An Improved Simulated Annealing Algorithm for Bandwidth Minimization

Eduardo Rodriguez-Tello [a,*], Jin-Kao Hao [a]
Jose Torres-Jimenez [b]

[a] *LERIA, Université d'Angers*
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
[b] *Mathematics Department, University of Guerrero*
54 Carlos E. Adame, 39650 Acapulco Guerrero, Mexico

Abstract

In this paper, a simulated annealing algorithm is presented for the Bandwidth Minimization Problem for Graphs. This algorithm is based on three distinguished features including an original internal representation of solutions, a highly discriminating evaluation function and an effective neighborhood. The algorithm is evaluated on a set of 113 well-known benchmark instances of the literature and compared with several state-of-the-art algorithms, showing improvements of some previous best results.

Key words: Bandwidth Minimization, Heuristics, Simulated Annealing

1 Introduction

The Bandwidth Minimization Problem for Graphs (BMPG) was originated at the Jet Propulsion Laboratory at Pasadena in 1962, in the study to minimize the maximum absolute errors of six-bit picture codes that were represented by edge differences in a hypercube [14].

The BMPG can be defined formally as follows. Let $G = (V, E)$ be a finite undirected graph, where V ($|V| = n$) defines the set of vertices and $E \subseteq V \times V$

* Corresponding author.
Email addresses: ertello@info.univ-angers.fr (Eduardo Rodriguez-Tello), hao@info.univ-angers.fr (Jin-Kao Hao), jose.torres.jimenez@acm.org (Jose Torres-Jimenez).

$= \{\{i, j\} \mid i,\ j \in V\}$ is the set of edges. Given a one-to-one labeling function $\tau : V \to \{1, 2, .., n\}$, the bandwidth β of G for τ is defined according to the Equation 1.

$$\beta_\tau(G) = \max\{|\tau(i) - \tau(j)| : (i, j) \in E\} \quad (1)$$

Then the BMPG consists in finding a labeling τ^* for which $\beta_{\tau^*}(G)$ is minimum, in mathematical terms:

$$\beta_{\tau^*}(G) = \min\{\beta_\tau(G) : \tau \in \mathcal{T}\} \quad (2)$$

where $\mathcal{T}$ is the set of all possible labeling functions. Please observe that a labeling can also be seen as a permutation. Thus, it is easy to verify that a labeling τ can be transformed into any other labeling τ' by applying to it at most $n - 1$ exchanges between two labels.

Since there are $n!$ possible labelings for a graph with n vertices, the BMPG is a highly combinatorial problem. Papadimitriou has shown that finding the minimum bandwidth of a graph is NP-Complete [23]. This means that it is highly unlikely that there exists an algorithm which finds the minimum bandwidth in time polynomial in the size of the graph. Later, it was demonstrated that the BMPG is NP-Complete even for trees with a maximum degree of three [9]. Only in very special cases it is possible to find the optimal ordering in polynomial time (see for example [9, 17, 26]).

The BMPG is also known under the name of Matrix Bandwidth Minimization Problem (MBMP) [20], which is defined as follows: Given a 0-1 matrix $A = \{a_{ij}\}_{n\times n}$, then the problem consists in finding a permutation of the rows and columns that keeps all the non-zero elements of A in a band that is as close as possible to the main diagonal. The equivalence between the BMPG and the MBMP can be easily observed, given that a graph can be transformed into an incidence matrix $A = \{a_{ij}\}_{n\times n}$ where a_{ij} equal 1 if the vertices i and j are adjacent and 0 otherwise [4].

The bandwidth minimization problem for matrices and graphs has been found to be relevant to a wide range of applications. For instance, in solving large linear systems, the Gaussian elimination can be performed in $O(n\beta^2)$ time on matrices with bandwidth β, which is faster than the normal $O(n^3)$ algorithm if $\beta \ll n$. Other applications include problems in finite element methods for approximating solutions of partial differential equations, large-scale power transmission systems, circuit design, chemical kinetics and numerical geophysics. Recently Berry *et al.* [2] applied bandwidth minimization to the information layout and retrieval in hypertext.

Given its importance, the bandwidth minimization problem has been the object of extensive research (see [4] for a survey up to the early 1980s) and several

algorithms for the BMPG have been reported. They can be divided into two classes: exact and heuristic algorithms. Exact algorithms guarantee always to discover the optimal bandwidth. In [12] is presented an algorithm which solves the BMPG in $O(N^{\beta})$ steps, where β is the bandwidth searched for that graph. Del Corso and Manzini proposed in [5] two other exact algorithms. Both methods solve problems up to 100 vertices, for randomly generated graphs.

On the other hand, heuristic algorithms try to find good solutions as fast as possible, but they do not guarantee the optimality of the solution found. Examples of heuristics are: the Cuthill–McKee algorithm [6] and the Gibbs Poole and Stockmeyer algorithm (GPS) [10]. More recently, metaheuristics have been applied to the BMPG: Simulated Annealing [7], Tabu Search [22], an improved version of the Cuthill–McKee algorithm [8], Genetic Algorithms [18], and GRASP with path relinking [24]. These algorithms are highly successful in solving large instances.

In this paper, we present an improved Simulated Annealing algorithm for the BMPG. This algorithm integrates several important features such as an original internal representation of solutions, a highly discriminating evaluation function called δ and a rotation-based neighborhood function. The performance of this algorithm is assessed with a set of 113 benchmark instances taken from the literature. The computational results are reported and compared with previously published ones, showing that our algorithm is able to improve on some previous best results.

The rest of the paper is organized as follows. In Section 2, a brief review is given to present four most representative solution procedures for the BMPG. In Section 3, the components of our Simulated Annealing algorithm are discussed in detail. The issue of parameter tuning is studied in Section 4. Section 5 is dedicated to computational experiments and comparisons with previous results. Influences of some important parameters are also discussed and analyzed here. The last section summarizes the main contributions of this work.

2 Relevant Existing Procedures

Because of the importance of the bandwidth minimization problem, much research has been carried out in developing effective algorithms for it. The purpose of this section is to give a brief review of four representative algorithms, which are also used for our comparison. For a more detailed description of these algorithms we refer the reader to the corresponding articles.

2.1 GPS

In 1976 Gibbs, Poole and Stockmeyer developed a constructive heuristic, called GPS, for the bandwidth minimization problem [10]. This method is based on a level structure L, generated by breadth first search. The GPS procedure has the following phases:

- The algorithm starts by finding the shortest path connecting two vertices (u, v) of maximal distance apart (the diameter of G). Then it creates two level structures rooted at these endpoints. These level structures are created in the following way: The root node is inserted in level L_1, the rest of the nodes are inserted in such a way that: a) all the adjacent vertices are in the same level or in contiguous levels, and b) The maximum cardinality of all the levels is minimum. By using these endpoints as roots the resulting level structures will have a small width and therefore a maximal depth.
- The two level structures previously created are combined into a new one, whose width usually is smaller than that of the original structures.
- Finally, it assigns consecutive integers to the vertices of the graph by following the resulting level structure and starting at each level by the vertex with the smaller degree. In this way the vertex from L_1 who has the smallest degree will have the number 1 and the vertex with the highest degree in the last level will have the number n.

In [10], computational experiments were performed over 19 matrices with different sizes ranging from 68 to 918. These matrices were taken from the solution of several differential equations and variational problems in structural engineering applications of the finite element method. The authors compared GPS against another constructive heuristic, the reverse Cuthill-McKee (RCM) algorithm [6]. These experiments have shown that on average GPS gives a slightly smaller (better) bandwidth than RCM. Although GPS is actually better than RCM only in 9 out of the 19 cases, it is much faster than RCM.

2.2 Simulated Annealing

In 1995, Dueck and Jeffs implemented the first Simulated Annealing (SA) algorithm for the BMPG [7]. This algorithm represents a labeling of the graph by a permutation of the vertices $\{1, 2, ..., n\}$. The algorithm begins with an initial labeling and carries out a series of iterations to visit the search space $\mathcal{T}$ according to a neighborhood. At each iteration, a neighboring labeling τ' is generated by exchanging randomly two values in the current permutation τ. The cost of τ' is then directly calculated by using Equation 3.

$$\Delta C = \beta_{\tau'}(G) - \beta_{\tau}(G) \tag{3}$$

If ΔC is negative or equal to zero then the neighboring labeling τ' is accepted. Otherwise, it is accepted with probability $P(\Delta C) = e^{-\Delta C/T}$ where T is determined by a cooling schedule.

