Information Sciences 303 (2015) 33–49

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs

Jose Torres-Jimenez [a,b,*], Idelfonso Izquierdo-Marquez [a], Alberto Garcia-Robledo [a], Aldo Gonzalez-Gomez [a], Javier Bernal [b], Raghu N. Kacker [b]

[a] CINVESTAV-Tamaulipas, Information Technology Laboratory, Km. 5.5 Carretera Cd., Victoria-Soto la Marina, 87130 Cd. Victoria, Tamps., Mexico
[b] National Institute of Standards and Technology, Gaithersburg, MD 20899-8910, USA

ARTICLE INFO

Article history:
Received 9 April 2014
Received in revised form 17 December 2014
Accepted 28 December 2014
Available online 16 January 2015

Keywords:
Bandwidth minimization
Simulated annealing
Combinatorial optimization
Meta-heuristic

ABSTRACT

The bandwidth minimization problem on graphs (BMPG) consists of labeling the vertices of a graph with the integers from 1 to n (n is the number of vertices) such that the maximum absolute difference between labels of adjacent vertices is as small as possible. In this work we develop a DRSA (Dual Representation Simulated Annealing) algorithm to solve BMPG. The main novelty of DRSA is an internal dual representation of the problem used in conjunction with a neighborhood function composed of three perturbation operators. The evaluation function of DRSA is able to discriminate among solutions of equal bandwidth by taking into account all absolute differences between labels of adjacent vertices. For better performance, the parameters of DRSA and the probabilities for selecting the perturbation operators were tuned by extensive experimentation carried out using a full factorial design. The benchmark for the proposed algorithm consists of 113 instances of the Harwell-Boeing sparse matrix collection; the results of DRSA included 31 new upper bounds and the matching of 82 best-known solutions (22 solutions are optimal). We used Wilcoxon signed-rank test to compare best solutions produced by DRSA against best solutions produced by three state of the art methods: greedy randomized adaptive search procedure with path relinking, simulated annealing, and variable neighborhood search; according to the comparisons done, the quality of the solutions with DRSA is significantly better than that obtained with the other three algorithms.

1. Introduction

The bandwidth minimization problem on graphs (BMPG) was originated at the Jet Propulsion Laboratory in 1962 when Harper [16] studied the problem of labeling the vertices of a hypercube in n dimensions with the integers from 1 to 2^n in such a way that summation, over all neighboring pairs of vertices, of the absolute differences of integers assigned to vertices is minimal. Harary [15] proposed the problem independently of Harper.

Let $G = (V, E)$ be a finite undirected graph with $|V| = n$ and $|E| = m$. A labeling τ of G is a bijective mapping from the vertices of G to the set $\{1, 2, \ldots, n\}$. The bandwidth of G for τ, written $B_\tau(G)$, is $B_\tau(G) = \max\{|\tau(i) - \tau(j)| : (i,j) \in E\}$. BMPG is the problem of finding a labeling τ^* such that the bandwidth of G for τ^* is as small as possible: $B_{\tau^*}(G) = \min\{B_\tau(G)\}\ \forall\tau$.

* Corresponding author at: CINVESTAV-Tamaulipas, Information Technology Laboratory, Km. 5.5 Carretera Cd., Victoria-Soto la Marina, 87130 Cd. Victoria, Tamps., Mexico.

E-mail addresses: jtj@cinvestav.mx, jnt5@nist.gov (J. Torres-Jimenez), iizquierdo@tamps.cinvestav.mx (I. Izquierdo-Marquez), algarcia@tamps.cinvestav.mx (A. Garcia-Robledo), ajgonzalez@tamps.cinvestav.mx (A. Gonzalez-Gomez), javier.bernal@nist.gov (J. Bernal), raghu.kacker@nist.gov (R.N. Kacker).

http://dx.doi.org/10.1016/j.ins.2014.12.041
0020-0255/© 2015 Elsevier Inc. All rights reserved.

In the adjacency matrix $A = (a_{ij})$ of graph $G = (V, E)$ each non-zero element a_{ij} of A is in a diagonal located $|i - j|$ positions away from main diagonal. The distance from the farthest non-zero element to main diagonal is the bandwidth of the matrix; the term bandwidth is used because the non- zero elements are located in a band with respect to main diagonal. The bandwidth of a matrix A can be reduced by simultaneous permutations of rows and columns of the adjacency matrix; in fact simultaneous exchange of rows i and j, and columns i and j is equivalent to a label exchange of nodes i and j in the graph G.

Bandwidth minimization is desirable for a wide range of applications, for instance see [24,3,12,2]. The BMPG is NP-Complete [25], and a number of exact, greedy, and meta-heuristic algorithms have been developed to solve it. The most important exact algorithms are those developed by Del Corzo and Manzini [10]; Caprara and Salazar-González [5]; Martí et al. [20]; and Cygan and Pilipczuk [8,9]. In the group of greedy algorithms the best known are the Gibbs, Poole, and Stockmeyer (GPS) algorithm [13]; and the Cuthill–McKee (CM) algorithm [6]. In the class of meta-heuristic algorithms we found genetic programming [17], genetic algorithm [19], ant colony [18], tabu search [21], simulated annealing (SA) [11,27], greedy randomized adaptive search procedure with path relinking (GRASP-PR) [26], scatter search–tabu search [4], node-shift with hill climbing [19], and variable neighborhood search (VNS) [23].

In this work we propose a meta-heuristic algorithm called DRSA (Dual Representation Simulated Annealing) to solve BMPG. The novelty of this algorithm is the use of two internal representations of solutions for the problem; given a permutation $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$ of the integers from 1 to n, where n is the number of vertices of the graph, the first representation considers element τ_i as the label assigned to vertex i, and the second representation interprets τ_i as the vertex to which label i is assigned. In conjunction with the two internal representations we use a neighborhood function composed of three perturbation operators; two of the three operators work over the first internal representation and the third operator works over the second one; the combined effect of the three operators is more powerful than the effect of each operator alone. The evaluation function of DRSA is a modified version of the evaluation function introduced in [27].

To change a solution of the BMPG, one of the three perturbation operators of the neighborhood function is applied. The operator to be used is selected according to certain precomputed probabilities determined using a full factorial tuning process that will be explained in Section 4. In the same tuning process values for parameters of the simulated annealing algorithm, specifically for initial temperature, cooling rate, and final length of the Markov chain are determined.

DRSA was tested with a well-known benchmark of 113 instances of Harwell-Boeing sparse matrix collection [22]; we improved 31 best-known solutions and matched 82 upper bounds (22 of them are optimal). According to Wilcoxon signed-rank test, DRSA outperforms the methods GRASP-PR [26], SA [27], and VNS [23], with respect to the quality of the solutions.

The remainder of this document is organized as follows: Section 2 presents a review of some of the most relevant methods for bandwidth minimization; Section 3 presents DRSA; Section 4 describes the process for tuning the parameters of DRSA using a full factorial design; Section 5 shows the results of applying DRSA to the 113 instances of the benchmark; and Section 6 is a summary of main contributions of this paper.

2. Relevant related work

In this section some relevant algorithms for solving BMPG are briefly described. The reviewed algorithms are grouped in three categories: exact, greedy, and meta-heuristic. In Table 1 we summarize significant information about algorithms including a short note about each one.

2.1. Exact algorithms

Del Corzo and Manzini [10] introduced two exact algorithms for solving BMPG. First one is the MB-ID algorithm (Minimum Bandwidth by Iterative Deepening), and second one is the MB-PS algorithm (Minimum Bandwidth by Perimeter Search).

MB-ID algorithm is based on the concept of upper partial ordering (UPO): for a graph G with vertex set $V = \{1, 2, \ldots, n\}$, a UPO of length $k < n$ is an injective function from $\{1, 2, \ldots, k\}$ to $\{1, 2, \ldots, n\}$. A UPO assigns the integers $1, 2, \ldots, k$ to k vertices of G, say $\{v_{j_1}, v_{j_2}, \ldots, v_{j_k}\}$. MB-ID searches for a b-bandwidth solution, where b is initially a lower bound of the bandwidth, i.e., $b \leqslant B_{\tau^*}(G)$. If this search fails, the algorithm searches for a $(b + 1)$-bandwidth solution and so on. In each search the algorithm begins with a UPO of length zero, and by means of a depth first search, vertices are added to the current UPO.

Caprara and Salazar-González [5] proposed an enumerating scheme for computing $B_{\tau^*}(G)$ based on the construction of partial layouts, where a partial layout of a subset of vertices $S \subset V$ is a function $\tau : S \rightarrow \{1, 2, \ldots, n\}$ such that $\tau_u \neq \tau_v$ for $u, v \in S$ and $u \neq v$. The basic idea is determining the minimum reachable bandwidth knowing the current partial layout. Based on which labels are already assigned, the algorithm estimates the minimum bandwidth that can be reached.

Martí et al. [20] introduced another exact algorithm for solving BMPG. This algorithm uses a greedy randomized adaptive search procedure (GRASP) [26] for determining initial upper bound of the bandwidth. Suppose GRASP finds a bandwidth of size k; then the exact algorithm begins by searching a bandwidth of size $k - 1$. If a solution is found the algorithm searches for a bandwidth of size $k - 2$, and so on. When the algorithm does not find a solution having a bandwidth of size $k - i$ it terminates and reports the optimal bandwidth of size $k - i + 1$. The initial solution provided by the GRASP algorithm allows for the reduction of the search space that is explored.

Table 1
Summary of significant information about approaches to solve BMPG.

