Adaptive Memory Programming for Matrix Bandwidth Minimization*

Vicente Campos, Estefanía Piñana and Rafael Martí+

Departamento de Estadística e Investigación Operativa,
Facultad de Matemáticas, Universitat de València
Dr. Moliner 50, 46100 Burjassot (València), Spain.

Latest version: January 27, 2006

Abstract

In this paper we explore the influence of adaptive memory in the performance of heuristic methods when solving a hard combinatorial optimization problem. Specifically, we tackle the adaptation of tabu search and scatter search to the bandwidth minimization problem. It consists of finding a permutation of the rows and columns of a given matrix which keeps the non-zero elements in a band that is as close as possible to the main diagonal. This is a classic problem, introduced in the late sixties, that also has a well-known formulation in terms of graphs. Different exact and heuristic approaches have been proposed for the bandwidth problem. Our contribution consists of two new algorithms, one based on the tabu search methodology and the other based on the scatter search framework. We also present a hybrid method combining both for improved outcomes. Extensive computational testing shows the influence of the different elements in heuristic search, such as neighbourhood definition, local search, combination methods and the use of memory. We compare our proposals with the most recent and advanced methods for this problem, concluding that our new methods can compete with them in speed and running time.

Key Words: Heuristic Search, Memory Programming, Tabu Search, Scatter Search.

* Research partially supported by the *Ministerio de Educación y Ciencia* (ref. TIC2003-C05-01) and by the *Agencia Valenciana de Ciencia y Tecnologia* (ref. GRUPOS03/189).
+ Corresponding author: Rafael.Marti@uv.es

1. Introduction

When solving a combinatorial optimization problem, the degree to which memory is exploited varies according to the type of procedure. Virtually all metaheuristics induce a pattern whose present state depends on the sequence of past states and therefore incorporate an implicit form of memory. Such a memory, however, does not necessarily take the form of an intelligent memory construction since it does not use explicit designs for recording past elements but uses this information in a strategic way. Based on the explicit use of memory, algorithms can be classified as *memory-based* or *memory-less* methods. Approaches such as Tabu Search (TS) or Scatter Search (SS) are memory oriented methods in which records about past choices and decisions determine future strategies. On the other hand, methods such as Simulated Annealing (SA) or Genetic Algorithms (GA) do not incorporate the explicit use of memory structures and are based on other strategies, mostly relying on randomization for decision making in the search process.

TS (Glover and Laguna, 1997) incorporates *adaptive memory* which allows the implementation of procedures that are capable of searching the solution space in an efficient way. The memory used in tabu search is both *explicit* and *attributive.* Explicit memory records complete solutions, typically consisting of elite solutions visited during the search, while attributive memory is mainly used for guiding purposes. This latter type of memory records information about solution attributes that change in moving from one solution to another. *Short* and *long-term* memory structures are responsible for the specific composition of the solution neighborhood. In other words, the neighborhood in a given iteration is the result of maintaining a selective history of the states encountered during the search. In this sense TS can be viewed as a dynamic neighborhood method.

SS (Laguna and Martí, 2003) maintains a set of solutions throughout the search (the Reference Set) and combines these solutions to generate new ones. GA operate on a population of solutions and the "survivable of the fittest" philosophy translates into an implicit use of memory. In SS however, from one iteration to the next, the method keeps track of the subsets of reference solutions that have already been combined. When new solutions enter the reference set, the method generates only those subsets that are admissible for combination in the current iteration, using a memory structure that allows it to identify the subsets that contain new reference solutions. There is no equivalent use of memory in genetic algorithms, since they select solutions for combination purposes using a random scheme.

TS and SS have a common history as their basic principles were suggested by Glover (1977). They are probably the metaheuristic procedures that employ memory in the most strategic and direct way, constituting the core of what has been coined as Adaptive Memory Programming in recent years. In this paper we study the adaptation of both methodologies to solve the matrix bandwidth minimization problem.

Let $G=(V,E)$ be a graph with vertex set V ($|V|=n$) and edge set E ($|E|=m$). A labeling or linear layout f of G assigns the integers 1, 2, …, 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:

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

The bandwidth $B(G)$ of graph G is thus the minimum $B_f(G)$ value over all possible labelings f. In other words, the matrix bandwidth minimization problem consists of finding a labeling f that minimizes $B_f(G)$. If we consider the incidence matrix of graph G, the problem can be formulated in terms of matrices as finding a permutation of the rows and the columns of this matrix that keeps all the non-zero elements in a band that is as close as possible to the main diagonal. For that reason this problem is known as the Matrix Bandwidth Minimization Problem (MBMP).

The main application of this problem is to solve non-singular systems of linear algebraic equations. The preprocessing of the coefficient matrix to reduce its bandwidth results in substantial savings in the computational effort associated with solving the system of equations. The context of these applications includes aircraft structures, liquid nitrogen gas tanks, propel blades and submarines. The MBMP is known to be NP-hard (Papadimitriou 1976).