In their implementation, Dueck and Jeffs use a simple linear function $T_n = 0.95T_{n-1}$ with an initial temperature fixed at $T_i = 1.0$. At each temperature, a maximum number $max\_moves = 4\,|E|$ of neighboring labelings are allowed to be accepted and a maximum number $max\_attempted\_moves = 80 * max\_moves$ of neighboring labelings can be generated. The algorithm stops either if the current temperature reaches a limit ($T_f = 0.0001$), or if the number of accepted configurations at each temperature falls below the limit of $max\_frozen = 50$. The authors justify their choice of these parameter values based on experimental tuning and on their own experience.

Dueck and Jeffs tested their implementation (SA-DJ) using 18 graphs with the number of vertices ranging from 13 to 255. This set includes 8 different types of graphs. Their experiments show that the SA-DJ implementation is inferior to the GPS algorithm in terms of solution quality on structured graphs (grids, paths, circles, etc.). However, SA-DJ outperforms GPS in terms of solution quality on ternary trees and random graphs. It is important to remark that even if SA-DJ finds better labelings than GPS for 11 graphs, it employs longer CPU times. The authors argue that although the computational effort of SA-DJ limits its applicability, it can be used as a reference tool to compare the performance of other procedures.

2.3 Tabu Search

In 2001, Martí *et al.* [22] proposed a Tabu Search (TS) algorithm for the BMPG. It is based on move operations that exchange the labels of a pair of vertices. The operator $move(u, v)$ assigns the label $\tau(u)$ to vertex v and the label $\tau(v)$ to vertex u. These moves are based on the elimination of critical vertices to reduce the current value of $\beta_\tau(G)$. The list of critical and near-critical vertices $C(\tau)$ is defined by Equation 4.

$$C(\tau) = \{v : max\{|\tau(v) - \tau(u)| : u \in A(v)\} \geq \alpha\beta_\tau(G)\} \tag{4}$$

where $A(v)$ is the set of vertices adjacent to v and $1 > \alpha > 0$. In order to create a set of suitable swapping vertices $S(v)$ the authors introduce the following three quantities for a vertex v and a labeling τ:

$$max(v) = max\{\tau(u) : u \in A(v)\} \tag{5}$$

$$min(v) = min\{\tau(u) : u \in A(v)\} \tag{6}$$

$$mid(v) = \left\lfloor \frac{max(v) + min(v)}{2} \right\rfloor \tag{7}$$

The best label for v in the current labeling τ then is given by $mid(v)$, therefore $S(v)$ is defined as follows:

$$S(v) = \{u : |mid(v) - \tau(u)| < |mid(v) - \tau(v)|\} \tag{8}$$

The candidate list of moves associated with a vertex $v \in C(\tau)$ is defined as:

$$CL(v) = \{move(u, v) : u \in S(v)\} \tag{9}$$

In their implementation Martí *et al.* calculate the value of a $move(u, v)$ as the number of vertices adjacent to v or u (including u) whose bandwidth increases due to the move. The basic TS implementation consists of a short-term memory. Specifically, after a $move(u, v)$ is executed, the labels of vertices v and u are not allowed to change until the Tabu tenure expires. A longer-term diversification based on frequency information is added to this basic implementation. The method incorporates a memory structure to record information about previously found solutions. This information is used to restart the search with a new labeling, which is built considering both diversity and quality criteria.

The authors presented a computational comparison of the GPS procedure, the SA-DJ method and two versions of their TS algorithm (with and without the restarting mechanism). For this comparison they have used 113 standard test matrices taken from the Harwell–Boeing Sparse Matrix Collection [1], which includes matrices produced from a wide variety of scientific and engineering disciplines.

The experiments have shown that their TS algorithm yields better solutions than SA-DJ both in terms of quality and speed. The basic implementation of TS is superior to GPS in terms of solution quality and competitive in terms of speed in the small instances. 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.

2.4 GRASP with Path Relinking

Very recently, Piñana *et al.* [24] presented the results of applying a greedy randomized adaptive search procedure combined with a path relinking strategy (GRASP-PR) to the BMPG.

[1] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing

The GRASP-PR procedure starts with the creation of an initial elite set of labelings. The GRASP method is used to build a large set of labelings from which the *n_elite* bests are stored in the elite set. The relinking process then explores all pairs of elements in the elite set using a lexicographical order. Each time the relinking process replaces the worst element in the elite set with the improved labeling generated. The GRASP-PR procedure ends when no more improvements can be done to the elite set.

The GRASP method has two main steps: the construction phase and the improvement phase. During the *construction phase* a level structure, similar to the one used by the GPS procedure, is created. The *improvement phase* typically consists of a local search procedure and in this implementation is partially based on the Tabu Search algorithm proposed in [22]. The authors consider a set of critical vertices $C(\tau)$, the operator $move(u, v)$ and a candidate list of moves $CL(v)$ associated with a vertex $v \in C(\tau)$.

The main differences with the TS proposed by Martí *et al.* are the following: The near-critical vertices in the set $C(\tau)$ are not considered and the value of a move is defined as the difference between the number of critical vertices before and after the move, *i.e.*: $move\_value(u, v) = C(\tau) - C(\tau')$, where τ' is the labeling obtained when applying $move(u, v)$ to the current labeling τ. Thus, a positive move value indicates that the solution improves in the sense that the number of critical vertices decreases, although the $\beta_\tau(G)$ value may or may not be reduced.

On the other hand the *Path Relinking* approach is used to generate new solutions (*i.e.* labelings) by exploring trajectories that connect high-quality solutions, by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

The authors present an extensive computational comparison of the GPS procedure, the TS method and two versions of their GRASP algorithm (with and without the path relinking mechanism). For this comparison they have used 113 instances from the Harwell–Boeing Sparse Matrix Collection. The experiments shown that the best solution quality is obtained by GRASP-PR, which was able to match 81 out of the 113 best known solutions, and outperforms thus the TS heuristic that was only able to find 41 best solutions.

3 An Improved Simulated Annealing Algorithm

In this section a SA implementation for solving the BMPG is presented. This SA has the merit that it improves three key features that have a great impact in this heuristic search: the internal representation, the evaluation function, and the neighborhood function. Next all the details of the new implementation proposed, called SA-δ, are presented.

3.1 Internal Representation

Given a graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E. A potential labeling τ is defined as: $\tau : V \rightarrow \{1, 2, ..., n\}$. Then a labeling τ is represented as an array l of integers of length n, which is indexed by the labels and whose *i-th* value $l[i]$ denotes the vertex with the label i (see Fig. 2(a)). This representation is different from the one reported by other authors [7, 18, 22, 24] which also uses an array of n integers, but with another interpretation where the *i-th* element represents the label assigned to the vertex i.

The new representation proposed in this work has an important characteristic: given that the contiguous positions in the array represent labels in the graph with an unitary difference, an interchange of two adjacent elements produces smooth changes in the configuration, because the contribution of these labels to the graph bandwidth will change at maximum in one unit.

This concept will be best understood with the following example. Consider the graph in Fig. 1(a). It consists of 5 vertices, identified with letters. The current labeling τ is represented as a number inside each vertex and over each edge the absolute difference between the labels of adjacent vertices is indicated. Suppose that we take the contiguous labels 2 and 3 (*i.e.* the second and third elements in the new representation) and we interchange them. Then, the label 2 will be assigned to the vertex e and the label 3 will be assigned to the vertex

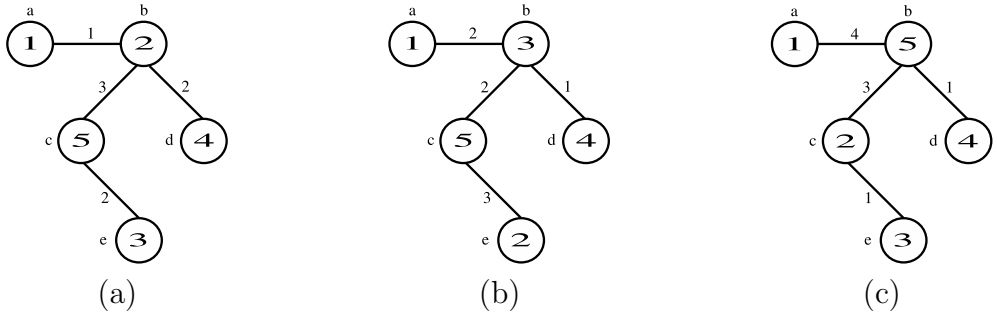


Fig. 1. (a) Labeling τ with β=3. (b) Labeling τ' with β=3. (c) Labeling τ'' with β=4.

b. The resulting labeling τ' is presented in Fig. 1(b). It is important to note that the changes in the absolute differences between the labels of adjacent vertices have been changed at maximum in one unit (see for example the edges $\{c, b\}$ and $\{c, e\}$), even if it is recognized that the change is maximum in the extremes that limit the rotation.