Class	Authors	Main idea
Exact	Del Corzo and Manzini [10]	Creation of Upper Partial Orderings (UPO)
Exact	Caprara and Salazar-González [5]	Construction of partial layouts
Exact	Martí et al. [20]	Search starts with a solution obtained using a greedy randomized adaptive search procedure approach
Exact	Cygan and Pilipczuk [8]	Partitioning of labels into segments, then computation of solution for each segment and verification of full solution
Greedy	Gibbs, Poole and Stockmeyer (GPS) algorithm [13]	Construction of a level structure such that every edge has its endpoints either in the same level or in consecutive levels, using pseudo-peripheral vertices separated by a distance equal to the graph diameter
Greedy	Cuthill–McKee (CM) algorithm [6]	Construction of a level structure such that every edge has its endpoints either in the same level or in consecutive levels, and then using the vertices in increasing order of their degrees
Greedy	Cuthill–McKee algorithm in reverse order (RCM) [6]	Construction of a level structure such that every edge has its endpoints either in the same level or in consecutive levels, and then using the vertices in decreasing order of their degrees
Meta-heuristic	Piñana et al. [26]	Greedy randomized adaptive search procedure using Tabu Search and path relinking to expand the exploration
Meta-heuristic	Dueck and Jeffs [11]	Simulated annealing that uses as perturbation the exchange of two labels
Meta-heuristic	Rodriguez-Tello et al. [27]	Simulated annealing that uses as perturbation the rotation of the labeling and whose evaluation function uses all differences between labels of adjacent vertices
Meta-heuristic	Mladenović et al. [23]	Variable neighborhood search that controls the exploration of increasingly distant neighbors of a solution

Cygan and Pilipczuk [8] developed an exact algorithm with $O(5^n)$ running time for solving BMPG. A *b-ordering* is an ordering of vertices of the graph such that bandwidth is of size at most b. Algorithm works by partitioning the positions $\{1, 2, \ldots, n\}$ into $\lceil \frac{n}{b+1} \rceil$ segments of size $b + 1$, except possibly for the last segment whose size is $n \bmod (b + 1)$. Next, algorithm generates several assignments of vertices to segments, and checks each assignment for compatibility with the *b-ordering*. An improvement of this algorithm with a running time of $O(4.83^n)$ was reported in [9].

2.2. Greedy algorithms

Three of the best known greedy algorithms to solve BMPG are the Gibbs, Poole and Stockmeyer (GPS) algorithm [13]; Cuthill–McKee (CM) algorithm [6]; and improvement of the CM algorithm: reverse Cuthill–McKee (RCM) algorithm. The common core between GPS, CM, and RCM algorithms is a breadth-first search (BFS) procedure that partitions vertices in level structures such that every edge has its endpoints either in the same level or in consecutive levels. This level organization is then used to label the vertices according to their level number.

CM algorithm begins by choosing a starting vertex of minimum degree. After that, it constructs a level structure rooted at the starting vertex and sorts vertices in increasing order of their degrees. From an implementation perspective, a queue is used to insert/delete neighbors of the root in increasing order of their degrees and a vector p is used to store consecutively the new labels. The level-by-level renumbering is a consequence of the order in which vertices are discovered by BFS procedure. The only difference between CM and RCM algorithms is that RCM algorithm reverses the vector p. GPS algorithm is not very different: it finds pseudo-peripheral vertices separated by a distance equal to graph diameter; as in CM and RCM algorithms, GPS algorithm renumbers vertices level by level.

2.3. Meta-heuristic algorithms

Meta-heuristic algorithms allow one to find BMPG solutions usually better than those produced by greedy algorithms but certainly in more time. These algorithms represent a solution for the problem with a permutation of the labels of vertices, and have different ways of measuring the goodness of solutions.

2.3.1. GRASP with path relinking

Piñana et al. [26] proposed a greedy randomized adaptive search procedure (GRASP) combined with a path relinking (PR) strategy to solve BMPG. In the GRASP phase the method generates a set of elite solutions using one of five constructive methods C1, …, C5. The phase begins with the creation of a list of unlabeled vertices U, which initially is equal to the vertex set V; after that, first vertex is selected randomly from U and a random label is assigned to it. In the following steps the next vertices to be removed from U are selected randomly from a list formed by all vertices of U that are adjacent to at least one labeled vertex. The selected vertex is labeled with the closest number not yet assigned to a previously selected vertex. The method is adaptive because after selecting and relabeling one vertex it updates some relevant information to be used in the next construction step. A procedure based on the tabu search algorithm [21] completes the GRASP phase.

Finally, PR strategy tries to improve the solution. In this phase the algorithm generates new solutions by exploring trajectories that connect the elite solutions from the GRASP phase. Starting from one of these solutions, called the initial solution, the algorithm generates a path in the neighborhood space that leads to better solutions.

2.3.2. Simulated annealing

A simulated annealing (SA) algorithm to solve BMPG was used first by Dueck and Jeffs [11]. In this algorithm the neighborhood function interchanges a pair of distinct labels (a *move*). If a move generates a solution with a cost lower than the cost of the current solution the move is accepted and the current solution is updated. Otherwise, the move is accepted with probability $P(\Delta E) = e^{-\Delta E/T}$ where ΔE is the increment in cost that would result from the prospective move, and T is the current temperature.

Rodriguez-Tello et al. [27] introduced a neighborhood function called *rotation*. For a labeling τ the rotation between two labels $\tau(i)$ and $\tau(j)$, with $i < j$, can be expressed as a product (or sequence) of $j - i$ swaps: rotation$(\tau(i), \tau(j))$ = swap$(\tau(i), \tau(j)) *$ swap$(\tau(i), \tau(j-1)) * \cdots *$ swap$(\tau(i), \tau(i+1))$. This compound move gives to the algorithm the capacity to explore extensively the neighborhood of current solution. Another important characteristic of this algorithm is its evaluation function called δ based on the bandwidth of the permutation being evaluated and on the number of absolute differences with value $x \in \{1, 2, \ldots, n-1\}$ induced by the permutation.

2.3.3. Variable neighborhood search

Mladenović et al. [23] developed an algorithm based on variable neighborhood search (VNS) meta-heuristic in order to solve BMPG. This meta-heuristic explores increasingly distant neighbors of a solution. A main loop performs the following three steps: shaking, local search, and neighborhood change. First, the neighborhood index k is initialized with k_{min} (this is the size of the neighborhood). Second, in the shaking phase the meta-heuristic generates a solution y within current neighborhood. Using the solution y, a local search method produces a local optimum z. Third, if the solution z is better than the solution x, then it replaces x, and k is set to k_{min}; otherwise neighborhood is expanded incrementing k while $k < k_{max}$. When $k \geqslant k_{max}$, k is set to k_{min} again. VNS algorithm improved 42 previous best-known solutions of the benchmark of 113 instances taken from the Harwell-Boeing sparse matrix collection [22].

3. The Dual Representation Simulated Annealing (DRSA) algorithm

This section presents our proposed method of solution: the Dual Representation Simulated Annealing (DRSA). DRSA uses the simulated annealing (SA) meta-heuristic, but it is different from previously reported SA algorithms as previous implementations only use one internal representation of the problem whereas DRSA uses two. Moreover, DRSA has a neighborhood function composed of three perturbation operators, which give DRSA a balance between exploration and exploitation of the search space.

3.1. Structure of DRSA

Simulated annealing heuristic was inspired by the analogy of the process of heating and cooling a metal to obtain a highly stable crystalline structure. This process consists of increasing temperature to a maximum value at which a metal melts, and then decreasing it until the melted metal reaches its ground state; in that state the particles of the metal get arranged such that the energy of the system is minimal [1].

There are three parameters for simulating temperature scheduling: initial temperature T_0, final temperature T_f and cooling rate α $(0 < \alpha < 1)$. At the beginning of the algorithm current temperature T is initialized at T_0, but as execution proceeds it is repeatedly reduced using $T = \alpha T$ until it reaches its final value T_f. To simulate the change of states in the metal the algorithm maintains a current state x that has energy E_x. The next state y with energy E_y is obtained by perturbing the current state x. If E_y is less than E_x then y becomes the new current state; otherwise state y is accepted with probability $e^{-(E_y - E_x)/k_B T}$ (T is the current temperature and k_B is the Boltzmann constant). The SA requires a fourth parameter L (called the length of Markov chain) that defines the number of perturbations performed at the same temperature T.

The DRSA algorithm is summarized in Algorithm 1. Algorithm receives three input parameters: initial temperature T_0, cooling rate α, and final length of Markov chain L_f. At the beginning final temperature T_f is defined as 1×10^{-7}, current length of Markov chain L is defined as 40, and current temperature T is initialized at T_0.

Algorithm 1. DRSA algorithm

```
Data: $T_0, \alpha, L_f$
Result: The best solution found $w$.
$T_f \leftarrow 1 \times 10^{-7}$;
$L \leftarrow 40$;
$T \leftarrow T_0$;
$\gamma \leftarrow$ compute-increment-factor($T_0, T_f, L, L_f$);
$x \leftarrow$ initial-random-solution();
$w \leftarrow x$;
while $T > T_f$ do
    $improvement \leftarrow$ false;
    for $i \leftarrow 1$ to $L$ do
        $y \leftarrow g(x)$;
        if $f(y) < f(x)$ then
            $x \leftarrow y$;
        else if $random(0,1) < e^{-(f(y)-f(x))/T}$ then
            $x \leftarrow y$
        end
        if $f(x) < f(w)$ then
            $w \leftarrow x$;
            $improvement \leftarrow$ true;
        end
    end
    if $improvement$ = false then
        $T \leftarrow T * \alpha$;
        $L \leftarrow L * \gamma$;
    end
end
return w;
```

In the fourth line of DRSA algorithm, γ value is computed. Its purpose is to increment the length of Markov chain so that when T reaches its final value T_f, L also reaches its final value L_f. Thus, the value of γ depends on the total number of temperature reductions r made in the algorithm. If r is the total number of temperature reductions then $T_f = \alpha^r T_0, r = \frac{\log(T_f) - \log(T_0)}{\log(\alpha)}$, and therefore $\gamma = \exp\left(\frac{\log(L_f) - \log(L)}{r}\right)$.