For many years researchers were only interested in designing relatively simple heuristic procedures and sacrificed solution quality for speed. This is the case of the Reverse Method (Cuthill and McKee, 1969) and the GPS procedure (Gibbs et al. 1976). These two methods yield similar results in terms of solution quality; however, GPS is considerably faster, with an average speed that is about 8 times faster than the reverse Cuthill-McKee procedure. Recently, metaheuristics have been adapted to this problem. An SA procedure was introduced by Dueck and Jeffs (1995), which is based on an insertion mechanism and does not take advantage of the graph structure as the GPS method does. Martí et al. (2001) propose a TS method for this problem, which is likewise based on swap moves that exchange the labels of a pair of vertices; however, it incorporates memory structures that prove to be remarkably effective. Piñana et al. (2004) proposed a GRASP method for the MBMP. The constructive step is based on GPS and the local search is based on exchanges. The GRASP method is coupled with a path relinking phase for improved outcomes. This algorithm clearly outperforms all the previous heuristic approaches. More recently, Lim et al. (2005) introduce the node-shift heuristic which computes the desired label of each vertex according to the label of its adjacent vertices and then orders all the vertices in the graph with respect to these desired labels, finally all the vertices are re-labeled following this ordering. This innovative method is repeated until no vertex changes its label and it is coupled with a local search hill climbing. These authors also propose a GA that generates the solutions in the initial population with a level structure procedure (as the GPS does) and implements a classic mid-point crossover as a combination operator. Rodríguez-Tello et al. (2006) propose a simulated annealing method based on a new neighborhood definition. Instead of swapping the labels of two vertices, they introduce a more elaborated move definition that leads to an efficient search as shown in their computational results. Finally, Martí et al. (2006) introduce an exact branch and bound method to compute the optimal solution for medium size instances, as well as a lower bound for large instances.

This paper is organized as follows. The next section discusses our new TS adaptation to this problem, and describes its differences with the previous TS approach mentioned above. Section 3 introduces a SS algorithm for this problem as well as a hybrid method combining it with the TS approach. Section 4 presents extensive computational experimentation with the *Harwell-Boeing Sparse Matrix Collection* which has been used as a benchmark for comparison in most of the papers above. We compare our proposals with the recently developed methods as well as with the lower bounds. Moreover, our extensive computational testing shows the influence of the different elements in heuristic search, such as neighbourhood definition, local search, combination methods and the use of memory. As far as we know, this is the first time that the value of heuristic solutions is compared with a lower bound for this problem. Finally, we outline our conclusions.

2. Tabu Search

Although exchanges are used as the primary mechanism to move from one solution to another in our implementation, we have also considered two additional moves to escape from local optima and perform a more robust search. In this section we first describe these three moves and then explain the other elements, such as the memory structures, of our TS method.

2.1 Short-Term Memory

Move 1: exchanges
The operator *move*1*(u,v)* assigns the label *f(u)* to vertex *v* and the label *f(v)* to vertex *u*.

Move 2: double exchanges
The operator *move*2*(u,v,w)* first performs *move*1*(u,v)* and then performs *move*1*(v,w)*. Thus, if *u, v* and *w* initially have labels *f(u), f(v)* and *f(w)* respectively, after performing *move*1*(u,v)* the labels are *f(v), f(u)* and *f(w)* for *u, v* and *w* respectively. Then when we apply *move*1*(v,w),* the labels are *f(v), f(w)* and *f(u)* for *u, v* and *w* respectively.

Move 3: multiple shifts

The operator *move3(u,v)* assigns the label *f(v)* to vertex *u*. For the other assignments in this move we distinguish two cases. If $f(u) < f(v)$, let $u_1, u_2, \ldots, u_k, u_{k+1}=v$ be the intermediate vertices with consecutive labels $f(u)+1, f(u)+2, \ldots f(u)+k, f(u)+k+1$ respectively, then this move assigns label *f(u)* to u_1, *f(u)*+1 to $u_2, \ldots f(u) + k-1$ to u_k, and $f(u)+k$ to *v*. Alternatively, if $f(u) > f(v)$, let $u_1, u_2, \ldots, u_k, u_{k+1}=v$ be the vertices with labels $f(u)-1, f(u)-2, \ldots f(u)-k, f(u)-k-1$ respectively, then this move assigns label *f(u)* to u_1, *f(u)*-1 to $u_2, \ldots f(u) - k+1$ to u_k, and $f(u)-k$ to *v*.

Glover and Laguna (1997) introduced *compound moves*, often called *variable depth methods*, constructed from a series of simpler components. As is well-known, one of the pioneering contributions to this kind of moves was Lin and Kernighan (1973). Within the class of variable depth procedures, a special subclass called *ejection chain procedures* has recently proved useful. An ejection chain is an embedded neighborhood construction that compounds the neighborhoods of simple moves to create more complex and powerful moves. It is initiated by selecting a set of elements to undergo a change of state (e.g. to occupy new positions or receive new values). The result of this change leads to identifying a collection of other sets, with the property that the elements of at least one must be "ejected from" their current states. State-change steps and ejection steps typically alternate, and the options for each depend on the cumulative effect of previous steps (usually, but not necessarily, being influenced by the immediately preceding step). In some cases, a cascading sequence of operations may be triggered, representing a domino effect.

Since our focus is to change the labels in order to reduce the current value of $B_f(G)$, we consider the set of critical vertices, where a vertex *v* is *critical if* $B_f(v)=B_f(G)$. Note that it is necessary to reduce the value $B_f(v)$ for all the critical vertices in order to improve the bandwidth of the graph *G*. Moreover, we define a near-critical vertex *v* as one for which $B_f(v) \geq \alpha\ B_f(G)$ with $0<\alpha<1$. 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. We then construct the *candidate list C(f)* of critical and near-critical vertices at each iteration. In order to construct the set of associated moves to each vertex *v* in *C(f)*, we define the following two quantities:

$$max(v) = max\{ f(u) : u \in N(v) \}$$
$$min(v) = min\{ f(u) : u \in N(v) \}$$

Note that the "best label" for *v* in the current labeling *f* is

$$mid(v) = \left\lfloor \frac{max(v) + min(v)}{2} \right\rfloor .$$

The neighborhood of a solution is defined from the set *C(f)*, their best labels and the moves introduced above. Let *v* be a critical or near critical vertex in *C(f)* and *mid(v)* its "best label", we then consider the three following neighborhoods:

Neighborhood 1: exchanges

This neighborhood (proposed by Martí et al., 2001) consists of the set of solutions obtained by applying *move*1*(v,u)* to all vertices *u* with labels *f(u)* that are "closer" to *mid(v)* than *f(v)*. In mathematical terms, we consider the moves *move*1*(v,u)* for all *u* in *N'(v)*:

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

Neighborhood 2: double exchange

This neighborhood consists of the set of solutions obtained by applying *move2(v,u,w)* to the vertex *u* with *f(u)=mid(v)* and to all vertices $w \in N'(u)$. This compound move, in which we first perform *move*1*(v,u)* and then *move*1*(u,w)*, is in fact an implementation of an ejection chain of depth 2. It is initiated by the critical or near critical vertex *v*, which "seeks" the vertex *u* with the best label for *v*, and then vertex *u* "seeks" the vertices *w* with a good label for *u*. Note that in order to reduce the size of this neighborhood, we restrict the search to the vertex *u* in *N'(v)* with the best label for *v*.