In contrast, if the classical representation for the BMPG is used and we take the contiguous vertex b and c (*i.e.* also the second and third elements in the classic representation) and we interchange them. Then, the vertex b will have the label 5 and the vertex c will have the label 2. As a result the labeling τ'', shown in Fig. 1(c), is produced. Observe that the changes in the absolute differences between labels of adjacent vertices have been strongly changed. They have even increased the bandwidth of the graph (see edge $\{a, b\}$).

3.2 Neighborhood Function

The search space $\mathcal{T}$ for the BMPG is composed of all possible labelings from V to $\{1, 2, ..., n\}$. It is easy to see then, that there are $n!$ possible labelings for a graph with n vertices. From this search space, a neighborhood function can be then introduced. Let τ be a labeling in $\mathcal{T}$, the neighborhood $N(\tau)$ of a labeling in our SA implementation is such that for each $\tau' \in \mathcal{T}$, $\tau' \in N(\tau)$ if and only if τ' can be obtained by rotating the labels of any group of vertices from τ. Let $swap(\tau(i), \tau(j))$ be a function allowing to exchange two labels of τ, then a rotation between two labels $\tau(i)$ and $\tau(j)$ can be expressed as a product of $(j - i)$ swaps with the Formula 10.

$$\begin{aligned} rotation(\tau(i), \tau(j)) = {} & swap(\tau(i), \tau(j)) * swap(\tau(i), \tau(j-1)) * \\ & swap(\tau(i), \tau(j-2)) * ... * \\ & swap(\tau(i), \tau(i+1)) \end{aligned} \tag{10}$$

where $0 \leq \tau(i) \leq n-1$, $0 \leq \tau(j) \leq n-1$ and $\tau(i) < \tau(j)$. In this way a rotation can be seen as a compound move. We have decided to use the rotation move because it is well known that compound moves can lead to better local search methods than those using only simple movements [3, 11, 19]. Experimental studies presented in Section 5.3.2 also allow to confirm the superiority of the

Fig. 2. (a) Original labeling τ. (b) The resulting neighboring labeling τ' generated by applying $rotation(\tau(3), \tau(7))$ over τ.

rotation move over the conventional *swap* move.

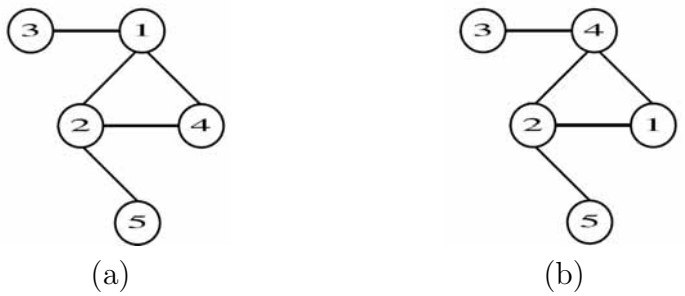
Let us present a graphical example of rotation to better understand the concept. In Fig. 2(a) a labeling τ for a graph with 9 vertices is presented using the representation proposed above. Suppose that we select $\tau(3)$ and $\tau(7)$ as the parameters for the rotation. Then by applying the definition, each label $\tau(y)$ where $4 \leq y < 7$ is displaced one position to the left in the representation. Finally the element that occupied the place $\tau(3)$ is moved to the position $\tau(7)$. The resulting neighboring labeling τ' is presented in Fig. 2(b).

3.3 Evaluation Function

The choice of the evaluation function is a very important aspect of any search procedure. Firstly, in order to efficiently test each potential solution, the evaluation function must be as simple as possible. Secondly, it must be sensitive enough to locate promising search regions on the space of solutions. Finally, the evaluation function must be consistent: a solution that is better than others must return a better value.

The most common practice in the algorithms for the BMPG reported in the literature [6, 8, 10, 18] is to evaluate the quality of a configuration as the change in the objective function β. This, however, provides little or no information during the search process because β only takes into consideration the maximum absolute difference between labels of adjacent vertices in the graph (see Equation 1). In this sense β is not sufficiently discriminating because there is no way to make distinctions between solutions with the same β value. For example, the two labelings for the graph showed in Fig. 3 have the same bandwidth ($\beta = 3$). However, a closer look allows one to confirm that the labeling in Fig. 3(b) is intrinsically better than the one in Fig. 3(a). Indeed it is better because it has only one absolute difference with value two.

Two exceptions are reported in the literature concerning the evaluation function for the BMPG. In [7], the authors take into account the five maximum



(a) (b)

Fig. 3. (a) Labeling τ with β=3. (b) Labeling τ' with β=3.

absolute differences to evaluate a configuration. In [28], an evaluation function, called γ, is presented. It takes into consideration all the edges of the graph. Unfortunately due to its nature it can only be calculated efficiently for graphs with less than 150 vertices.

Given the negative features of using β as an evaluation function and following the ideas presented in [7, 28] it has been developed a new evaluation function, called δ, which takes into account all the edges of the graph [25]. The proposed evaluation function for a labeling τ is defined by the Equation 11 where d_x refers to the number of absolute differences with value x between adjacent vertices, and β the bandwidth for the labeling τ.

$$\delta(\tau) = \beta + \sum_{x=1}^{\beta} \left(\frac{d_x}{\frac{(n+\beta-x+1)!}{n!}} \right) \tag{11}$$

To illustrate the computation of this new evaluation function, let us consider the graph in Fig. 3(a). For this particular graph: $d_1 = 1$, $d_2 = 2$, $d_3 = 2$, $d_4 = 0$; additionally it is easy to observe that $\beta = 3$ and $n = 5$. Then, by making the substitution of these values in the Formula 11 and simplifying we obtain: $\delta(\tau) = 3 + \frac{1}{336} + \frac{2}{42} + \frac{2}{6} = 3.3839$.

In contrast if δ is computed for the labeling τ' showed in Fig. 3(b) we obtain a different and smaller value $\delta(\tau') = 3 + \frac{2}{336} + \frac{1}{42} + \frac{2}{6} = 3.3631$.

The main idea of the new evaluation function δ is to penalize the absolute differences d_x with small values of x and to favor the absolute differences with values of x near to β. The logic behind this is that it is easier to reduce the absolute differences with value β to values close to β. In this sense the δ function is much more discriminating than β and leads to smoother landscapes during the search process. This is possible because δ has the ability to create more equivalence classes with a lower cardinality. This is an important characteristic which allows to capture even the smallest improvement that orients the searching process of solutions and permits to find configurations where all the absolute differences between labels of adjacent vertices are minimized; and not only the maximum one as it happens with β. Experimental studies presented in Section 5.3.3 allow to confirm the superiority of δ over β. A detailed comparative study between β and δ can be found in [25].

3.4 Initial Solution

The initial solution is the starting labeling used for the algorithm to begin the search of better configurations in the search space $\mathcal{T}$. In this implementation the starting solution is generated randomly.

3.5 Annealing Schedule

The annealing schedule determines the degree of uphill movement permitted during the search and is, thus, critical to the algorithm's performance. The parameters that define an annealing schedule are: an initial temperature, a final temperature or a stopping criterion, the maximum number of neighboring solutions that can be generated at each temperature, and a rule for decrementing the temperature.