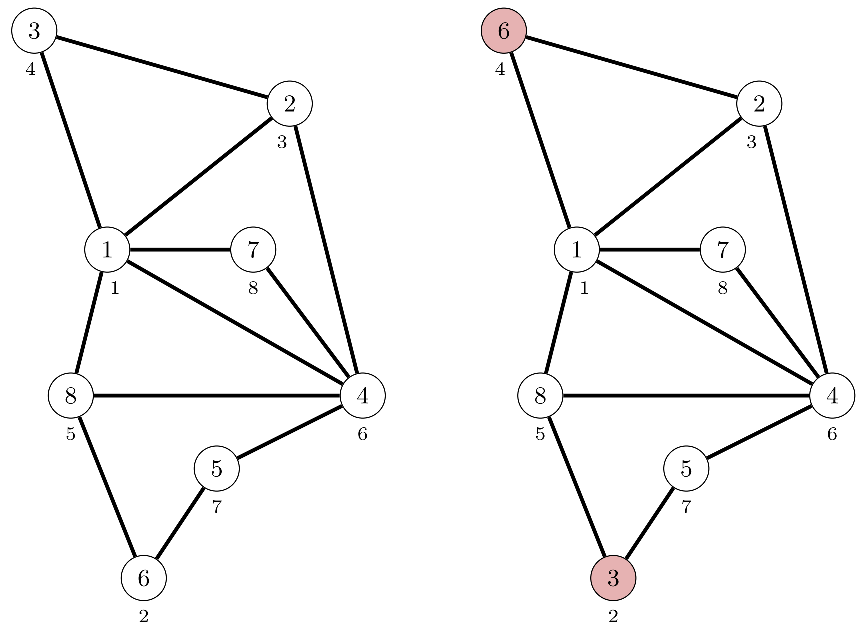
Basically DRSA consists of a *while* loop, which is executed while current temperature T does not reach its final value $T_f = 1 \times 10^{-7}$, and an inner *for* loop that performs L perturbations at the current temperature. Right before the inner *for* loop starts, the value of the variable *improvement* is set to **false** and right after it ends the value of *improvement* is tested. During the latter if *improvement* is **false** then current temperature T is decreased by α and current length L of Markov chain is incremented by γ; but if *improvement* is **true** then T and L remain unchanged; the idea behind this is to modify T and L only if the global best solution was not improved in the recently completed Markov chain.

Two main components of the algorithm are neighborhood function g and evaluation function f. The neighborhood function g perturbs the current solution x producing a new solution $y = g(x)$. If the cost $f(y)$ of the new solution is better than the cost of the current solution x, then y is accepted as the new current solution; otherwise y becomes current solution with probability $e^{-(f(y)-f(x))/T}$.

3.2. Internal dual representation

Given a graph with n vertices, BMPG consists of labeling the vertices of the graph with the integers $1, 2, \ldots, n$ such that the bandwidth of the graph is as small as possible. DRSA uses permutations of the set $\{1, 2, \ldots, n\}$ to represent BMPG solutions, but since we can denote the n vertices of the graph by $1, 2, \ldots, n$ respectively we have two possible interpretations of a permutation $\tau = (\tau_1, \tau_2, \ldots, \tau_n) : \tau_i$ as the label of vertex i; and τ_i as the vertex with label i. In order to distinguish the two interpretations of a permutation τ, we will use $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, where π_i represents the label of vertex i, and $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$, where ρ_i represents the vertex to which label i is assigned.

Each of the three perturbation operators is defined over one of the two internal representations, however every time one representation is modified we propagate this change to the other representation in order to maintain the consistency of the two representations. The three perturbation operators have as a basic operation the exchange of two elements in the representation over which they are defined. Suppose that we are working with the π representation and elements at positions i



$$\pi = (1,6,2,3,8,4,5,7) \qquad \pi = (1,3,2,6,8,4,5,7)$$
$$\rho = (1,3,4,6,7,2,8,5) \qquad \rho = (1,3,2,6,7,4,8,5)$$

Fig. 1. REX exchanges labels of two vertices i and j selected randomly. In the figure numbers inside circles are labels assigned to vertices, and numbers below circles are numbers of vertices. In this case, labels of vertices 2 and 4 were exchanged by the *REX* operator. The state of π and ρ representations is shown before (left) and after (right) application of *REX*.

and j of π were exchanged ($\pi_i \leftrightarrow \pi_j$); to propagate this exchange in ρ the elements of ρ at positions π_i and π_j are exchanged ($\rho_{\pi_i} \leftrightarrow \rho_{\pi_j}$). Similarly, an exchange of elements at positions i and j of ρ ($\rho_i \leftrightarrow \rho_j$) is propagated in π by exchanging its elements at positions ρ_i and ρ_j ($\pi_{\rho_i} \leftrightarrow \pi_{\rho_j}$).

The reason for using two representations π and ρ is that the neighborhood function is composed of operators that work over the π representation and operators that work over the ρ representation and this way DRSA can have a balance of exploration and exploitation of the BMPG search space.

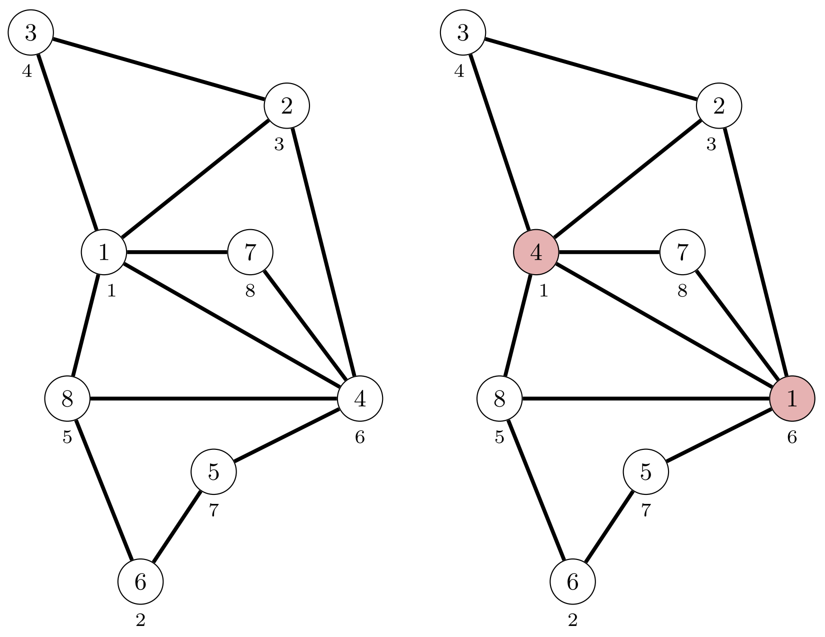
3.3. Neighborhood function

DRSA uses neighborhood function g to perturb the current solution and obtain a new solution. The algorithm evaluates the new solution using the evaluation function f and determines whether the new solution replaces the current one. As previously stated, the neighborhood function consists of three perturbation operators:

- First operator is *Random Exchange Operator* (*REX*) that works over π representation. This operator selects randomly two positions i and j of π and exchanges their contents.
- Second operator is *Neighbor Exchange Operator* (*NEX*) that also works over π representation. This operator selects randomly a vertex i of the graph and exchanges its label with the label of one of its neighbors j also selected randomly; since the indices of π represent the vertices of the graph, the *NEX* operator exchanges the contents of positions i and j of π.
- Third operator is *Rotation Operator* (*ROT*), introduced in [27], that works over ρ representation. This operator selects two positions i and j of ρ with $i < j$; then, element at position i is moved to position j, and elements between positions $i+1$ and j are shifted one position to the left. *ROT* operator makes a smooth change in the affected vertices as each vertex from ρ_i to ρ_{j-1} changes its label from i to $i+1$, with the only big change occurring for vertex ρ_j as it changes its label from j to i.

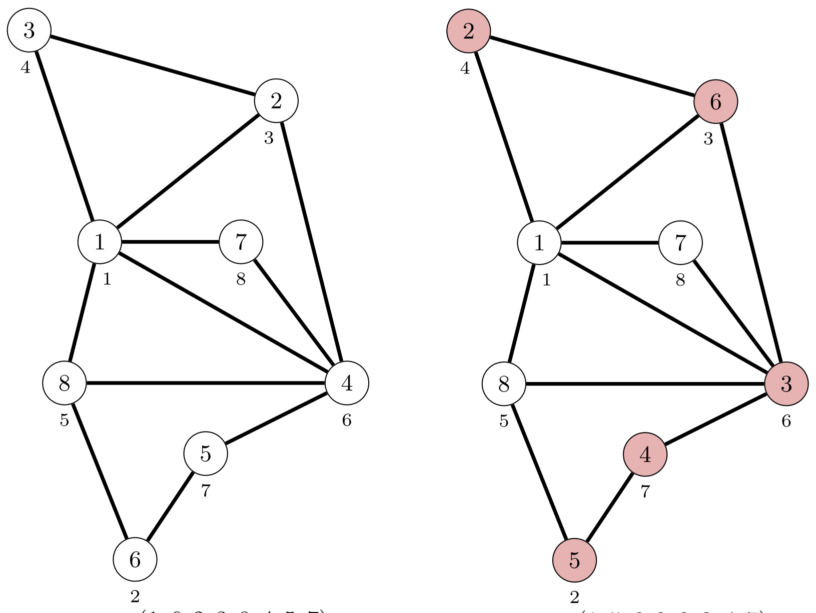
In a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, the absolute differences between labels of adjacent vertices varies from 0 to $n-1$ (0 occurs for reflexive edges). Next we analyze the three perturbation operators in greater depth taking into account the number of absolute differences affected by one application of each operator, the magnitude of the change in these differences, and the number of possible applications of the operator.