Neighborhood 3: multiple shifts

This is the most elaborate neighborhood in our method (proposed by Rodríguez-Tello et al. 2006), since it implies the change of the label in a relatively large number of vertices. It consists of the set of solutions that can be obtained applying *move3(v,u).* Note that this move implements an ejection chain since it involves the change of label of the intermediate vertices $u_1, u_2, \ldots, u_k, u_{k+1}=v$. Specifically, *move3(v,u)* is in fact the result of the consecutive movements *move1(v, u_k)*, *move1(u_k, u_{k-1})*, …, *move1(u_2, u_1)* and *move1(u_1, u).* This move could result in a re-labeling of a large number of vertices. To limit this number and reduce the computational effort, we have restricted its application to those cases in which $k \leq n/5$.

Regardless of the neighborhood that we are using, given a vertex *v* in *C(f)*, $B_f(v)$ will decrease after performing the corresponding move; but we must also consider the change in the bandwidth value of all the vertices involved in the move. For example, in neighborhood 1, when we exchange the labels of *v* and *u*, we need to consider the change in $B_f(v)$, $B_f(u)$ and $B_f(z)$ for all $z \in N(v) \cup N(u)$, and likewise in the other two neighborhoods (in which we need to check the change in the B_f value of a larger number of vertices compared with move 1).

In the computational study in Section 4 we compare the performance of these three neighborhoods. Specifically, we consider three TS algorithms, each one based on each of the neighborhoods above. Moreover, we also study the combination of the three moves into a single method.

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. However, in the context of the MBMP, 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 $B_f(G)$ after a move is computationally expensive since the new value of the bandwidth of the graph can be achieved in one or several vertices not involved in the move. Therefore to update $B_f(G)$ we have to examine all the vertices in the graph.

Given a critical or near critical vertex *v*, the value of an associated move (in any of the neighbourhoods) is the difference between the number of vertices in *C(f)* before and after the move. In mathematical terms:

$$MoveValue(v) = |C(f)| - |C(f')|$$

where *f '* is the labeling obtained when applying the move to the current labeling *f*. A positive *MoveValue* indicates that the solution "improves" since the number of critical vertices decreases, although the objective value may or may not be reduced. This is an extension of the move value definition introduced in Piñana et al. (2004) which differs from the move value proposed in Martí et al. (2001). In addition, we modified this evaluation which takes the value *-K* (the penalization *K* being a huge number) when the application of the move leads to a bandwidth of the vertex *v*, $B_{f'}(v)$ which is larger than the current bandwidth of the graph $B_f(G)$.

Each step of the TS method consists of computing the set of vertices *C(f)* and exploring them in search of improving moves. First we scan all the vertices in the graph to determine $B_f(G)$ and compute *C(f)*. Since this is a time-consuming operation we do not re-compute $B_f(G)$ until all the vertices in *C(f)* have been examined and the selected moves performed. Then we consider each critical vertex *v* in *C(f)* and implement a *first* strategy to explore the set *M(v)* of its associated moves. As opposed to the *best* strategy that selects the move with the largest move value in *M(v)*, the *first* strategy scans *M(v)* in search of the first movement with a strictly positive value. (If no move in *M(v)* is positive, then the one with the largest value is selected) The superiority of the first strategy in the context of different combinatorial optimization problems has been well documented (see for example Laguna et al. 1999).

In the short-term memory design, the identity of a vertex whose label has been changed is the attribute used to impose a tabu restriction. Specifically, after a move in *M(v)* is executed, the labels of the vertex *v* and the other vertices involved are not allowed to change until the tabu tenure expires. This is a straightforward memory structure implementation introduced in Martí et al. (2001), which basically employs a one-dimensional array to store the iteration number each time vertex *v* gains its tabu status, and compares its value with *tenure* (the number of iterations in which a vertex is not allowed to change its label).

We have considered a more elaborate memory structure that implements a different *tenure* value for vertex *v* from the other vertices involved in the move. In such a design, regarding neighborhood 1 (when *move*1*(v,u)* is performed) the *tenure* value for vertex *v* is 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. The same situation appears in neighborhood 2 when *move2(v,u,w)* is performed. On the other hand, since *move3(v,u)* could modify the labels of a large number of vertices ($u, u_1, u_2, \ldots, u_k, v$), we only set the tabu status for the "extreme vertices" *v, u* to avoid an excessive reduction in the search space. We define *c-tenure* as the tabu tenure of the critical or near critical vertex *v* and *nc-tenure* as the tabu tenure of the non-critical vertices in the move. We have also implemented an *aspiration criteria* that permits changing the label of a tabu vertex *u* if its current label *f(u)* is the best label for a vertex *v* in *C(f)*, i.e. *mid(v)=f(u)*, and the associated move has a positive value.