The literature offers a number of cooling schedules, see for instance [1, 13, 15, 29] for several examples. In the SA-δ implementation we preferred a geometrical cooling scheme mainly for its simplicity. It starts at an initial temperature T_i and, at each round, decrements the current temperature by a factor of α using the relation $T_k = \alpha T_{k-1}$. For each temperature, the maximum number of neighboring labelings accepted is $NA_{max} = \mu|E|$, while the maximum number of visited neighboring labelings is $NV_{max} = \mu NA_{max}$. Both NA_{max} and NV_{max} depend directly on the number of edges ($|E|$) of the graph and on the numeric constant μ, that we call *moves factor*. This is because more moves are required for denser graphs. We will see later that thanks to the three main original features presented previously, SA-δ using this simple cooling scheme gives remarkable results.

3.6 Termination Condition

The algorithm stops either if the current temperature reaches T_f, or when it ceases to make progress. In the proposed implementation a lack of progress exists when the number of accepted configurations at each temperature goes below the limit ϕ (*frozen factor*).

All the parameters of this SA algorithm were chosen experimentally, and taking into account some related work reported in [16, 28]. In the next section the computational experiments performed to identify these parameter values are presented. Additional experiments are reported in Section 5.3.1.

4 Parameter Tuning

It is well known that the performance of the SA algorithm is sensitive to parameter tuning. Next we present the experimentation methodology used consistently throughout the tuning process, to obtain good values for the key parameters.

First, a subset of 10 representative instances was selected from well-known benchmark problems. All the experiments showed in this section were executed over this set of instances.

For each parameter, an interval of reasonable size is determined based on some preliminary experiments. Then 20 executions with each values combination are performed for each problem instance. Finally, the data generated by these tests is analyzed to obtain the parameter combination which yields the best performance for the algorithm, both in solution quality and in execution time.

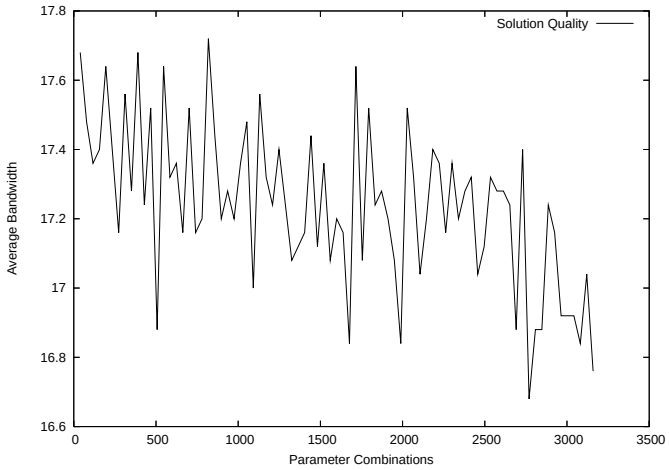
Next, the key parameters used in this study are presented. For each one of them we indicate also its interval and step S:

- Initial temperature $4.0E^{-4} \leq T_i \leq 1.2E^{-2}$ with $S = 2.0E^{-3}$
- Final temperature $8.0E^{-10} \leq T_f \leq 1.6E^{-9}$ with $S = 2.0E^{-10}$
- Cooling rate $0.88 \leq \alpha \leq 0.96$ with $S = 0.02$
- Moves factor $10 \leq \mu \leq 14$ with $S = 1$
- Frozen factor $10 \leq \phi \leq 30$ with $S = 5$

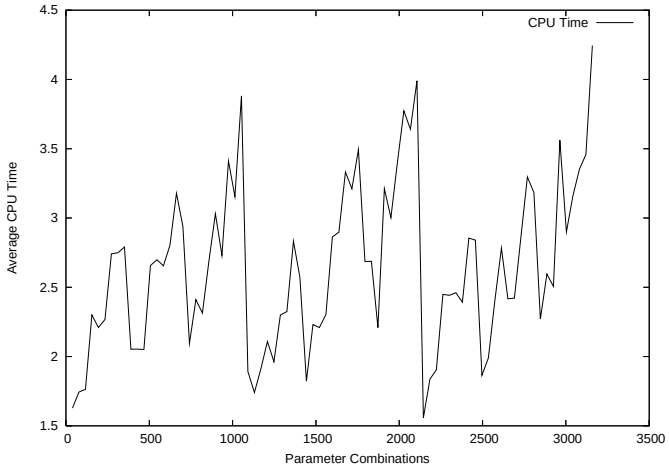
These intervals and steps give a total of 3125 parameter value combinations that we have tested. In Fig. 4(a) the average bandwidth reached over the 10 representative instances for each of these combinations is presented, while in Fig. 4(b) the average CPU time expended is shown. From these graphics we have selected the 10 best combinations. Their average bandwidth and the average CPU time in seconds over the 10 problem instances are presented in Table 1. From these results we have selected the parameter combination that gives the best trade-off between solution quality and computational effort. The best average bandwidth with an acceptable speed is reached when $T_i = 1.0E^{-2}$, $T_f = 1.0E^{-9}$, $\alpha = 0.92$, $\mu = 12$ and $\phi = 25$. These values are thus used in the experimentation reported in the next section.

Table 1
Average results from the 10 best parameter value combinations in the tuning experiments.

T_i	T_f	α	μ	ϕ	$\beta_\tau(G)$	t
$4.0E^{-3}$	$8.0E^{-10}$	0.94	13	15	17.72	2.314
$4.0E^{-3}$	$8.0E^{-10}$	0.94	13	20	17.28	2.730
$8.0E^{-3}$	$1.4E^{-9}$	0.90	11	20	17.24	2.109
$8.0E^{-3}$	$1.2E^{-9}$	0.94	12	15	17.36	2.210
$8.0E^{-3}$	$1.0E^{-9}$	0.90	12	25	16.84	3.332
$8.0E^{-3}$	$8.0E^{-10}$	0.92	12	25	17.64	3.211
$1.0E^{-2}$	$1.2E^{-9}$	0.90	11	15	17.20	1.558
$1.0E^{-2}$	**$1.0E^{-9}$**	**0.92**	**12**	**25**	**16.68**	**3.297**
$1.0E^{-2}$	$8.0E^{-10}$	0.90	13	20	16.92	3.561
$1.0E^{-2}$	$1.4E^{-9}$	0.94	13	25	16.76	4.245



(a)



(b)

Fig. 4. Graphics showing the average results obtained in the tuning experiments using 3125 parameter value combinations over a set of 10 problem instances.

5 Computational Experiments

In this section, we present the experiments accomplished to evaluate the performance of SA-δ over a set of 113 benchmark instances. 12 more instances are also used to study the influence of some critical elements of the SA-δ algorithm. For these experiments the SA described in Section 3 is used, together with the parameter values obtained in Section 4. The algorithm was coded in C and compiled with *gcc* using the optimization flag -$O2$. It was run into a Pentium 4 at 2.8 GHz. with 1 GB of RAM. Due to the incomplete and non-deterministic nature of the method 20 independent runs were executed for each of the selected benchmark instances.

5.1 Benchmark Instances and Comparison Criteria

The set of 113 problem instances used in our experimentation come from the Harwell-Boeing Sparse Matrix Collection [2]. This is the same test data used by Martí *et al.* [22] and Piñana *et al.* [24]. These 113 problem instances are divided into two subsets. The first subset is composed of 33 instances with 30 to 199 vertices. The second subset consists of 80 large instances whose sizes vary from 200 to 1000.

To assess the performance of our SA-δ, we show comparative results on these benchmark instances. The main criterion used for the comparison is the same as the one commonly used in the literature: the minimum bandwidth obtained. Computing time is also given for indicative purpose. Notice that a finer comparison may be achieved by a new methodology recently proposed by E. Taillard [27] which is based on success rates. However, this testing is impossible in our case since we have no access to the success rates of the main competing algorithms (TS and GRASP-PR). Consequently and like many previous studies, we make our comparisons based on information such as the best and average result obtained by each algorithm.