REX exchanges the labels π_i and π_j of two random vertices i and j. Let $\lambda = m/n$ be the average degree of the vertices in the graph. One label exchange affects on average 2λ absolute differences when vertices i and j are not adjacent, and $2\lambda - 1$ absolute differences when vertices i and j are adjacent. Now, for each edge incident to vertex i or vertex j, the magnitude of the change of its value (assuming the other vertex of the edge is vertex k) is: $||\pi_k - \pi_i| - |\pi_k - \pi_j|| = |2\pi_k - \pi_i - \pi_j|$ given that difference $|\pi_k - \pi_i|$ is replaced by difference $|\pi_k - \pi_j|$ or difference $|\pi_k - \pi_j|$ is replaced by difference $|\pi_k - \pi_i|$ after



$$\pi = (1,6,2,3,8,4,5,7) \qquad \pi = (4,6,2,3,8,1,5,7)$$
$$\rho = (1,3,4,6,7,2,8,5) \qquad \rho = (6,3,4,1,7,2,8,5)$$

Fig. 2. The neighbor exchange operator *NEX* exchanges labels of neighbor vertices *i* and *j*. In the figure labels of neighbor vertices 1 and 6 were exchanged by one application of the *NEX* operator. The state of the π and ρ representations is shown before (left) and after (right) the application of the operator.



$$\pi = (1,6,2,3,8,4,5,7) \qquad \pi = (1,5,6,2,8,3,4,7)$$
$$\rho = (1,3,4,6,7,2,8,5) \qquad \rho = (1,4,6,7,2,3,8,5)$$

Fig. 3. *ROT* moves the element at position *i* of ρ to position *j*, and shifts the elements $i+1, i+2, \ldots, j$, one position to the left. In this case $i = 2$ and $j = 6$, so $\rho_2 = 3$ is moved to position $j = 6$, and the elements $\rho_3 = 4, \rho_4 = 6, \rho_5 = 7, \rho_6 = 2$ are shifted one position to the left. The state of the π and ρ representations is shown before (left) and after (right) the application of the operator.

exchanging labels of vertices *i* and *j*. *REX* can be applied over any pair of vertices, without considering how close or far apart they are in the graph, and thus the number of possible applications of *REX* is $\binom{n}{2}$. Fig. 1 shows the use of the *REX* operator.

The neighbor exchange operator (*NEX*) is a specialized version of *REX* operator. *NEX* operator takes two neighbor vertices *i* and *j* and exchanges their labels π_i and π_j. On average, the number of absolute differences affected by one application of the

Table 2
Number of possible applications of the *ROT* operator as function of *r*.

r	Number of possible applications
1	$n-1$
2	$(n-1)+(n-2)=2n-3$
3	$(n-1)+(n-2)+(n-3)=3n-6$
...	...
k	$(n-1)+\cdots+(n-k)=k\cdot n-\frac{k(k+1)}{2}$

operator is $2\lambda-1$, and the magnitude of the change in the values of the edges incident to vertex i or vertex j is equal to (again assuming that the other vertex of the edge is vertex k): $||\pi_k-\pi_i|-|\pi_k-\pi_j||=|2\pi_k-\pi_i-\pi_j|$.

All edges affected by *NEX* are connected to edge (i,j), so every application of *NEX* involves vertices and edges that are located in the same region of the graph, in contrast with *REX* that can modify very distant regions of the graph on every application.

It is worth mentioning that the work done by *NEX* can also be realized by *REX*, but the probability $m/\binom{n}{2}$ that vertices i and j are neighbors in the graph is low, especially for sparse graphs. Fig. 2 shows the use of *NEX* operator.

ROT was introduced in [27]. If i and j are the bounds of *ROT* then this operator can be decomposed into a sequence of $|i-j|$ exchanges of adjacent positions in the ρ representation. The first exchange is between the elements at positions i and $i+1$, the second exchange is between the elements at positions $i+1$ and $i+2$, and the last exchange is between the elements at positions $j-1$ and j. This way, the element at position i is moved to position j and the elements in positions $i+1, i+2, \ldots, j$ are moved one position to the left. Fig. 3 gives an example of applying *ROT*.

On average, one application of *ROT* affects $(j-i+1)\lambda$ absolute differences, as $j-i+1$ vertices change their labels. This number is greater than the absolute differences affected by the other two operators, but the magnitude of the change is smaller than with those operators. Since *ROT* works over ρ representation the magnitude of the change in the affected differences is of one unit, except for the differences defined by the edges incident to vertex ρ_j. The vertices with labels $i, i+1, \ldots, j-1$ get the labels $i+1, i+2, \ldots, j$ respectively, so the magnitude of the change in the differences defined by the edges incident to these vertices is of one unit. The vertex with label j gets the label i and so the magnitude of the change in the affected differences is of $j-i$.

In this work i and j in *ROT* operator satisfy $1 \leqslant |i-j| \leqslant 5$. For this, a random value between 1 and 5 is selected and assigned to a variable r. Then the position i is selected randomly from the set $\{1, 2, \ldots, n-r\}$, and position j is set to $i+r$. The restriction $1 \leqslant |i-j| \leqslant 5$ is for keeping the value of $|i-j|$ small since then *ROT* does more exploitation as less absolute differences are affected. This is in contrast with the situation in which the value of $|i-j|$ is large so that *ROT* must then do more exploration as more absolute differences are affected and the magnitude of the change in the absolute differences defined by the vertex that changes its label from j to i is increased. Another reason to restrict the bounds of *ROT* is that one application of this operator requires more computational work than one application of the other two operators.

Let r be the maximum separation between the limits i and j of *ROT*; the number of possible applications of *ROT* is a function of r. For $r=1$ there are $n-1$ possible applications, where n is the number of vertices in the graph. Table 2 shows the number of possible applications of the operator as a function of r.

In every call of the neighborhood function g one of the three operators is applied to perturb the current solution. Each of the operators *REX*, *NEX*, and *ROT* has a probability value of being selected, values denoted by PROBABILITY_REX, PROBABILITY_NEX, and PROBABILITY_ROT, respectively. To select an operator the function g gets a random number $0 \leqslant p \leqslant 1$ and based on the value of p the function applies one of the three operators to the current solution x, as shown in Algorithm 2. In Section 4 we explain the process for determining probabilities for the operators that maximize the effectiveness of g.

Algorithm 2. The neighborhood function g.

```
Data: $x$
Result: Result $w$ of applying one of the three perturbation operators to current solution $x$.
$a \leftarrow$ PROBABILITY_REX;
$b \leftarrow$ PROBABILITY_NEX;
$c \leftarrow$ PROBABILITY_ROT;
$p \leftarrow$ random-number$(0,1)$;
if $p \leqslant a$ then
    $w \leftarrow$ REX$(x)$;
else if $p \leqslant a+b$ then
    $w \leftarrow$ NEX$(x)$;
else
    $w \leftarrow$ ROT$(x)$;
end
return $w$;
```

3.4. Evaluation function

This section describes the DRSA evaluation function f. The choice of this function was inspired by the evaluation function introduced in [27]. A main feature of this function is that it takes into account all the absolute differences between labels of adjacent vertices in order to discriminate among solutions with the same bandwidth.

For an undirected graph of n vertices there are $2^{n(n+1)/2}$ possible adjacency matrices. Each of these matrices has a bandwidth $\beta \in \{1, 2, \dots, n-1\}$, so the $2^{n(n+1)/2}$ matrices can be partitioned into n equivalence classes by putting matrices M_1 and M_2 in the same equivalence class if and only if they have equal bandwidth. Average cardinality of these equivalence classes is $\frac{2^{n(n+1)/2}}{n}$.

One possible evaluation function is the bandwidth β of the solution being evaluated. A solution τ is better than another solution τ' if $B_\tau(G) < B_{\tau'}(G)$. However, an evaluation function based only on bandwidth of the solution makes the algorithm susceptible to being trapped in plateaus, as the algorithm cannot differentiate among solutions that have same bandwidth. A better evaluation function should discriminate among such solutions in order to escape plateaus.

Consider a solution $\tau = (\tau_1, \tau_2, \dots, \tau_n)$. Let d be the vector of size n whose i-th element d_i $(0 \leqslant i \leqslant n-1)$ is the number of absolute differences equal to i induced by the labeling τ. The interpretation of each d_i in the adjacency matrix of the graph is the number of non-zero elements in a diagonal located i positions away from main diagonal.

For $i = 0, 1, \dots, n-1$, the number of absolute differences equal to i is at most $n-i$; for example, there can be at most $n-1$ differences equal to 1. This way $0 \leqslant d_i \leqslant n-i$ for each d_i so that the number of possible values for each d_i is $n-i+1$.

Let v be the vector of size n such that v_i $(0 \leqslant i \leqslant n-1)$ is the number of possible values that d_i can have; so $v = (n+1, n, \dots, 2)$. With vectors d and v we define a positional numbering system of variable base. Under this system, vector d is interpreted to represent a number with respect to the base, a number whose digits are the components of d, with the number of possible values for the positions of d given by v.