2.2 Long Term Memory

In the long term memory we re-start the search from a new solution after a number of consecutive iterations without improvement. The new solution is generated using a level structure initiated in a vertex which has been randomly selected according to the frequencies collected during the application of the short term memory phase. Let *CFreq(i)* be the number of times that vertex *i* has been included in the set *C(f)*. We consider that the vertices with a large *CFreq*-value usually receive a label that determines the bandwidth of the graph, and thus they are good candidates for initiating a label assignation.

A level structure is a partition of V into sets $L_1, L_2, \ldots, L_k$, called levels, in which vertices adjacent to a vertex in L_i are either in L_{i-1}, L_i or L_{i+1} (if i=1 or k, we do not consider L_{i-1} or L_{i+1} respectively). The re-starting mechanism first finds a level structure initiated in a vertex (root) with a high *CFreq* value. L_1 consists only of this root and the procedure constructs a level structure from this starting vertex as in the GPS method (Gibbs et al. 1976). A second level structure is built with its root in one vertex in L_k, and both structures are merged into a single one as in the GPS method. Then the re-starting mechanism assigns, level-by-level, consecutive labels to the vertices V of G, beginning with label 1. At each iteration of the construction phase, the candidate list *CL* is formed with those vertices in the current level. Therefore, initially $CL=L_1$, and when all the vertices in L_1 have been labeled, $CL=L_2$ and so on, until all the vertices in L_k have been labeled.

We adapt the following constructive method *C*5 introduced in Piñana et al. (2004), which presents a good balance between quality and diversity compared with the other four proposed constructions. Given a label l to be assigned (labels 1,2,.., l-1, have been previously assigned), and a vertex v in level L_i, we define *LeftB(v,l)* as the difference between l and the minimum label of its adjacent vertices in L_{i-1}. We also define *RightB(v,l)* as the difference between the maximum label of its adjacent vertices in L_{i+1} and l. If label l is assigned to vertex v, then $B_f(v)$ would be the maximum between *LeftB(v,l)* and *RightB(v,l)*. However, since the vertices in level L_{i+1} are not yet labeled, we cannot compute *RightB(v,l)* exactly. We compute a lower bound of this value as the number of unlabeled vertices in L_i (excluding v) plus the number of adjacent vertices of v in L_{i+1}. If we wait until the next step to label vertex v (with label $l+1$), *LeftB(v,l)* will increase by a unit, while *RightB(v,l)* will decrease by a unit. Therefore, if *LeftB(v,l)* is greater than *RightB(v,l)* it is better to label vertex v as soon as possible in order to obtain a low $B_f(v)$ value; on the other hand, if it is lower, there is no need to label it now (and thus we can wait until later steps). The restricted candidate list of vertices *RCL* is then built with those vertices v in *CL* with *LeftB(v,l)* > *RightB(v,l)*. A vertex v is randomly selected from *RCL* and labeled with l. The construction phase terminates after n steps, when all the vertices have been selected and labeled.

Our TS algorithm for the MBMP executes the short-term memory component, computing the set of critical and near-critical vertices *C(f)* at each iteration and performing moves to re-label these vertices in order to reduce $B_f(G)$. After a number *tabu_iter* of consecutive iterations without reducing the value of $B_f(G)$, the search is re-initiated by applying the long-term component. Then, a new labeling (solution) is constructed, starting with the node with the largest *Cfreq* value (most of the times the critical one). We have empirically found that an improved strategy alternates this criterion with the selection of the vertex with the lowest degree (for root selection in the construction of the level structure) for diversification purposes. In the construction of a new solution, namely f_k, we apply a filter to discard low quality solutions. Let c_i be the value of solution f_i according to the expression:

$$c_i = \frac{1}{n}\sum_{v \in V} B_{f_i}(v)$$

Then if the value c_k of the new solution f_k satisfies the inequality

$$c_k \leq (1+\beta) \min_{i=1,\ldots,k-1} c_i$$

the solution is considered for improvement and the short-term component is applied. Otherwise, it is discarded and a new solution is generated with the root in the next vertex according to the *Cfreq* value. The algorithm finishes after *max_iter* re-starts. In the computational section we study the influence of the three parameters in the short term component, α (which defines the near critical vertices), *c-tenure* and *nc-tenure* (which controls the tabu tenure); as well as the parameter β that defines the filter in the long-term component.

3. Scatter Search

SS methodology is very flexible, since each of its elements can be implemented in a variety of ways and degrees of sophistication. In this section we propose a SS algorithm for the MBMP based on the well-known "five-method template" (Laguna and Martí 2003):

1. *Diversification Generation*
2. *Improvement*
3. *Reference Set Update*
4. *Subset Generation*
5. *Solution Combination*

The SS procedure starts with the creation of an initial reference set of solutions *RefSet*. The Diversification Generation Method is used to build a set *P* of diverse solutions. The size of *P* (*PSize*) is typically at least 10 times the size of *RefSet* (*b*). The initial reference set is built according to the Reference Set Update Method. It usually consists of selecting *b*/2 best solutions from *P*, as measured by the objective function value and *b*/2 distinct and maximally diverse solutions from *P*. The solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list.

Most of the SS approaches are based on combinations of two elements (subsets of size 2) from the *RefSet*. Although this is a straightforward implementation of the Subset Generation Method, it has proved to be very effective (Campos et al. 2001). According to this, the search is initiated by generating all pairs of reference solutions (resulting in $(b^2-b)/2$ pairs). These pairs are selected one at a time in lexicographical order and the Solution Combination Method is applied to generate one or more trial solutions. These trial solutions are subjected to the Improvement Method. The Reference Set Update Method is now applied to build the new *RefSet* with the best solutions, according to the objective function value, from the current *RefSet* and the set of trial solutions. If *RefSet* changes after the application of the reference set update method, the search continues by applying the subset generation method again. However, now we only consider those pairs in which at least one solution has not been combined in the past (in other words, we do not combine pairs already combined in previous iterations). If *RefSet* does not change after the application of the Reference Set Update Method, the search finishes.