The comparison is carried out in two parts. The first part gives an instance-by-instance comparison of our SA-δ algorithm with respect to two highly effective heuristic algorithms *i.e.* Tabu Search (TS) [22] and GRASP with Path Relinking (GRASP-PR) [24]. Indeed, TS and GRASP-PR held the best results on the benchmark instances before SA-δ was developed. The second part shows a global performance comparison using averaged results. This second comparison includes not only SA-δ, TS and GRASP-PR, but also two other well-known algorithms, *i.e.* GPS [10] and Dueck and Jeff's Simulated Annealing (SA-DJ) [7].

5.2 Comparison Between SA-δ and the Best Known Results

Tables 2 and 3 present the detailed computational results of our SA-δ algorithm as well as those obtained by TS and GRASP-PR over the two subsets of benchmark instances. In both tables, we indicate the name of the graph, its number of vertices, the bandwidth found (β) and the execution time in seconds (t) for the TS and GRASP-PR algorithms [3]. Columns 7 to 11 show the best (β_b), worst (β_w), average ($Avg.$) and standard deviation ($Dev.$) of the bandwidth found by SA-δ over 20 independent executions and its average

[2] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing

[3] The results of TS and GRASP-PR are taken from [21, 24] and correspond to one single run for each algorithm and each problem instance.

Table 2
Performance comparison using 33 small instances from the Harwell-Boeing Sparse Matrix Collection. Results for the TS and GRASP-PR algorithms were taken from [21, 24] where a K-7 Athlon at 1.2 GHz. was used.

		TS		GRASP-PR		SA-δ						
Graph	n	β	t	β	t	β_b	β_w	*Avg.*	*Dev.*	t	β^*	Δ
arc130	130	63	4.8	63	1.9	63	64	63.8	0.4	23.2	63	0
ash85	85	10	0.7	9	0.4	9	9	9.0	0.0	1.1	9	0
bcspwr01	39	5	0.1	5	0.1	5	6	5.8	0.4	0.4	5	0
bcspwr02	49	7	0.2	7	0.6	7	8	7.8	0.4	0.2	7	0
bcspwr03	118	11	1.7	11	0.9	10	11	10.6	0.5	1.2	11	-1
bcsstk01	48	16	0.9	16	1.0	16	20	18.6	1.4	0.6	16	0
bcsstk04	132	38	16.7	37	5.4	37	42	37.6	1.2	79.2	37	0
bcsstk05	153	23	4.7	20	7.1	20	22	20.8	1.0	10.6	20	0
bcsstk22	110	11	1.1	10	1.6	11	11	11.0	0.0	5.5	10	1
can_144	144	14	1.7	14	3.1	13	13	13.0	0.0	17.6	14	-1
can_161	161	20	4.0	18	0.7	18	19	18.8	0.4	3.0	18	0
curtis54	54	10	0.7	10	0.7	10	10	10.0	0.0	0.5	10	0
dwt_234	117	11	1.2	11	1.9	11	13	12.0	0.9	1.1	11	0
fs_183_1	183	63	32.4	61	11.8	61	66	62.6	2.1	8.7	61	0
gent113	104	27	6.3	27	1.0	27	27	27.0	0.0	3.9	27	0
gre_115	115	25	2.4	24	3.2	23	24	23.6	0.5	1.6	24	-1
gre_185	185	22	6.2	22	6.1	22	24	22.2	0.5	6.8	22	0
ibm32	32	12	0.2	11	0.3	11	13	11.8	0.8	0.3	11	0
impcol_b	59	21	1.3	21	1.2	20	23	21.0	1.3	1.2	21	-1
impcol_c	137	33	3.5	31	4.5	30	31	30.6	0.5	3.1	31	-1
lns_131	123	21	3.4	22	2.6	20	21	20.6	0.5	1.8	21	-1
lund_a	147	25	8.8	23	5.4	23	23	23.0	0.0	40.6	23	0
lund_b	147	26	5.3	23	5.5	23	23	23.0	0.0	40.8	23	0
mcca	168	38	23.9	37	10.8	37	38	37.4	0.5	81.8	37	0
nos1	158	7	1.3	3	2.6	3	3	3.0	0.0	1.1	3	0
nos4	100	10	1.1	10	1.4	10	10	10.0	0.0	0.9	10	0
pores_1	30	7	0.3	7	0.3	7	9	8.0	0.9	3.1	7	0
steam3	80	7	0.7	7	1.0	7	7	7.0	0.0	8.9	7	0
west0132	132	36	3.4	35	8.5	33	36	34.0	1.1	5.4	35	-2
west0156	156	39	7.4	37	8.4	36	37	36.8	0.4	2.8	37	-1
west0167	167	35	5.8	35	5.6	34	35	34.4	0.5	4.8	35	-1
will199	199	70	12.0	69	26.9	65	67	65.7	0.8	6.1	69	-4
will57	57	7	0.4	7	0.4	6	6	6.0	0.0	1.1	7	-1
Average		23.33	4.99	22.52	4.03	22.06	23.36	22.62	0.52	11.18	22.48	

CPU time in seconds (t) [4]. The last two columns show the previous best-known solution (β^*) and the difference between the best solution found by SA-δ (β_b) and the previous best-known value (Δ).

From Table 2, it can be observed that globally the results of TS and GRASP-PR, obtained in only one execution, are slightly better than the worst value (β_w) attained by SA-δ (23.33 and 22.52 against 23.36) over 20 runs. This difference is reduced when the average result (*Avg.*) of SA-δ (22.62) is compared with that achieved by GRASP-PR and it is even transformed into an improvement of 3.05% with respect to the TS algorithm. Finally, one observes easily that the best solution (β_b) found by SA-δ, is better than the previous best known solution (β^*) in 11 out of 33 instances.

[4] Computing times are given only for indicative purpose since TS and GRASP-PR algorithms were run on a K-7 Athlon at 1.2 GHz which is slower than our Pentium 4 at 2.8 GHz.

On the other hand, the data presented in Table 3 show that globally the worst quality solution (β_w) of SA-δ (97.44) is better than that achieved by TS and GRASP-PR (100.78 and 99.43), which represents an improvement of 3.31% and 2.00% respectively. These improvements increase to 5.04% and 3.75% when the average result ($Avg.$) of SA-δ (95.70) is considered. Finally, a check of the best result (β_b) obtained by SA-δ shows that in 46 out of 80 cases the previous best known result is improved. Notice that for several instances the improvement is relatively important; leading to a significant decrease of the bandwidth (Δ up to -19) and only in 4 instances, SA-δ has a slightly inferior result ($\Delta = 1$). Finally, notice that the computing time required by SA-δ to achieve these results remains competitive.

Table 3
Performance comparison using 80 large instances from the Harwell-Boeing Sparse Matrix Collection. Results for the TS and GRASP-PR algorithms were taken from [21, 24] where a K-7 Athlon at 1.2 GHz. was used.