In a positional numbering system with fixed base, the positional value of the digit at position i (from right to left) is the base raised to $i-1$. This generalizes to a variable base numbering system as the product of the number of possible values for each of the last $i-1$ digits. Thus, the positional value for each d_i is $(n-i)!$. The number of possible values that this positional numbering system can represent is $\sum_{i=1}^{n} i \cdot i! = (n+1)! - 1$ and given that adjacency matrices with the same vector d belong to the same equivalence class, the average cardinality of the classes is $\frac{2^{n(n+1)/2}}{(n+1)!-1}$. This average is much smaller than the average cardinality of the equivalence classes based only on the bandwidth of the matrices.

It is desirable to have an evaluation function composed of two parts: an integer part (the bandwidth β) and a fractional part (with values in the rank 0 to 1, not including 1). This way the fractional part enables us to discriminate among solutions of equal bandwidth. One way to compute the fractional part is to transform the vector d into an integer z $(0 \leqslant z < (n+1)! - 1)$ using the positional numbering system of variable base described above and then compute $z = z/(n+1)!$ $(0 \leqslant z < 1)$, see Algorithm 3. Unfortunately during this computation z and $(n+1)!$ can be very large making this computation unfeasible. However not all is lost since the way in which z is computed and results reported by Cygan and Pilipczuk [7] bring up the idea of a computation using continued fractions in terms of d and v, a computation that does not involve huge integer values and still maps vector d to a normalized value δ $(0 \leqslant \delta < 1)$:

$$\delta = \cfrac{\cfrac{\begin{matrix}\cfrac{\cfrac{\frac{d_0}{v_0}+d_1}{v_1}+d_2}{v_2}+d_3 \\ \vdots \end{matrix}}{\ddots}\;\; \frac{v_{n-3}}{v_{n-2}} + d_{n-1}}{v_{n-1}};$$

this computation is implemented in Algorithm 4.

Algorithm 3. Mapping of vector d to a normalized z value $(0 \leqslant z < 1)$.

```
Data: $d, v$
Result: $z, 0 \leqslant z < 1$.
$z \leftarrow d_0$;
for $i \leftarrow 1$ to $n-1$ do
    $z = z * v_i + d_i$;
end
$z = z/(n+1)!$
```

Algorithm 4. Mapping of vector d to a normalized δ value ($0 \leqslant \delta < 1$).

```
Data: $d, v$
Result: $\delta, 0 \leqslant \delta < 1$.
$\delta \leftarrow 0$;
for $i \leftarrow 0$ to $n - 1$ do
    $\delta = (\delta + d_i)/v_i$;
end
```

Using β ($1 \leqslant \beta \leqslant n - 1$) and δ ($0 \leqslant \delta < 1$) the evaluation function f for DRSA is $f = \beta + \delta$ where β is the bandwidth of the solution and δ is the fractional part corresponding to vector d. The inclusion of δ in the evaluation function permits discrimination among solutions with same bandwidth. However, since using zero differences $d_{\beta+1}, \ldots, d_{n-1}$ in Algorithm 4 can produce very small values of δ, Algorithm 4 has been implemented to consider only the differences from d_0 to d_β.

4. Tuning of parameters

As shown in Algorithm 1, DRSA requires three parameters: initial temperature T_0, cooling rate α, and final length of the Markov chain L_f. For better performance, we established the values for these parameters through a full factorial tuning process. In addition, we also tuned the probabilities of being selected of each perturbation operator.

We carried out the tuning process using a full factorial design for five factors:

1. Initial temperature T_0.
2. Cooling rate α.
3. Final length of the Markov chain L_f.
4. Instance of the BMPG.
5. Probabilities of the three operators of the neighborhood function.

The levels or possible values for the first four factors are shown in Table 3. For the probabilities of using the perturbation operators we used the sixty-six non-negative integer solutions of the Diophantine equation $a + b + c = 10$. By dividing by 10 the values a, b, and c of every solution of the Diophantine equation we get probabilities for the operators *REX*, *NEX*, and *ROT*:

$$P(REX) = a/10$$
$$P(NEX) = b/10$$
$$P(ROT) = c/10$$

The BMPG instances selected were *mcca*, *fs_183_1*, and *impcol_e*. These instances were selected because of the difficulty to reach their best known bandwidths [26,27,23], so they are good options to identify best combinations of factor values. This way, the possible values for factors in the full factorial design are five for T_0, three for α, three for L_f, three for the BMPG instances, and sixty-six for the probabilities of the three operators. The number of test cases, or combinations of factor values, for the full factorial design is: $5 \times 3 \times 3 \times 3 \times 66 = 8910$. For statistical validity we repeat each test case 31 times [14]. Therefore, the number of times that DRSA was executed for determining the best combination of values for the factors was: $31 \times 8910 = 276,210$.

The selection of the best combinations of factor values takes into account the criteria:

1. Average bandwidth over the 31 runs.
2. Standard deviation of the bandwidth over the 31 runs.
3. Average, over the 31 runs, of the number of times that the neighborhood function was applied to reach the bandwidth.

The main criterion is average bandwidth; to break a tie standard deviation of the bandwidth is used; and to break a second tie average of the number of applications of the neighborhood function is used. For the three selected instances the best

Table 3
Values for the factors T_0, α, L_f, and instance of the BMPG. The values for factor L_f are in terms of the number of vertices n and the number of edges m of the instance.

Factor	Value 1	Value 2	Value 3	Value 4	Value 5
T_0	1	4	40	200	1000
α	0.8	0.9	0.99	–	–
L_f	nm	10 nm	40 nm	–	–
Instance	mcca	fs_183_1	impcol_e	–	–

Table 4
The best seven combinations for the instances *mcca*, *fs_183_1*, and *impcol_e*.

Instance	REX	NEX	ROT	T_0	α	L_f	AVG β	STDV β	AVG # g
mcca	8	0	2	1000	0.80	40 nm	39.19	2.30	1261541.06
mcca	1	4	5	1000	0.99	10 nm	39.38	2.84	2224687.48
mcca	3	6	1	40	0.80	40 nm	39.41	2.45	1760819.77
mcca	6	2	2	4	0.90	40 nm	39.41	2.56	1150486.48
mcca	5	4	1	1	0.99	nm	39.41	2.67	824155.25
mcca	4	0	6	1	0.99	10 nm	39.51	2.78	1529674.74
mcca	3	4	3	1	0.99	10 nm	39.58	2.49	1293136.61
fs_183_1	7	1	2	1000	0.80	10 nm	60.00	0.00	3379060.77
fs_183_1	7	0	3	40	0.99	40 nm	60.00	0.00	4090972.87
fs_183_1	7	3	0	200	0.90	10 nm	60.00	0.00	4258474.35
fs_183_1	7	0	3	1000	0.99	10 nm	60.00	0.00	4271006.64
fs_183_1	7	3	0	40	0.99	nm	60.00	0.00	4330434.48
fs_183_1	5	3	2	200	0.80	40 nm	60.00	0.00	5017583.35
fs_183_1	4	5	1	1000	0.90	10 nm	60.00	0.00	5018727.93
impcol_e	5	4	1	200	0.99	40 nm	43.35	1.55	7970134.90
impcol_e	9	0	1	1000	0.80	10 nm	43.41	1.91	4213421.12
impcol_e	5	4	1	200	0.99	nm	43.45	1.68	6088128.61
impcol_e	8	1	1	1000	0.90	10 nm	43.48	1.54	5678842.90
impcol_e	9	0	1	1000	0.99	10 nm	43.48	1.62	6330422.80
impcol_e	6	3	1	1000	0.99	10 nm	43.48	2.40	6124540.90
impcol_e	8	1	1	1000	0.90	10 nm	43.51	1.60	5850374.25

seven parameter combinations are shown in Table 4. The first column of the table is the name of the instance; columns 2–4 contain the solution (a, b, c) of the Diophantine equation $a + b + c = 10$, where a corresponds to the *REX* operator, b corresponds to the *NEX* operator, and c corresponds to the *ROT* operator. Columns 5–7 are the other three factors of the full factorial design, T_0, α, and L_f. Columns 8 and 9 are average bandwidth and standard deviation of the bandwidth of the 31 runs; and the last column is the average of the number of times that the neighborhood function g was applied.

Table 4 shows that there is no absolute winning combination of parameters, so we choose the winning combination as follows: for the parameters initial temperature (T_0), cooling rate (α), and final length of the Markov chain (L_f) we select the statistical mode value:

- $T_0 = 1000$ (it occurs ten times in the twenty-one results).
- $\alpha = 0.99$ (it occurs eleven times in the twenty-one results).
- $L_f = 10$ nm (it occurs twelve times in the twenty-one results).

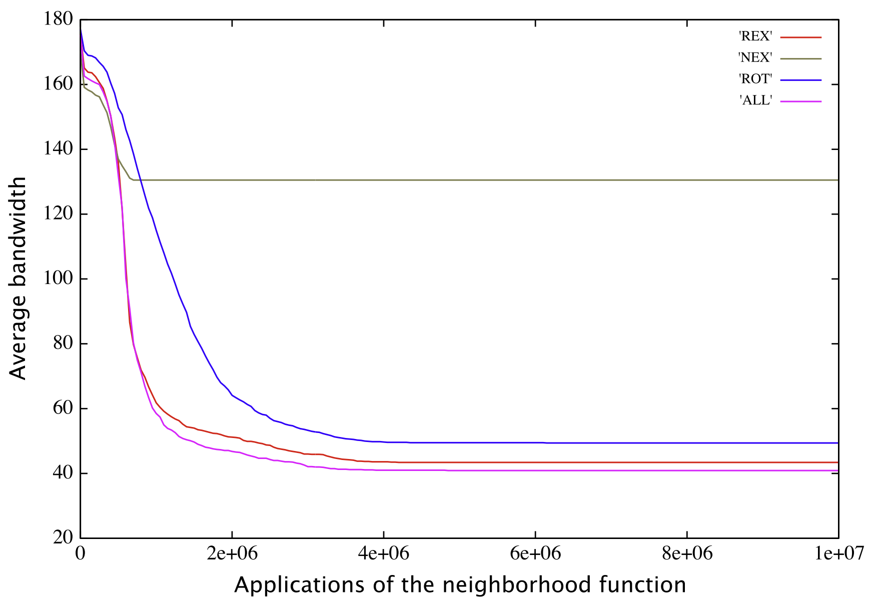


Fig. 4. Evolution of DRSA for the instance *mcca* using only one operator and using the three operators with the probabilities obtained with the tuning process.