It should be noted that the advanced features of SS are related to the way these five methods are implemented. That is, the sophistication comes from the implementation of the SS methods instead of the decision to include or exclude certain elements (as in the case of TS mentioned above). Three of these Methods, the Diversification Generation, the Improvement and the Combination, are problem dependent and should be designed specifically for the problem at hand (although it is possible to design "generic" procedures, it is more effective to base the design on the specific characteristics of the problem setting). The other two, the Reference Set Update and the Subset Generation Methods, are context independent, and usually have the standard implementation described above (as is the case with our procedure). Our three methods follow:

The ***Diversification Generation Method*** generates a collection of diverse trial solutions with good quality and diversity among them. Piñana et al. (2004) proposed five constructive algorithms for the MBMP within the GRASP methodology. *C*1, *C*2 and *C*3 are based on a node assignment, while *C*4 and *C*5 are based on a level structure. We have considered the methods *C*2 and *C*5 (the best one in each class) to obtain solutions with different structures. Moreover, we propose a new method, *C*6, to increase diversity

in the set *P*. Therefore, in this phase of the SS method, we generate |*P*|/3 solutions with *C*2, |*P*|/3 solutions with *C*5 and |*P*|/3 solutions with *C*6.

The method *C*5 was introduced in the previous section. A description of *C*2 follows. This method starts by creating a list of unlabeled vertices *U*, which at the beginning consists of all the vertices in the graph (i.e. initially *U*=*V*). The first vertex *v* is randomly selected from all those vertices in *U*. We assign a random label to *v* and delete it from *U*. In subsequent construction steps, the candidate list *CL* consists of all the vertices in *U* that are adjacent to at least one labeled vertex. The restricted candidate list *RCL* is formed from those vertices that have been in *CL* for a maximum number of construction steps. A vertex *v* is randomly selected from *RCL* in order to be labeled. Once *v* is selected it is labeled with the best available label according to the vertices already labeled (we compute *mid(v)* considering only the vertices already labeled). Then *U, CL* and *RCL* are updated for the next step. The construction *C*2 terminates after *n* steps, when all vertices have been selected and labeled (i.e. when $U=\varnothing$).

Construction *C*6 implements a deterministic version of *C*5. It assigns, level-by-level, consecutive labels to the vertices *V* of *G* beginning with label 1. Given a label *l* to be assigned (labels 1,2,.., *l*-1, have been previously assigned), we compute as in *C*5 for each vertex *v* in level L_i, *LeftB(v,l)* and *RightB(v,l)*. However, instead of generating a list of candidate vertices based on these values, we directly assign label *l* to the vertex *v* with the minimum difference *LeftB(v,l) - RightB(v,l).* The algorithm first builds a level structure with its root on a vertex with minimum degree. As in *C*5, the construction of the level structure to be labeled consists of the construction of two different structures that are merged into a single one. Then it selects one vertex in L_1 to receive label 1, and finishes after *n* steps according to the values computed above. Further constructions start with the creation of a level structure in another root vertex with minimum degree from those not used and proceed in the same way.

The ***Improvement Method*** transforms a trial solution into one or more enhanced trial solutions. (If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)

Each step in our improvement method consists of selecting a vertex *v* in *C(f)* to be considered for a move. *M(v)* is computed, and a vertex *u* is selected in *M(v)* to perform *move*1*(v,u)*. We implement the *first* strategy that scans *M(v)* in search for the first vertex *u* whose movement results in a strictly positive move value. The local search phase terminates when improvement is no longer possible (i.e. when there is no move that reduces the number of vertices in *C(f)*).

The ***Solution Combination Method*** transforms a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors. As in most SS implementations, we restrict our attention to pairs of solutions. Specifically, we have considered four different combination methods to obtain a new solution from two given solutions *f* and *g*.

Combination Method 1 is based on computing the "average label" of each vertex according to the labels in two given solutions. Let *f* and *g* be two labelings, then we compute for each vertex *v* the "average label" *Avg(v)* as

$$Avg(v) = (1/2)\,(f(v)+g(v))$$

Note that *Avg(v)* is not necessarily an integer number and, moreover, two different vertices could have the same value. Then, in order to obtain a new labeling, we order the vertices according to their Avg-values (the first being the vertex with the lowest value) and assign consecutive labels to them from 1 to *n*.

Combination Method 2 generalizes the previous method. Given two solutions *f* and *g*, it computes the "convex combination label" *Conv(v)* for each vertex *v* with the value

$$Conv(v) = f(v)+\lambda(g(v)-f(v))$$

As in the previous method, vertices are ordered according to these values and consecutive labeling. We will study the influence of the parameter λ (in (0,1)) in the solution quality.

Combination Method 3 selects in step *i* (from *i*=1 to *n*) the label for vertex v_i. It scans (from left to right) each given solution *f* and *g*, and uses the rule that each labeling votes for its first label that has not yet

been included in the combined solution (referred to as the "incipient label"). The voting determines the next label to be assigned to the first as yet unlabeled vertex of the combined solution. This is a min-max rule in the sense that if any label of the solution is chosen other than the incipient label, then it would increase the deviation between the original and the combined permutations. Similarly, if the incipient label were placed later, in the combined solution, than its next available position, this deviation would also increase. Hence the rule attempts to minimize the maximum deviation of the combined solution from the original solutions. This method was successfully applied to the linear ordering problem (Campos et al. 2001).