		TS		GRASP-PR		SA-δ						
Graph	n	β	t	β	t	β_b	β_w	$Avg.$	$Dev.$	t	β^*	Δ
494_bus	494	32	29.2	35	13.5	32	37	33.8	1.8	24.4	32	0
662_bus	662	42	113.8	44	32.9	43	44	43.2	0.4	70.4	42	1
685_bus	685	38	90.4	46	12.7	36	38	37.0	1.0	62.0	38	-2
ash292	292	24	7.9	22	8.7	21	21	21.0	0.0	34.4	22	-1
bcspwr04	274	26	9.6	26	5.2	25	25	25.0	0.0	28.0	26	-1
bcspwr05	443	29	24.7	35	16.3	30	32	31.4	0.8	26.8	29	1
bcsstk06	420	47	47.6	50	50.6	45	45	45.0	0.0	247.9	47	-2
bcsstk19	817	20	174.3	16	86.6	15	16	15.2	0.4	222.9	16	-1
bcsstk20	467	21	27.8	19	11.0	14	18	14.9	1.1	44.6	19	-5
bcsstm07	420	50	40.3	48	113.7	48	50	48.4	0.8	215.7	48	0
bp_0	822	252	386.8	258	483.0	240	251	241.7	2.7	140.7	252	-12
bp_1000	822	313	886.7	297	886.2	291	295	292.2	1.4	216.2	297	-6
bp_1200	822	320	674.8	303	897.1	296	299	297.2	1.2	218.2	303	-7
bp_1400	822	327	521.0	313	741.2	300	306	302.8	2.5	217.4	313	-13
bp_1600	822	322	546.8	317	783.7	299	308	300.6	2.8	231.8	317	-18
bp_200	822	281	315.2	271	550.3	267	272	268.7	1.8	168.4	271	-4
bp_400	822	288	355.7	285	560.6	276	282	277.8	2.3	180.0	285	-9
bp_600	822	299	480.9	297	556.8	279	287	281.8	2.9	193.3	297	-18
bp_800	822	305	520.9	307	636.9	286	289	286.9	1.2	219.9	305	-19
can_292	292	41	19.0	42	7.9	41	46	42.5	1.8	61.5	41	0
can_445	445	56	64.5	58	47.2	56	63	57.8	2.7	114.0	56	0
can_715	715	74	183.1	78	14.2	74	78	75.1	1.4	223.0	74	0
can_838	838	91	158.9	88	37.4	88	89	88.2	0.4	384.2	88	0
dwt_209	209	25	10.8	24	1.3	23	27	23.8	0.9	29.2	24	-1
dwt_221	221	15	5.0	13	7.0	13	13	13.0	0.0	23.4	13	0
dwt_245	245	25	7.5	26	14.4	24	25	24.2	0.4	9.3	25	-1
dwt_310	310	13	15.2	12	8.6	12	13	12.2	0.4	13.0	12	0
dwt_361	361	16	11.8	15	0.8	14	14	14.0	0.0	8.3	14	0
dwt_419	419	32	23.7	29	28.0	26	26	26.0	0.0	59.9	29	-3
dwt_503	503	45	99.3	45	7.9	44	47	44.7	0.9	163.5	45	-1
dwt_592	592	33	52.9	33	106.8	32	33	32.4	0.5	123.7	33	-1
dwt_878	878	31	195.0	35	99.6	26	26	26.0	0.0	106.7	27	-1
dwt_918	918	37	290.8	36	12.6	33	36	34.4	1.0	243.0	36	-3
dwt_992	992	64	272.5	49	306.4	35	36	35.6	0.5	116.4	35	0
fs_541_1	541	270	54.4	270	26.5	270	270	270.0	0.0	82.4	270	0
fs_680_1	680	19	43.4	17	33.6	17	19	18.4	0.8	39.8	17	0
fs_760_1	760	40	101.1	39	97.9	38	38	38.0	0.0	95.6	39	-1
gr_30_30	900	44	282.2	58	85.4	45	49	46.6	1.7	370.1	44	1
gre_216a	216	22	7.2	21	9.5	21	21	21.0	0.0	3.5	21	0
gre_343	343	29	16.6	29	21.2	28	28	28.0	0.0	7.4	28	0

Continues in the following page ...

Table 3 – continued from previous page

		TS		GRASP-PR		SA-δ						
Graph	n	β	t	β	t	β_b	β_w	$Avg.$	$Dev.$	t	β^*	Δ
gre_512	512	37	77.1	36	92.7	36	36	36.0	0.0	14.7	36	0
hor_131	434	60	26.0	64	27.7	55	56	55.4	0.5	154.1	60	-5
impcol_a	206	35	5.5	34	3.4	32	33	32.6	0.5	5.4	34	-2
impcol_d	425	44	21.9	42	30.5	42	49	43.6	2.8	77.5	42	0
impcol_e	225	44	10.7	42	4.0	42	43	42.2	0.4	49.6	42	0
jagmesh1	936	31	150.6	27	107.8	27	27	27.0	0.0	33.8	27	0
jpwh_991	983	94	312.6	96	65.1	94	103	96.6	3.5	104.3	94	0
lns_511	503	48	74.2	49	58.3	44	46	44.8	0.8	123.8	48	-4
mbeacxc	487	270	3409.1	272	5464.5	262	264	263.0	0.9	1774.0	270	-8
mbeaflw	487	270	3409.4	272	5467.8	261	263	261.8	0.8	1744.9	270	-9
mbeause	492	260	2637.6	269	5494.3	255	260	256.6	1.9	1289.5	260	-5
mcfe	731	135	800.2	130	247.2	126	127	126.2	0.4	1868.9	130	-4
nnc261	261	26	8.6	25	22.4	25	25	25.0	0.0	8.7	25	0
nnc666	666	45	138.5	45	55.8	42	43	42.2	0.4	108.5	45	-3
nos2	638	9	43.7	3	15.7	3	5	3.8	0.8	114.5	3	0
nos3	960	76	143.8	79	209.1	62	64	63.0	0.6	811.6	75	-13
nos5	468	67	60.8	69	103.1	64	64	64.0	0.0	121.9	67	-3
nos6	675	21	70.9	16	42.8	16	16	16.0	0.0	12.5	16	0
nos7	729	73	74.5	66	89.9	65	65	65.0	0.0	23.9	65	0
orsirr_2	886	95	203.0	91	42.5	88	89	88.4	0.5	164.5	91	-3
plat362	362	38	24.8	36	3.4	36	39	36.5	1.1	189.4	36	0
plskz362	362	19	27.4	20	7.1	19	24	21.2	2.4	20.9	19	0
pores_3	456	17	16.8	13	3.2	13	15	14.2	1.0	13.0	13	0
saylr1	238	16	4.3	15	8.5	14	14	14.0	0.0	2.3	14	0
saylr3	681	53	107.2	52	77.8	53	57	54.4	1.5	15.6	52	1
sherman1	681	53	107.2	52	78.0	52	57	53.4	1.9	15.6	52	0
sherman4	546	29	33.0	27	4.1	27	27	27.0	0.0	11.5	27	0
shl_0	663	235	153.1	241	110.2	229	231	230.2	0.8	211.5	235	-6
shl_200	663	245	161.3	247	98.4	235	241	238.4	2.0	213.5	245	-10
shl_400	663	243	188.3	242	121.2	235	239	237.0	1.5	221.4	242	-7
steam1	240	50	16.9	46	12.7	44	47	45.2	1.2	79.1	46	-2
steam2	600	80	242.2	65	182.5	65	67	66.0	0.9	687.4	65	0
str_0	363	124	120.8	124	119.1	119	121	120.0	0.9	39.9	124	-5
str_200	363	136	90.9	135	114.1	128	135	132.0	2.7	47.3	135	-7
str_600	363	143	180.3	144	92.0	132	134	132.8	0.8	55.9	143	-11
west0381	381	164	113.0	159	185.4	153	157	154.4	1.4	38.1	159	-6
west0479	479	130	81.2	127	163.0	123	125	123.6	0.8	40.5	127	-4
west0497	497	90	78.3	92	88.7	87	89	88.2	0.8	137.9	90	-3
west0655	655	171	150.1	167	245.8	161	169	162.1	2.0	80.5	167	-6
west0989	989	228	372.7	217	416.9	215	217	216.0	0.9	172.1	217	-2
Average		100.78	263.97	99.43	339.97	94.80	97.44	95.70	0.97	199.25	97.98	

In the literature, the BMPG performance evaluation is often based on the form

Table 4
Global performance comparison according to problem size. Results for the GPS, TS and GRASP-PR algorithms were taken from [24] where a K-7 Athlon at 1.2 GHz. was used.

	33 instances with $n = 30, ..., 199$					
	GPS	SA-DJ	TS	GRASP-PR	SA-δ β_b	SA-δ $Avg.$
Average β	31.424	29.364	23.333	22.515	22.061	22.621
Deviation	35.20%	56.20%	9.43%	2.27%	0.30%	3.83%
CPU sec.	0.003	2063.85	4.985	4.025	11.177	11.177
	80 instances with $n = 200, ..., 1000$					
	GPS	SA-DJ	TS	GRASP-PR	SA-δ β_b	SA-δ $Avg.$
Average β	156.375	164.588	100.775	99.425	94.800	95.699
Deviation	46.52%	221.71%	11.44%	6.28%	1.14%	1.97%
CPU sec.	0.111	26448.83	263.969	339.969	199.252	199.252

of averaged results over the whole set of the tested instances. Table 4 shows such a comparison of the best (β_b) and average ($Avg.$) results obtained by SA-δ with those reached by TS, GRASP-PR, GPS and SA-DJ. We indicate for each algorithm the average bandwidth over each instance subset along with the average CPU time in seconds and the average deviation from the best solutions found by applying all the heuristics (including SA-δ) to the same instance. Recall that the timings for GPS, TS and GRASP-PR are taken from [24] where a K-7 Athlon at 1.2 GHz. was used for the experiments. The results for SA-DJ are obtained by running the original source code in our computational platform.