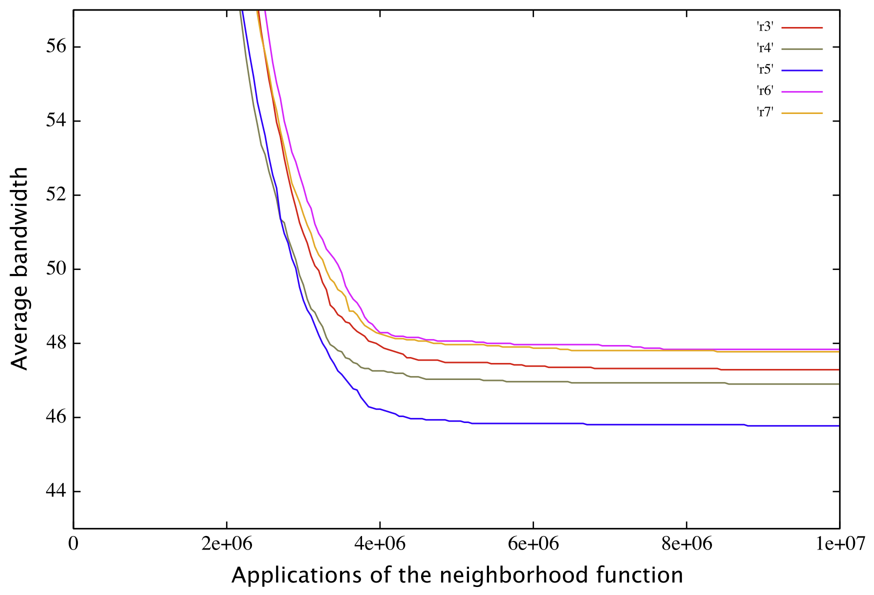


Fig. 5. Performance of DRSA using only the *ROT* operator, but with maximum separation $r = 3, 4, 5, 6, 7$ between the bounds of the *ROT* operator. The best results were obtained with $r = 5$.

We determine the value for the three perturbation operators of the neighborhood function g using the average of the number of times each operator was applied, see Table 4:

$$REX = (8 + 1 + 3 + \cdots + 6 + 8)/21 = 124/21 \approx 6$$
$$NEX = (0 + 4 + 6 + \cdots + 3 + 1)/21 = 48/21 \approx 2$$
$$ROT = (2 + 5 + 1 + \cdots + 1 + 1)/21 = 38/21 \approx 2$$

Then, the probabilities for the three perturbation operators are:

$$P(REX) = 6/10 = 0.6$$
$$P(NEX) = 2/10 = 0.2$$
$$P(ROT) = 2/10 = 0.2$$

This result shows that the neighborhood function requires the three operators (each with its own probability of being applied). Fig. 4 shows the evolution of DRSA with the instance *mcca* for each operator alone, and for the mixture of the three operators using the determined probabilities. The graph shows the results of computing the average bandwidth of 31

Table 5
The nineteen instances with two or more connected components.

Instance	Total number of vertices	Vertices in the largest component
bcsstk20	485	467
bcsstk22	138	110
dwt_234	234	117
gent113	113	104
impcol_a	207	206
jpwh_991	991	983
lns_131	131	123
lns_511	511	503
mbeacxc	496	487
mbeaflw	496	487
mbeause	496	492
mcca	180	168
mcfe	765	731
nos1	237	158
nos2	957	638
pores_3	532	456
saylr3	1000	681
sherman1	1000	681
sherman4	1104	546

Table 6
Results for the 33 small instances from the Harwell-Boeing Sparse Matrix Collection (in bold font improved solutions, in italics font optimal solutions).

Instance	n	LB	Best	GRASP-PR		SA		VNS		DRSA		Δ
				β	t	β	t	β	t	β	t	
arc130	*130*	*63*	*63*	*63*	*0.49*	*63*	*11.83*	*63*	*0.01*	*63*	*18.27*	*0*
ash85	*85*	*9*	*9*	*9*	*0.14*	*9*	*0.56*	*9*	*1.71*	*9*	*0.35*	*0*
bcspwr01	*39*	*5*	*5*	*5*	*0.04*	*5*	*0.20*	*5*	*0.21*	*5*	*0.06*	*0*
bcspwr02	*49*	*7*	*7*	*7*	*0.21*	*7*	*0.10*	*7*	*0.12*	*7*	*0.10*	*0*
bcspwr03	118	9	10	11	0.37	10	0.61	10	0.73	10	0.49	0
bcsstk01	*48*	*16*	*16*	*16*	*0.38*	*16*	*0.31*	*16*	*0.15*	*16*	*0.15*	*0*
bcsstk04	132	36	37	37	1.68	37	40.39	37	0.02	37	2.33	0
bcsstk05	153	19	20	20	2.47	20	5.41	20	0.10	20	2.11	0
bcsstk22	138	9	10	10	0.60	11	2.80	10	19.06	10	1.68	0
can_144	*144*	*13*	*13*	*14*	*1.10*	*13*	*8.98*	*13*	*0.12*	*13*	*4.15*	*0*
can_161	*161*	*18*	*18*	*18*	*0.25*	*18*	*1.53*	*18*	*0.24*	*18*	*3.92*	*0*
curtis54	*54*	*10*	*10*	*10*	*0.24*	*10*	*0.25*	*10*	*0.01*	*10*	*0.13*	*0*
fs_183_1	183	52	60	61	4.08	61	4.44	60	7.27	60	6.71	0
gent113	113	25	27	27	0.32	27	1.99	27	1.09	27	1.89	0
gre_115	115	20	23	24	1.24	23	0.82	23	8.13	23	1.42	0
gre_185	185	17	21	22	2.30	22	3.47	21	0.58	21	4.62	0
ibm32	*32*	*11*	*11*	*11*	*0.11*	*11*	*0.15*	*11*	*0.02*	*11*	*0.20*	*0*
impcol_b	59	19	20	21	0.38	20	0.61	20	0.07	20	0.62	0
impcol_c	137	23	30	31	1.70	30	1.58	30	5.00	30	2.15	0
lns_131	131	18	20	22	0.97	20	0.92	20	1.30	20	1.70	0
lund_a	147	19	23	23	1.82	23	20.71	23	0.01	23	8.06	0
lund_b	147	19	23	23	1.82	23	20.81	23	0.01	23	6.69	0
mcca	180	32	37	37	3.37	37	41.72	37	15.41	37	6.96	0
nos4	*100*	*10*	*10*	*10*	*0.58*	*10*	*0.46*	*10*	*0.45*	*10*	*1.16*	*0*
pores_1	*30*	*7*	*7*	*7*	*0.11*	*7*	*1.58*	*7*	*0.45*	*7*	*0.09*	*0*
steam3	*80*	*7*	*7*	*7*	*0.37*	*7*	*4.54*	*7*	*0.00*	*7*	*0.39*	*0*
west0132	132	23	32	35	3.21	33	2.75	32	21.78	32	1.16	0
west0156	156	33	36	37	3.33	36	1.43	36	6.49	36	0.81	0
west0167	167	31	34	35	2.18	34	2.45	34	35.33	34	1.08	0
will199	**199**	**55**	**64**	**69**	**10.26**	**65**	**3.11**	**65**	**5.75**	**64**	**1.33**	**−1**
will57	*57*	*6*	*6*	*7*	*0.15*	*6*	*0.56*	*6*	*0.64*	*6*	*0.17*	*0*

executions of DRSA with each operator alone and with the mixture of operators. From the plot we can conclude that the use of the mixture of the three operators is more powerful than the use of each operator alone.

Finally, Fig. 5 shows the performance of DRSA with instance *mcca* when *ROT* is the only perturbation operator. The plot shows the results of running DRSA 31 times for each $r \in \{3, 4, 5, 6, 7\}$ and computing the average bandwidth over the 31 runs. This confirms the choice $r = 5$ as the maximum separation between the limits i and j of *ROT*.

5. Computational results

In the previous section we computed values for the parameters of DRSA, and the probabilities of using the three perturbation operators. In this section, we describe the computational experimentation done to test DRSA. DRSA was coded in standard C, compiled with GCC 4.4.6 using the optimization flag -O3, and executed in a processor AMD Opteron™ 6274 at 2.2 GHz.

The benchmark to test the algorithm consists of 113 instances of the Harwell-Boeing sparse matrix collection[22]. This benchmark has been used in important works for bandwidth minimization based on meta-heuristics [4,19,23,26,27]. Some instances of the benchmark have more than one connected component, and in previous works the number of vertices in the largest connected component was considered as the number of vertices of the instance; however, we take into account the entire graph and compute the labeling for all vertices in the graph. Table 5 lists the instances with more than one connected component.

According to the number of vertices in an instance, the benchmark can be divided in two sets: the set of small instances formed by 31 instances with less than 200 vertices, and the set of large instances formed by 82 instances with more than 200 vertices.