Note that the MBMP is a problem in which relative positioning of the elements is more important than their absolute positioning. Therefore, before combining two solutions with method 3, we rotate one of them to maximize the number of vertices with the same label in both solutions. Figure 1 shows a representation of two solutions *f* and *g* of a graph with eight vertices in which the label of each vertex is depicted, and Figure 2 shows three consecutive rotations of solution *g*.

f	1	3	4	2	5	8	7	6
g	6	8	7	1	2	3	4	5

Figure 1. Representation of two solutions

g'	**1**	2	3	4	**5**	6	8	7
g''	2	**3**	**4**	5	6	**8**	**7**	1
g'''	8	7	1	**2**	3	4	5	**6**

Figure 2. Rotations of a solution

In Figure 1 we can see that, for example, the label of vertex 2 is 3 in solution *f* and 8 in solution *g*. Solutions *g*', *g*'' and *g*''' are obtained by rotating *g* (shifting all the labels the same number of positions in the permutation vector). Figure 2 depicts in bold the labels that match with solution *f*, for example, vertex 2 is labeled with 3 in *f* and in *g*''. We can see that *g*' has two coincident labels with *f*, *g*'' has four and *g*''' has two, then we would combine *f* with *g*''.

Combination Method 4 first constructs two level structures. The first one is initiated (rooted) in the vertex that has label 1 in *f* and the procedure constructs a level structure from this starting vertex as in the GPS method (Gibbs et al. 1976). Then, from those vertices in the last level of this structure, we select the vertex with the largest label in *g* and the method constructs a second level structure rooted in it. Both structures are then combined and within each level we assign the first labels to vertices with lower Avg-value as computed in Combination Method 1.

Different advanced designs have been proposed for the SS methodology (Laguna and Martí, 2003). In this paper we focus on the use of memory. We have considered the hybridization between SS and TS. Specifically, we propose a version in which we replace the local search procedure with the TS method introduced above. Moreover, we study not only the inclusion of this advanced improvement method, but a simpler version with no improvement at all. This way in the next section we study the contribution of a memory-less local search, as well as a memory-based local search in the SS procedure.

4. Computational Results

For our computational testing, we implemented, in C, our two solving methods, TS and SS, described in the previous sections. We first study the contribution of their different elements and then compare their solutions with those obtained with the previous tabu search procedure of Martí et al. (2001), pTS, the GRASP method by Piñana et al. (2004), the Node Shift and the GA methods by Lim et al. (2005) and the SA method by Rodriguez-Tello et al. (2005). Our codes were compiled with Microsoft Visual C++ 6.0, optimizing for maximum speed. The experiments with the 113 previously reported instances were run on a Pentium IV at 3 GHz with 1GB of RAM. These instances are from the public domain *Harwell-Boeing Sparse Matrix Collection* (http://math.nist.gov/MatrixMarket/data/Harwell-Boeing).

In Martí et al. (2005) we proposed a branch and bound algorithm to obtain lower bounds for the MBMP. In some cases we know that the bound matches the optimal solution, but in most large instances we do not know how far this value is with respect to the optimal solution. We used these lower bounds in all the experiments in this section.

We performed four sets of experiments with the following goals:

1. A preliminary experimentation with 16 instances to find the best values for the key search parameters of our tabu search algorithm TS, α, β, *c-tenure, nc-tenure* and the neighborhood.

2. A preliminary experimentation with 16 instances to study the different combination methods as well as the contribution of the improvement phase in the SS algorithm. We also test the hybridization of the TS and the SS methods.

3. An experiment with the entire set of 113 instances to compare the performance of our two proposals TS and SS, with the best known methods for this problem: pTS, GRASP, Node_Shift and SA.

4. A final experiment to measure the robustness of the proposed procedures on the 16 instances used in experiments 1 and 2. We replicated the methods over 20 independent runs and measure the variation across runs.

The preliminary experimentation was performed on the following 16 representative problem instances (8 medium-size problems and 8 large-size problems): *will57, impcol_b, dwt__234, west0132, impcol_c, west0167, fs_183_1, will199, impcol_a, bcspwr04, gre__343, plskz362, bcspwr05, dwt__592, steam2*, and *662_bus*.

In our first experiments with the TS algorithm we tested values for α in the range [0.1, 0.7] and *c-tenure, nc-tenure* in the range [2, 10]. In these experiments we run the short-term component of the method. Following the recommendations in Martí et al. (2001) we perform it with *tabu_iter*=100 in all the experiments. Table 1 shows the results for different α values (with *c-tenure* = *nc-tenure*=5) and Table 2 shows the results for different tenure values (with α=0.1). Both tables show the average value of the bandwidth in the 16 instances, the average percentage deviation from the lower bound and the running time in seconds.

α	Value	% LB Dev.	Time
0.1	36.13	28.9%	0.60
0.3	36.81	30.5%	0.54
0.5	36.44	29.5%	0.41
0.7	37.19	32.5%	0.37

Table 1. Preliminary experiment. TS with α

c-tenure	nc-tenure	Value	% LB Dev.	Time
5	5	36.13	28.9%	0.60
7	5	36.94	31.3%	0.56
10	5	37.44	32.7%	0.51
7	2	36.44	29.0%	0.58
10	2	36.75	30.1%	0.58
0	0	36.50	31.6%	0.62
5	7	36.63	29.9%	0.55
5	10	36.06	28.4%	0.60
2	7	36.63	30.0%	0.49
2	10	35.94	28.4%	0.59

Table 2. Preliminary experiment. TS with tenure

Table 1 shows that the best results are achieved with α=0.1 and Table 2 shows that, regarding the tabu tenure, the best solutions are obtained with *c-tenure* = 2 and *nc-tenure*=10. This is an interesting result since the foundations of an asymmetric tenure given in Section 2 were based on the hypothesis that the

tenure of a critical vertex *c-tenure* is probably larger than the tenure of a non-critical vertex *nc-tenure*. Note that Table 2 also shows that the results of the local search method without memory structures (*c-tenure* =*nc-tenure*=0) are clearly inferior to those obtained with the selected version with memory, since the former obtains an average percentage deviation from the lower bound of 31.6% while the TS method with *c-tenure* = 2 and *nc-tenure*=10 presents a deviation value of 28.4%. Hence, we use these values to perform the rest of our experimentation.

