3. Results of using GPS to generate a starting solution for TS.

	37 instances with $n = 30, \dots, 199$		
	GPS	TS(100,10)	GPS-TS
$B_f(G)$	37.73	27.68	57.54
Deviation	31.05%	4.61%	1.34%
CPU sec.	0.02	24.82	22.27
	89 instances with $n = 200, \dots, 1000$		
	GPS	TS(100,10)	GPS-TS
$B_f(G)$	161.07	101.79	100.5
Deviation	42.25%	6.95%	0.70%
CPU sec.	0.71	711.42	721.02

In our last experiment, we focus on our TS(*maxiter,maxstart*) implementation. Our research question centers on whether it is better to search more from a given point or to search less from multiple starting points. To answer this, we compare the results obtained from running TS(100,10) with those obtained from running TS(10,100). The results of these experiments are summarized in Table 4.

Table 4. Comparing TS(100,10) with TS(10,100).

37 instances with $n = 30, \dots, 199$			
	GPS	TS(100,10)	TS(10,100)
$B_f(G)$	37.73	27.68	28.30
Deviation	31.05%	4.61%	4.73%
CPU sec.	0.02	24.82	27.05
89 instances with $n = 200, \dots, 1000$			
	GPS	TS(100,10)	TS(10,100)
$B_f(G)$	161.07	101.79	104.34
Deviation	40.54%	5.62%	7.67%
CPU sec.	0.71	711.42	1152.38

The results strongly suggest that our TS implementation is capable of finding better solutions when allowed to run longer from a few starting points instead of using the alternative strategy. While TS(10,100) competes with TS(100,10) in the 37 instances with less than 200 vertices, TS(100,10) is clearly superior in the larger instances.

5. Conclusions

In this paper, we report our research on the use of a candidate list strategy that allows us to cope with a computationally expensive objective function evaluation within a tabu search procedure. Our procedure selects moves from a candidate list of moves whose move values are not updated after every iteration. The list follows the TS principle that the values of a set of elite moves do not drastically change from one iteration to the next and therefore it is not necessary to update them after the execution of every move. In addition to the application of this candidate list strategy, our procedure employs an unconventional definition of move value, which is not based on the change of the objective function value to direct the search. In this way, our move value definition conveys information that is not available when the change in the objective function value is calculated.

The results of our computational experiments reveal that the strategies implemented within a relatively simple tabu search procedure are capable of outperforming a competing metaheuristic, such as simulated annealing. In particular, the restarting procedure based on frequency information confirms the benefit of encouraging the diversification of the search. The additional diversification results in a solution method capable of finding high-quality solutions to the bandwidth reduction problem, although we must admit that the procedure consumes considerably more computer time than a constructive heuristic such as GPS.

An interesting question related to search diversification deals with the balance between restarting and search-depth (i.e., the time spent searching from a single starting point). Some metaheuristics, such as GRASP (Feo and Resende 1995), launch limited local searches from numerous constructions (i.e., starting points). Typically in tabu search, the search starts from one initial point and if a restarting procedure is also part of the implementation, it is invoked only a limited number of times. In our last set of experiments in this paper, we tested both alternatives and concluded that it was better to invest the time searching from a few starting points than restarting the search more often. Although we cannot draw a general conclusion from these experiments, our experience in the current context and in previous projects indicates that tabu search procedures need to reach a critical depth search to be effective. If this search depth is not reached,

the effectiveness of the method is severely compromised. So we can conclude that while our TS(100,10) runs effectively reached the critical search-depth, the TS(10,100) runs did not.

References

Cuthill, E. and J. McKee (1969) "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. ACM National Conference*, Association for Computing Machinery, New York, pp. 157-172.

Dueck, G. H. and J. Jeffs (1995) "A Heuristic Bandwidth Reduction Algorithm," *J. of Combinatorial Math. And Comp.*, vol. 18, pp. 97-108.

Duff, I. S., R. G. Grimes and J. G. Lewis (1992) "Users' Guide for the Harwell-Boeing Sparse Matrix Collection," Research and Technology Division, Boeing Computer Services.

Feo, T. and M. G. C. Resende (1995) "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 2, pp. 1-27.

Gibbs, N. E., W. G. Poole and P. K. Stockmeyer (1976) "An Algorithm for Reducing the Bandwidth and Profile of Sparse Matrix," *SIAM Journal of Numerical Analysis*, vol. 13, no. 2, pp. 236-250.

Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.