Currently, the state of the art algorithm for bandwidth minimization is the variable neighborhood search (VNS) method of Mladenović et al. [23]. And previous to the VNS algorithm, the state of the art method was the improved simulated annealing (SA) algorithm of Rodriguez-Tello et al. [27]. So we compare DRSA against these two methods, and against the GRASP with path relinking method of Piñana et al. [26] which is another important algorithm for solving BMPG.

DRSA was run on an AMD Opteron 6274 at 2.2 GHz, while the other three algorithms were executed on different processors. We ran the VNS algorithm on the Opteron 6274 processor and got an execution time of 35.4 s for the instance *dwt_419*. The time reported in [23] for this instance is 69.21 s, so we can conclude that the Opteron 6274 is $69.21/35.4 = 1.95$ times faster than the Pentium 4 used in [23]. Using this valuable information we scaled the results reported in [23] for the

Table 7
Part I of the results for 80 large instances from the Harwell-Boeing Sparse Matrix Collection (in bold font improved solutions, in italics font optimal solutions).

Instance	n	LB	Best	GRASP-PR		SA		VNS		DRSA		$\varDelta$
				β	t	β	t	β	t	β	t	
494_bus	**494**	**25**	**28**	**35**	**5.46**	**32**	**12.44**	**29**	**10.24**	**28**	**18.27**	**−1**
662_bus	662	36	39	44	13.25	43	35.90	39	36.41	39	13.57	0
685_bus	685	30	32	46	4.91	36	31.62	32	29.34	32	145.02	0
ash292	292	16	19	22	3.53	21	17.54	19	20.07	19	3.53	0
bcspwr04	274	23	24	26	1.90	25	14.28	24	17.10	24	3.22	0
bcspwr05	443	25	27	35	6.57	30	13.67	27	14.56	27	6.56	0
bcsstk06	420	38	45	50	17.26	45	126.43	45	106.54	45	15.15	0
bcsstk19	817	13	14	16	34.04	15	113.68	14	101.66	14	96.76	0
bcsstk20	485	8	13	19	4.26	14	22.75	13	26.59	13	10.04	0
bcsstm07	420	37	45	48	39.23	48	110.01	45	106.86	45	13.65	0
bp_0	**822**	**174**	**234**	**258**	**192.81**	**240**	**71.76**	**236**	**63.04**	**234**	**1218.72**	**−2**
bp_1000	**822**	**197**	**283**	**297**	**321.25**	**291**	**110.26**	**287**	**118.17**	**283**	**4988.25**	**−4**
bp_1200	**822**	**197**	**287**	**303**	**321.21**	**296**	**111.28**	**291**	**91.38**	**287**	**1165.12**	**−4**
bp_1400	**822**	**199**	**291**	**313**	**261.68**	**300**	**110.87**	**293**	**109.28**	**291**	**2140.12**	**−2**
bp_1600	**822**	**199**	**292**	**317**	**275.24**	**299**	**118.22**	**294**	**123.74**	**292**	**2261.37**	**−2**
bp_200	**822**	**186**	**257**	**271**	**210.74**	**267**	**85.88**	**258**	**91.43**	**257**	**1638.61**	**−1**
bp_400	**822**	**188**	**267**	**285**	**207.18**	**276**	**91.80**	**269**	**104.37**	**267**	**2340.62**	**−2**
bp_600	**822**	**190**	**272**	**297**	**203.94**	**279**	**98.58**	**274**	**81.00**	**272**	**1315.45**	**−2**
bp_800	**822**	**197**	**278**	**307**	**231.73**	**286**	**112.15**	**283**	**126.35**	**278**	**1064.59**	**−5**
can_292	292	34	38	42	2.83	41	31.36	38	28.25	38	5.59	0
can_445	445	46	52	58	18.80	56	58.14	52	61.04	52	10.90	0
can_715	**715**	**54**	**71**	**78**	**4.99**	**74**	**113.73**	**72**	**98.27**	**71**	**181.88**	**−1**
can_838	838	75	86	88	12.54	88	195.94	86	205.14	86	70.16	0
dwt_209	209	21	23	24	0.43	23	14.89	23	12.90	23	2.06	0
dwt_221	221	12	13	13	2.71	13	11.93	13	12.18	13	6.54	0
dwt_234	*234*	*11*	*11*	*11*	*0.76*	*11*	*0.56*	*12*	*5.23*	*11*	*1.26*	*0*
dwt_245	*245*	*21*	*21*	*26*	*5.66*	*24*	*4.74*	*21*	*5.20*	*21*	*8.95*	*0*
dwt_310	310	11	12	12	3.30	12	6.63	12	5.84	12	10.48	0
dwt_361	*361*	*14*	*14*	*15*	*0.30*	*14*	*4.23*	*14*	*3.68*	*14*	*23.26*	*0*
dwt_419	419	23	25	29	10.85	26	30.55	25	35.30	25	44.80	0
dwt_503	503	29	41	45	2.71	44	83.38	41	88.94	41	49.60	0
dwt_592	592	22	29	33	43.62	32	63.09	29	56.77	29	52.74	0
dwt_878	878	23	25	35	39.69	26	54.42	25	53.40	25	85.27	0
dwt_918	918	27	32	36	4.61	33	123.93	32	113.83	32	61.93	0
dwt_992	*992*	*35*	*35*	*49*	*105.13*	*35*	*59.36*	*37*	*63.54*	*35*	*107.00*	*0*
fs_541_1	*541*	*270*	*270*	*270*	*8.26*	*270*	*42.02*	*270*	*47.44*	*270*	*0.77*	*0*
fs_680_1	*680*	*17*	*17*	*17*	*14.31*	*17*	*20.30*	*17*	*21.78*	*17*	*27.11*	*0*
fs_760_1	**760**	**36**	**37**	**39**	**38.92**	**38**	**48.76**	**38**	**40.47**	**37**	**38.91**	**−1**
gr_30_30	900	31	33	58	36.64	45	188.75	33	226.00	33	56.98	0
gre_216a	216	17	21	21	3.54	21	1.78	21	1.92	21	4.86	0
gre_343	343	23	28	29	8.11	28	3.77	28	4.13	28	4.13	0

GRASP-PR, SA, and VNS algorithms. In Tables 6–8 the execution times for these algorithms of the 113 instances are presented in scaled form as described below. In what follows we describe Tables 6–8. Each line in the tables corresponds to an instance and data associated with the instance. Columns 1–4 contain the name of the instance, the number of vertices (n), the lower bound (LB), and the best-known bandwidth (Best) for the instance. The next six columns contain data from the three methods against which we compare DRSA (GRASP-PR, SA, and VNS). For each of them we show the best bandwidth (β) reported and the time (t) in seconds required to reach the bandwidth. For DRSA we also show the best bandwidth achieved (β) and the time (t) required to reach that bandwidth. Finally, the last column $\varDelta$ is the difference between the best result of DRSA and the best result among the other three algorithms (when the value is negative it means DRSA improves the previous best known result). We show in bold font the cases where DRSA improves the previous upper bound and in italics the cases that are optimal.

The DRSA final numbers are 31 new upper bounds and the matching of 82 best-known solutions (22 optimal). The best permutations found by DRSA for the 113 instances are available at [28]. With respect to the VNS algorithm (the current state of the art algorithm) DRSA improved 34 solutions and executed faster in 50 instances.

A statistical comparison of best solutions produced by DRSA against best solutions produced by the algorithms GRASP-PR [26], SA [27], and VNS [23], was done using Wilcoxon signed-rank test for two dependent samples. The three tests defined were: GRASP vs DRSA, SA vs DRSA, and VNS vs DRSA. The null hypothesis (H_0) and the alternative hypothesis (H_a) for each test were:

- H_0: There is not a significant difference between solutions produced by DRSA and solutions produced by the GRASP (SA or VNS) algorithm.

Table 8
Part II of the results for 80 large instances from the Harwell-Boeing Sparse Matrix Collection (in bold font improved solutions, in italics font optimal solutions).