In the next preliminary experiment we compare the three neighborhoods introduced in Section 2 for the TS method. Table 3 shows the results of three algorithms, each one implementing the TS short-term component with a different neighborhood. Moreover, we consider a fourth method, Mixed, in which the three neighborhoods are combined applying the three associated moves consecutively.

Neighb.	Value	% LB Dev.	Time
1	35.94	28.4%	0.59
2	46.56	62.4%	1.16
3	59.44	114.1%	52.63
Mixed	35.94	28.4%	54.92

Table 3. Preliminary experiment. TS with different neighborhoods

Table 3 shows that the best solutions are obtained with neighborhood 1. This is the simplest neighborhood in terms of the number of operations to perform a move and it explains its extremely low running times (0.59 seconds on average). This experiment shows that within a tabu search framework this is the most effective neighborhood and even the combination of the three moves considered is unable to improve its results. Hence, we use this neighborhood in the rest of our experimentation. Note that these results differ in part with the experiments by Rodríguez-Tello et al. (2006) since their SA implementation based on neighborhood 3 produces remarkable results.

Table 4 reports the results of the long-term TS method for different values of parameter β in the range [0.1, 0.5]. In this experiment we apply the short-term component with *c-tenure* = 2, *nc-tenure*=10, α=0.1, and the global number of iterations *maxiter* set to 25.

β	Value	% Dev.	Time
0.1	33.50	16.8%	14.05
0.3	33.56	16.8%	14.06
0.5	33.69	17.4%	14.67

Table 4. Preliminary experiment. TS with β

Table 4 shows that there are no significant differences among the β values, with only a moderate improvement in low values. We will use β=0.1 in the final experiments.

In our second set of experiments we use the same 16 problem instances to study some key elements in the performance of our SS procedure. Table 5 reports the results of four versions of the SS algorithm, each one with a different combination method (as described in Section 3). Table 6 shows the results of four versions of the SS algorithm (all of them with Combination Method 1), the first one with no improvement method (*No*), in the second one the improvement is applied after the Combination Method (*Post Combi.*), the third one is the standard SS design in which the improvement is applied after the Solution Generation Method and also after the Combination Method (*Standard*), finally, in the fourth version, we apply the short-term memory TS as the improvement (*Tabu*), instead of the local search applied in the other versions.

Combination	Value	% Dev.	Time
1	35.81	26.1%	1.06
2	36.06	26.8%	1.32
3	37.00	30.3%	2.30
4	38.06	33.8%	0.96

Table 5. Preliminary experiment. SS with combination methods

Improvement	Value	% Dev.	Time
No	46.63	61.2%	0.16
Post Combi.	36.31	27.8%	0.65
Standard	35.81	26.1%	1.05
Tabu	33.56	17.0%	111.79

Table 6. Preliminary experiment. SS with improvement methods

Table 5 shows that the best results are obtained with Combination Method 1, which presents an average percent deviation of 26.1%, closely followed by Combination Method 2, which presents a deviation of 26.8%. Table 6 shows that the classic design of SS in which the local search is applied after both, solution generation and combination, obtains better results (26.1%) compared with the version with no local search at all (61.2%) or the version in which the local search is applied only after combination (27.8%). As expected, the use of local search increases the running times (the version with no local search presents an average of 0.16 seconds and the classic SS design presents an average of 1.05 seconds). Finally, the version in which the TS method replaces the local search is the best one in terms of solution quality (17.0% deviation from lower bound) although it presents longer running times (111.79 seconds on average).

In our third set of experiments, we use the 113 *Harwell Boeing* problem instances to compare the performance of our proposed procedures TS and SS_TS (the hybrid version of SS with TS) with the best heuristics reported in the literature. Specifically, we compare their solutions with those obtained with the previous TS procedure of Martí et al. (2001), pTS; the GRASP method with Path Relinking by Piñana et al. (2004), GRASP-PR; the genetic and node shift methods by Lim et al. (2005) with 100 restarts, GA and NS; and the simulated annealing method by Rodríguez-Tello et al. (2005), SA. Tables 7 and 8 report the results on a single run of these seven methods when solving 33 medium (n<200) and 80 large (200 ≤ n ≤1000) instances, respectively. These tables show the average value of the best solution found in a single run, $B_f(G)$, the average percentage deviation of the best solutions from the lower bounds, Dev.; the CPU time in seconds, CPU; and the number of best solutions and optima that each method is able to match. Note that we do not know how far the lower bound is with respect to the optimal solution, so the "true" number of optima achieved by each method could be higher than the numbers in the table, but we can only certify the optimality of the cases indicated (further research in exact methods is needed to make these values accurate).

	pTS	GRASP-PR	TS	GA	NS	SA	SS_TS
$B_f(G)$	23.21	22.52	22.52	22.48	22.36	22.61	22.67
Dev.	15.98%	9.15%	9.26%	10.70%	8.21%	11.60%	9.80%
CPU	3.378	2.854	4.499	2.543	2.177	12.002	22.501
No. of best	11	22	19	16	21	26	21
No. of optima	9	15	11	11	16	9	12

Table 7. Comparison on 33 medium instances

	pTS	GRASP-PR	TS	GA	NS	SA	SS_TS
$B_f(G)$	99.86	99.09	96.16	97.01	97.61	96.59	96.72
Dev.	35.91%	33.41%	27.77%	33.17%	28.38%	30.26%	25.20%
CPU	156.890	151.161	139.417	85.219	240.642	210.255	456.216
No. of best	4	12	27	39	15	34	39
No. of optima	1	4	4	2	5	2	5

Table 8. Comparison on 80 large instances

This experiment clearly shows that the seven methods under comparison obtain solutions of similar values on average. In medium instances, the average value of the best solutions found ranges from 22.36 for the NS method to 23.21 for the pTS. In large instances, this value ranges from 96.16 for the TS to 99.86 for the pTS. Comparing the best with the worst results it represents a variation range of 3.8% on average.