From Table 4, one observes that SA-DJ is very slow. Our SA-δ uses a set of parameters, presented in Section 4, that speeds up the convergence and allows to reach better solutions in less CPU time. We can also observe that the performance of the classic GPS algorithm, though very fast, gives inferior results in comparison with the other heuristics. In particular, it has an average deviation several orders of magnitude larger than those obtained with SA-δ. For the subset of small instances the average deviation from the best known values for GRASP-PR (2.27%) is slightly smaller than that obtained by SA-δ $Avg.$ (3.83%), but considerably larger than the 0.30% obtained by SA-δ β_b. For the large instances, one observes a clear improvement in the average bandwidth obtained with our algorithm both in β_b and $Avg.$ (*i.e.*, 94.800 and 95.699 versus 99.425 for GRASP-PR). This comparison confirms clearly the competitive performance of SA-δ with respect to the state-of-the-art algorithms on the tested problem instances, specially on the large instances.

5.3 Discussions

It is well known that the parameters and some key features play an important role in the global performance of the SA algorithm. In this sense we have carried out some experiments to better understand these influences. For this experimentation a set of 12 structured instances were generated according to the model proposed in [7]. It consists of six different classes of graphs of increasing sizes including: grids, paths, cycles, binary trees, ternary trees and quaternary trees. This test suit was used consistently over all the experiments presented in the following subsections.

5.3.1 Influence of the Annealing Parameters

The chosen annealing schedule determines the degree of uphill movement permitted during the search process and is, thus, critical to the SA algorithm's performance. It has two important parameters: the initial temperature (T_i)

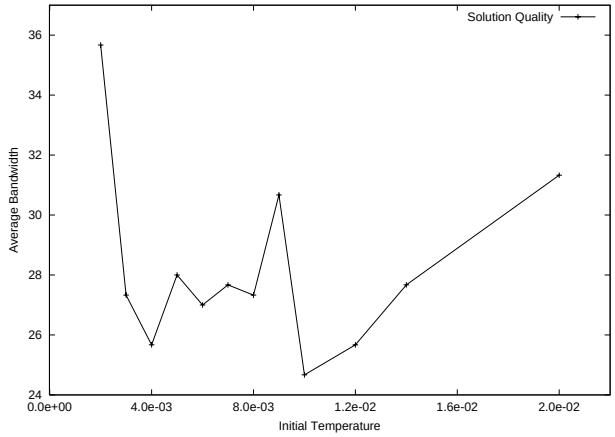


Fig. 5. Influence of T_i on the solution quality for the SA-δ algorithm. Results obtained on 20 executions at different initial temperatures over a set of 12 structured instances.

and the maximum number of neighboring solutions that can be generated at each temperature (NV_{max}). This point is confirmed in the following study.

In Fig. 5, the influence of the initial temperature (T_i) on the solution quality for the SA-δ algorithm is presented. The curve represents the average solution quality obtained over 20 executions at different initial temperatures. We can observe that the values that give good results are in the interval $4.0E^{-3} \leq T_i \leq 1.0E^{-2}$. This remark seems to be valid across the multiple problem instances that we have tested. The other important parameter is the maximum number of neighboring solutions that can be generated at each temperature ($NV_{\max}$). In our SA-δ implementation, it depends directly on the number of edges ($|E|$) of the graph and on the *moves factor* value (μ). In Fig. 6, the influence of μ on the average solution quality for the SA-δ algorithm is presented. The curve represents the average solution quality obtained over 20 executions at different values of μ. A second curve represents the average CPU time needed

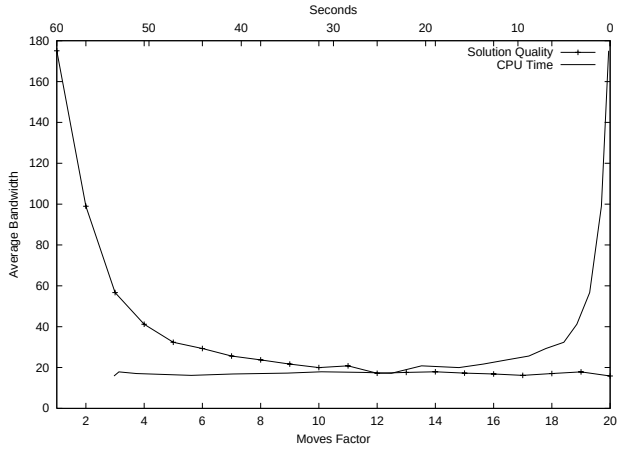


Fig. 6. Influence of μ on the solution quality for the SA-δ algorithm. Results obtained over 20 executions at different values of μ over a set of 12 structured instances.

to reach that solutions. We can observe from these curves that the best trade-off between solution quality and CPU time is obtained when μ equal 12. This graphic was created using data from our tuning experiments.

5.3.2 Influence of the Neighborhood Functions

The neighborhood functions is another critical element for the performance of any local search algorithm. In this study we have considered the following neighborhood functions:

- *Swap1*: Two labelings are neighbors if one can go from one to the other by exchanging the labels of two adjacent nodes in the graph.
- *Swap2*: Two labelings are neighbors if one can go from one to the other by exchanging the labels of any pair of nodes in the graph.
- *Rotation*: Two layouts are neighbors if one can go from one to the other by rotating the labels of any group of vertices in the graph according to the definition given in Subsection 3.2.

Experiments have been carried out to compare the performance of these three neighborhood functions. Fig. 7 shows the differences in terms of average solution quality for the SA-δ algorithm over 20 executions. For example, with the same test conditions, the neighborhood function *Rotation* allows SA-δ to get a bandwidth of 11.2 while *Swap1* and *Swap2* lead to a bandwidth of 17.5 and 16.3 respectively. Given this results it is clear that the rotation-based neighborhood function is much more powerful.

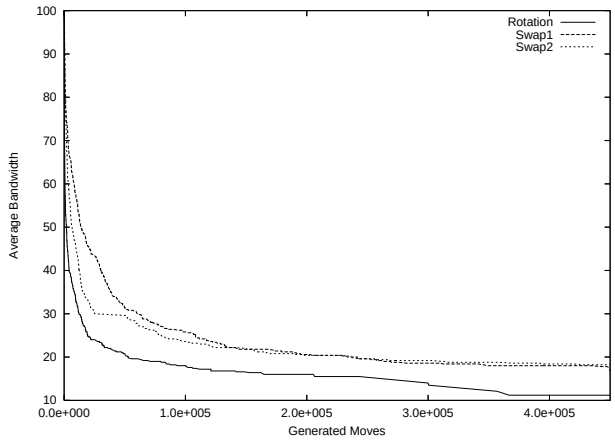


Fig. 7. Performance comparison of three neighborhood functions for SA-δ. Results obtained on 20 executions over a set of 12 structured instances.

5.3.3 Influence of the Evaluation Functions

The purpose of this experiment is to compare the new δ evaluation function and the classical β evaluation function. To do this, we use δ and β within

our SA algorithm presented in Section 3 (call these SA algorithms SA-δ and SA-β respectively) and test them on the set of 12 structured instances. These graphs were selected for this experiment because the optimum bandwidth is known for all of them.

Both SA-δ and SA-β were run 20 times on each instance and the results are presented in Table 5. In this table columns 1 to 3 show the name of the graph and the number of vertices and edges. Columns SA-β and SA-δ represent the average bandwidth for the 20 runs of the SA algorithm that uses the evaluation function β and δ respectively. The sixth and seventh columns show the best bandwidth obtained for each of the SA variants. Finally the last column presents the improvement obtained when the δ evaluation function was used.

The results presented in Table 5 show clearly that the SA that uses δ consistently has much better results for many classes of graphs than the one that uses β. We can observe an average improvement of 42% (see column Improvement). So we could conclude that δ is a better evaluation function than β.

In order to illustrate the behavior of SA-δ and SA-β in Fig. 8 the bandwidth reduction versus the number of generated moves, over the instance $Grid100$, is shown. In this figure it can be seen that the SA-δ reduces the bandwidth almost continuously while the SA-β gets stuck longer time, and the final bandwidth reached by SA-δ is significantly smaller than the bandwidth reached by SA-β. This behavior was valid across the multiple problem instances that we have tested.

Table 5
Comparison between two SA for the BMPG using the δ and β evaluation functions. Results obtained on 20 executions.

			Average		Best		
Graph	n	Edges	SA-β	SA-δ	SA-β	SA-δ	Improvement
Path100	100	99	10.0	1.2	10	1	90%
Path150	150	149	15.6	1.4	15	1	93%
Cycle100	100	99	10.6	2.2	10	2	80%
Cycle150	150	149	15.8	2.6	15	2	87%
TreeB63	63	62	8.0	7.0	8	7	13%
TreeB127	127	126	15.6	11.0	15	11	27%
TreeT40	40	39	7.0	7.0	7	7	0%
TreeT121	121	120	17.8	15.0	17	15	12%
TreeQ85	85	84	15.0	14.0	15	14	7%
TreeQ205	205	204	30.6	26.0	30	26	13%
Grid100	100	180	15.8	10.0	15	10	33%
Grid225	225	420	31.2	15.0	30	15	50%
Average					15.6	9.3	42%

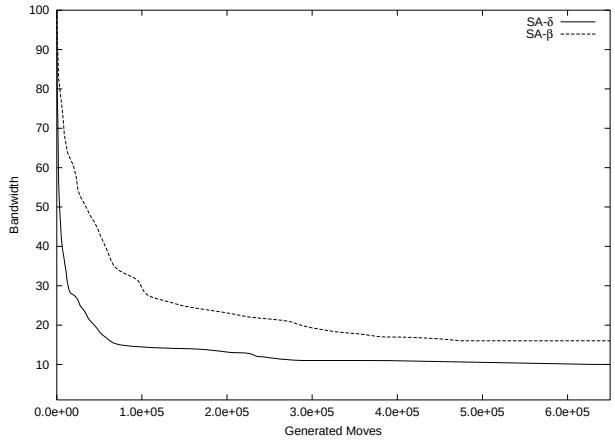


Fig. 8. SA-β and SA-δ best-so-far behavior comparison over the instance $Grid100$.

6 Conclusions

In this paper, we have introduced an improved Simulated Annealing algorithm (SA-δ). This algorithm integrates three key features that have a great impact in the heuristic search:

- The internal representation: it allows us to produces smooth changes in the configuration because of its intrinsical locality. Moreover, it has enabled the creation of other neighborhood functions which lead to good solutions.
- The evaluation function: it has two important features: a) It considers all the edges of the graph and b) It produces more equivalence classes with lower cardinality. The δ evaluation function orients better the search process with a smoother landscape and permits to find configurations where all the absolute differences between labels of adjacent vertices are minimized.
- The neighborhood function: it allows the SA-δ to better explore the search space. It presents a good trade-off between exploration and exploitation.

Extensive experimentation, on a set of 113 well-known benchmark instances of the literature, has been carried out to evaluate the performance of this SA algorithm. In these experiments our improved SA-δ algorithm has been compared with several state-of-the-art algorithms. These comparisons lead to the following main observations.

For the subset of small instances, the average solutions obtained by the GRASP-PR algorithm are slightly better than the average values ($Avg.$) attained by SA-δ (22.52 against 22.62), while SA-δ in average is 3.05% better than the TS algorithm (23.33); furthermore, the best results (β_b) found by SA-δ over 20 runs improve 11 out of 33 previous best known solutions.

Concerning the subset composed of large instances, the results show that in average even the worst quality solution (β_w) obtained by SA-δ (97.44) over 20

runs is better than that achieved by TS and GRASP-PR (100.78 and 99.43); while the average result ($Avg.$) reached by our algorithm (95.70) represents an improvement of 3.75% with respect to GRASP-PR. SA-δ shows to be able to improve on 46 out of 80 previous best known solutions when the best results of SA-δ over 20 runs are considered.

This study confirms once again that appropriated neighborhood functions and evaluation functions are indispensable for reaching good performance of a Simulated Annealing algorithm. It confirms also the importance to find a suitable tuning of parameters.

Finally, let us comment that although the importance of neighborhood functions is widely recognized, the role of evaluation functions is somewhat overlooked in the literature. We think that the research on more informative evaluation functions for combinatorial problems is certainly a very interesting topic to boost the performance of heuristic algorithms. Indeed, it is the evaluation function that guides a heuristic search process through a large and complex search landscape. It seems clear that it will be quite difficult to identify general rules for designing informative evaluation functions. For a given problem, it is indispensable to realize a deep analysis of the target problem and to integrate useful information into the searched evaluation function.

Acknowledgments. This work is supported by the CONACyT Mexico, the "Contrat Plan Etat Région" project COM (2000-2006) as well as the Franco-Mexican Joint Lab in Computer Science LAFMI (2005-2006). The authors would like to thank R. Martí who has kindly provided us with his test data and detailed results. The reviewers of the paper are greatly acknowledged for their constructive comments.

References

[1] E. Aarts and P. V. Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the IEEE ICCAD-85*, pages 206–208, Santa Clara, CA, 1985.

[2] M. Berry, B. Hendrickson, and P. Raghavan. Sparse matrix reordering schemes for browsing hypertext. *Lectures in Appl. Math.*, 32:99–123, 1996.

[3] L. Cavique, C. Rego, and I. Themido. Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, 50:608–616, 1999.

[4] P. Chinn, J. Chvatalova, A. Dewdney, and N. Gibbs. The bandwidth problem for graphs and matrices - a survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[5] G. D. Corso and G. Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.

[6] E. Cutchill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the ACM National Conference*, pages 157–172, New York, 1969.

[7] G. Dueck and J. Jeffs. A heuristic bandwidth reduction algorithm. *Journal of Combinatorial Mathematics and Computers*, 18:97–108, 1995.

[8] A. Esposito, M. F. Catallano, F. Malucelli, and L. Tarricone. A new matrix bandwidth reduction algorithm. *Operations Research Letters*, 23:99–107, 1999.

[9] M. Garey, R. Graham, D. Johnson, and D. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34:477–495, 1978.

[10] N. Gibbs, W. Poole, and P. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13:235–251, 1976.

[11] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65:223–253, 1996.

[12] E. Gurari and I. Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem. *Journal of Algorithms*, 5:531–546, 1984.

[13] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

[14] L. Harper. Optimal assignment of numbers to vertices. *Journal of SIAM*, 12(1):131–135, 1964.

[15] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *Proceedings of the IEEE ICCAD-86*, pages 381–384, Santa Clara, CA, 1986.

[16] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[17] D. Kratsch. Finding the minimum bandwidth of an interval graph. *Journal of Inform. Comput.*, 8:140–187, 1987.

[18] A. Lim, B. Rodrigues, and F. Xiao. Integrated genetic algorithm with hill climbing for bandwidth minimization problem. *Lecture Notes in Computer Science*, 2724:1594–1595, 2003.

[19] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[20] R. Livesley. The analysis of large structural systems. *Computer Journal*, 3(1):34–39, 1960.

[21] R. Martí. Personal communication. September 2004.

[22] R. Martí, M. Laguna, F. Glover, and V. Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):211–220, 2001.

[23] C. Papadimitriou. The NP-Completeness of the bandwidth minimization problem. *Journal on Computing*, 16:263–270, 1976.

[24] E. Piñana, I. Plana, V. Campos, and R. Martí. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153:200–210, 2004.

[25] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. An improved evaluation function for the bandwidth minimization problem. *Lecture Notes in Computer Science*, 3242:650–659, 2004.

[26] L. Smithline. Bandwidth of the complete k-ary tree. *Journal of Disc. Math.*, 142:203–212, 1995.

[27] E. Taillard. A statistical test for comparing success rates. In *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pages 110–1 – 110–5, 2003.

[28] J. Torres-Jimenez and E. Rodriguez-Tello. A new measure for the bandwidth minimization problem. *Lecture Notes in Artificial Intelligence*, 1952:477–486, 2000.

[29] J. M. Varanelli and J. P. Cohoon. A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Computers and Operations Research*, 26:481–503, 1999.