Regarding CPU time, the GRASP-PR, pTS, TS and SS_TS methods were run on a Pentium IV at 3GHz, the SA, as reported in Rodríguez-Tello et al. (2006) was run on a Pentium IV at 2.8GHz, and the GA and NS, as reported in Lim et al. (2005), were run on a Pentium IV at 1.6GHz. The hybrid SS_TS is clearly the most time-consuming method, since it presents an average of 22.501 seconds on medium instances and 456.216 seconds on large instances. The fastest one is surprisingly the GA in both medium and large-size instances.

As mentioned, the NS method provides good results; especially in medium instances in which it is able to obtain the lowest average value (22.36) and the largest number of optima (16) of all the methods in 2.177 seconds. Moreover, as reported in Lim et al. (2005), if we run the NS heuristic for 200 restarts it is able to reduce the average value in medium instances to 22.18, although CPU time increases up to 16.69 seconds on average.

Our tabu search method, TS, provides a good balance between solution quality and speed since it presents an average percentage deviation of 9.26% achieved in 4.49 seconds in medium instances (only improved by the NS and GRASP-PR methods), and a deviation of 27.77% achieved in 139.41 seconds in large instances (no method presents a lower average percentage deviation). Note that in large instances the TS method is closely followed by the NS method in terms of solution quality (27.77 % versus 28.38%); however, the NS needs 240.64 seconds compared with the 139.41 seconds of the TS.

Rodríguez-Tello et al. (2006) replicated their SA method 20 times and report the best, the worst and the average value of the best solutions found in each run. In our final experiment we will report these values on the 16 instances used in the preliminary experiments, collected after 20 replications of our TS and SS_TS methods, to measure their robustness and to compare them with the SA.

	SA	TS	SS_TS
Min. Value	33.37	33.06	33.31
Max. Value	35.19	34.00	34.06
Avg. Value	34.13	33.60	33.58
Avg. Run time	66.11	57.95	182.07

Table 9. Results on 20 independent runs

Table 9 shows that the three methods under comparison are quite robust. The SA presents a variation range between the average of the worst and the best value of the 16 instances in the 20 independent runs of 1.82, the TS 0.94 and the SS_TS 0.75 units. This table also shows a marginal improvement in the SS_TS algorithm with respect to the other two methods, although it presents significantly longer running times (almost 3 times longer than the others).

Conclusions

In this paper we explore the adaptation of the memory programming paradigm to the matrix bandwidth minimization problem. We propose two new methods, one based on TS and the other based on the SS methodology. We study the contribution of some key elements in heuristic search to the final results of the method. Specifically, we consider neighbourhood definition, local search, combination methods and the use of memory.

Our extensive computational testing on a public domain set of previously reported instances show the effectiveness of the use of memory in heuristic search as well as the convenience of employing relatively simple neighbourhoods and combination methods when memory is present. Moreover, the comparison with the best-known methods favours our TS implementation (although some memory-less methods such as simulated annealing or genetic algorithms also present very good results).

Acknowledgements

The authors would like to thank Professors Rodriguez-Tello, Hao and Torres-Jimenez for sharing their results with us in the appropriate format for reporting our experiments.

References

Campos, V., F. Glover, M. Laguna and R. Martí (2001), An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem, *Journal of Global Optimization* 21, 397-414

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

Dueck, G. H. and Jeffs, J., 1995. A Heuristic Bandwidth Reduction Algorithm, *J. of Combinatorial Math. and Comp*. 18, 97-108.

Gibbs, N. E., Poole, W. G. and Stockmeyer, P. K. (1976), An Algorithm for Reducing the Bandwidth and Profile of Sparse Matrix, *SIAM Journal of Numerical Analysis* 13 (2), 236-250.

Glover (1977), Heuristics for Integer Programming Using Surrogate Constraints, *Decision Sciences* 8, 156-166.

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

Laguna, M. and R. Martí (2003), *Scatter Search. Methodology and Implementations in C*, Kluwer academic Publishers, Boston.

Laguna, M., R. Martí, V. Campos (1999), Intensification and Diversification with Elite Tabu Search solutions for the LOP, *Computers and Operations Research* 26, 1217-1230

Lim, A., Rodrigues, B., Xiao, F. (2005), Heuristics for matrix bandwidth reduction, *European Journal of Operational Research*, forthcoming.

Lin, S. and B. Kernighan (1973), An effective heuristic algorithm for the traveling salesman problem, *Operations Research* 21, 498-516.

Martí, R., Laguna, M., Glover, F. and Campos, V. (2001). Reducing the Bandwidth of a Sparse Matrix with Tabu Search, European Journal of Operational Research 135 (2) 211-220.

Martí, R., Campos, V. and Piñana, E., (2006), A Branch and Bound Algorithm for the Matrix Bandwidth Minimization, Technical Report. University of Valencia, Spain.

Piñana, E., Plana, I., Campos, V. and Martí, R., (2004), GRASP and Path relinking for the matrix bandwidth minimization, *European Journal of Operational Research* 153, 200-210.

Papadimitriou, C.H., (1976), The NP-completeness of the bandwidth minimization problem, *Computing* 16 (3) 263-270.

Rodriguez-Tello, E., Jin-Kao, H., Torres-Jimenez, J. (2006), An Improved Simulated Annealing Algorithm for the Matrix Bandwidth Minimization, E*uropean Journal of Operational Research*, forthcoming.