Instance	n	LB	Best	GRASP-PR		SA		VNS		DRSA		Δ
				β	t	β	t	β	t	β	t	
gre_512	512	30	36	36	36.33	36	7.50	36	6.91	36	8.87	0
hor_131	**434**	**46**	**54**	**64**	**10.54**	**55**	**78.59**	**55**	**85.37**	**54**	**25.71**	**−1**
impcol_a	207	30	32	34	1.17	32	2.75	32	3.18	32	4.69	0
impcol_d	**425**	**36**	**39**	**42**	**12.30**	**42**	**39.52**	**40**	**47.11**	**39**	**843.50**	**−1**
impcol_e	225	34	42	42	1.28	42	25.30	42	22.27	42	8.47	0
jagmesh1	936	24	27	27	43.32	27	17.24	28	15.31	27	555.47	0
jpwh_991	**991**	**82**	**88**	**96**	**26.10**	**94**	**53.19**	**90**	**58.34**	**88**	**156.68**	**−2**
lns_511	511	33	44	49	22.49	44	63.14	44	64.02	44	30.26	0
mbeacxc	**496**	**246**	**260**	**272**	**719.34**	**262**	**904.74**	**261**	**500.99**	**260**	**313.20**	**−1**
mbeaflw	**496**	**246**	**260**	**272**	**719.91**	**261**	**889.90**	**261**	**491.25**	**260**	**307.09**	**−1**
mbeause	492	249	254	269	814.93	255	657.64	254	454.98	254	530.11	0
mcfe	765	112	126	130	32.36	126	953.14	126	403.99	126	325.44	0
nnc261	261	22	24	25	8.62	25	4.44	24	3.66	24	9.00	0
nnc666	666	33	40	45	21.27	42	55.33	40	54.29	40	552.22	0
nos1	*237*	*3*	*3*	*3*	*1.00*	*3*	*0.56*	*3*	*0.00*	*3*	*0.78*	*0*
nos2	*957*	*3*	*3*	*3*	*6.42*	*3*	*58.39*	*3*	*49.46*	*3*	*63.92*	*0*
nos3	**960**	**43**	**44**	**79**	**72.20**	**62**	**413.92**	**45**	**336.96**	**44**	**326.55**	**−1**
nos5	468	53	63	69	43.23	64	62.17	63	51.61	63	72.03	0
nos6	675	15	16	16	17.01	16	6.37	16	7.46	16	196.48	0
nos7	729	43	65	66	35.22	65	12.19	65	14.35	65	59.93	0
orsirr_2	**886**	**62**	**85**	**91**	**16.36**	**88**	**83.89**	**86**	**84.12**	**85**	**7216.42**	**−1**
plat362	362	29	34	36	1.04	36	96.59	34	91.59	34	10.13	0
plskz362	362	15	18	20	2.81	19	10.66	18	11.37	18	3.89	0
pores_3	*532*	*13*	*13*	*13*	*1.16*	*13*	*6.63*	*13*	*7.16*	*13*	*33.22*	*0*
saylr1	238	12	14	15	3.23	14	1.17	14	1.14	14	1.62	0
saylr3	**1000**	**35**	**46**	**52**	**30.91**	**53**	**7.96**	**48**	**8.04**	**46**	**93.23**	**−2**
sherman1	**1000**	**35**	**46**	**52**	**30.93**	**52**	**7.96**	**47**	**8.72**	**46**	**76.64**	**−1**
sherman4	1104	21	27	27	1.51	27	5.86	27	6.59	27	27.53	0
shl_0	**663**	**211**	**224**	**241**	**35.96**	**229**	**107.86**	**226**	**99.78**	**224**	**17.78**	**−2**
shl_200	663	220	233	247	30.59	235	108.88	233	113.37	233	15.28	0
shl_400	**663**	**213**	**229**	**242**	**38.82**	**235**	**112.91**	**230**	**91.77**	**229**	**20.86**	**−1**
steam1	240	32	44	46	4.32	44	40.34	44	46.11	44	11.02	0
steam2	600	54	63	65	57.71	65	350.57	63	325.98	63	338.34	0
str_0	**363**	**87**	**115**	**124**	**42.38**	**119**	**20.35**	**116**	**22.11**	**115**	**35.14**	**−1**
str_200	**363**	**90**	**124**	**135**	**36.71**	**128**	**24.12**	**125**	**19.52**	**124**	**27.86**	**−1**
str_600	**363**	**101**	**131**	**144**	**29.36**	**132**	**28.51**	**132**	**33.96**	**131**	**25.79**	**−1**
west0381	**381**	**119**	**149**	**159**	**70.27**	**153**	**19.43**	**151**	**17.85**	**149**	**26.23**	**−2**
west0479	**479**	**84**	**119**	**127**	**66.83**	**123**	**20.65**	**121**	**19.63**	**119**	**25.63**	**−2**
west0497	497	69	85	92	36.57	87	70.33	85	57.41	85	11.61	0
west0655	**655**	**109**	**157**	**167**	**96.14**	**161**	**41.05**	**163**	**38.68**	**157**	**471.50**	**−4**
west0989	**989**	**123**	**207**	**217**	**165.84**	**215**	**87.77**	**211**	**78.42**	**207**	**1047.40**	**−4**

- H_a: There is a significant difference between best solutions produced by DRSA and best solutions produced by the GRASP (SA or VNS) algorithm, in the sense that solutions of DRSA are better than solutions of the GRASP (SA or VNS) algorithm.

Table 9 contains the results of the Wilcoxon signed-rank tests; a negative rank appears when the solution produced by DRSA was better than the solution produced by the other algorithm, i.e., DRSA produced a smaller bandwidth; and a positive rank appears when DRSA was unable to reach best result of the other algorithm. According to the result of the 2-tailed p-value, we can accept the alternative hypothesis H_a at a level of significance of $\alpha = 0.01$. We can conclude that there is a statistically significant difference between best solutions produced by DRSA and best solutions produced by the other algorithms, in the sense that DRSA solutions are better.

Table 9
Results of the Wilcoxon signed-rank test.

	GRASP – DRSA	SA – DRSA	VNS – DRSA
Non-zero differences	81	62	34
Negative ranks	81	62	34
Positive ranks	0	0	0
Sum of negative ranks	3321	1953	595
Sum of positive ranks	0	0	0
Value of Z	7.81	6.84	5.08
2-tailed p-value	5.77×10^{-15}	7.91×10^{-12}	3.77×10^{-7}

As we have already mentioned, for comparison against previous works we made use of the benchmark of 113 instances; however, the instances *mbeacxc* and *mbeaflw* are identical in structure, and they are only different in the weight of their edges. Given that for bandwidth minimization the weight of the edges is not relevant these matrices are the same, so the benchmark really consists of 112 different instances. In this work we processed these two instances independently and for both DRSA improved their previous best-known bandwidth in one unit (from 261 to 260). In future works one of these matrices should be removed.

6. Conclusions

We proposed a new simulated annealing algorithm to solve BMPG called DRSA (Dual Representation Simulated Annealing). The name of the algorithm comes from the fact that it uses an internal dual representation of BMPG solutions. For the purpose of working over these representations we defined three perturbation operators called random exchange operator (*REX*), neighbor in graph exchange operator (*NEX*), and rotation operator (*ROT*) that together form the neighborhood function of the algorithm.

The parameters of DRSA were tuned using a full factorial design. The factors in the design were: (a) parameters for the simulated annealing algorithm: initial temperature, cooling rate, and final length of the Markov chain and (b) sixty-six solutions of the Diophantine equation $a + b + c = 10$ to assign probabilities to the three perturbation operators. The results of the tuning process gave an initial temperature of 1000, a cooling rate of 0.99, and a final length of the Markov chain of 10 nm, where n and m are the number of vertices and edges of the graph, respectively. For the three perturbation operators the tuning process gave a utilization rate of 60% for *REX* operator, 20% for *NEX* operator, and 20% for *ROT* operator.

We tested DRSA with the 113 instances of the Harwell-Boeing sparse matrix collection that has been used before for testing other bandwidth minimization algorithms; results of DRSA were 31 new upper bounds and the matching of 82 best-known solutions (22 solutions are optimal). We used Wilcoxon signed-rank test for two dependent samples to compare DRSA with state of the art methods GRASP-PR, SA, and VNS. The result of the test was that a statistically significant difference in favor of DRSA exists over best solutions produced by the others algorithms. The Wilcoxon signed-rank tests indicated that DRSA outperforms the other methods in the quality of solutions.

Encouraged by the results of DRSA with the Harwell-Boeing instances, we are currently planning to apply DRSA to other labeling problems.

Disclaimer

Any mention of commercial products in this paper is for information only; it does not imply recommendation or endorsement by NIST.

Acknowledgments

The authors thank Dragan Urošević (Mathematical Institute, Serbian Academy of Sciences and Arts) for providing us with an executable version of his VNS algorithm for a fair comparison of DRSA with VNS.

We acknowledge the General Coordination of Information and Communications Technologies (CGSTIC) at CINVESTAV for providing high performance computing resources on the hybrid cluster supercomputer "Xiuhcoatl", that have contributed to the research results reported in this paper. Also the authors acknowledge the support of access to the infrastructure of high performance computing of the Information Technology Laboratory at CINVESTAV-Tamaulipas. The second author acknowledges the support for this research to the Juárez Autonomous University of Tabasco. This research was partially funded by the following projects: CONACYT 58554 – Cálculo de Covering Arrays, 51623 – Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas. This paper was completed during the first author's stay as a guest researcher at the U.S. National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899, USA.

References

[1] E.H.L. Aarts, J.H.M. Korst, P.J.M. van Laarhoven, Simulated annealing, in: E. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Princeton University Press, Princeton, NJ, USA, 2003, pp. 91–120.
[2] M.W. Berry, B. Hendrickson, P. Raghavan, Sparse matrix reordering schemes for browsing hypertext, Lect. Appl. Math. 32 (1996) 99–123.
[3] S.N. Bhatt, F.T. Leighton, A framework for solving {VLSI} graph layout problems, J. Comp. Syst. Sci. 28 (2) (1984) 300–343.
[4] V. Campos, E. Piñana, R. Martí, Adaptive memory programming for matrix bandwidth minimization, Ann. Operat. Res. 183 (1) (2006) 7–23.
[5] A. Caprara, J.J. Salazar-González, Laying out sparse graphs with provably minimum bandwidth, INFORMS J. Comput. 17 (3) (2005) 356–373.
[6] E. Cuthill, J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: Proceedings of the 1969 24th National Conference, ACM '69, ACM, New York, NY, USA, 1969.
[7] M. Cygan, M. Pilipczuk, Faster exact bandwidth, Lecture Notes in Artificial Intelligence, vol. 1952, Springer, Berlin/Heidelberg, 2000, pp. 477–486.
[8] M. Cygan, M. Pilipczuk, Faster exact bandwidth, in: H. Broersma, T. Erlebach, T. Friedetzky, D. Paulusma (Eds.), Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, vol. 5344, Springer, Berlin/Heidelberg, 2008, pp. 101–109.
[9] M. Cygan, M. Pilipczuk, Even faster exact bandwidth, ACM Trans. Algor. 8 (1) (2012) 8:1–8:14.
[10] G.M. Del Corso, G. Manzini, Finding exact solutions to the bandwidth minimization problem, Computing 62 (1999) 189–203.
[11] G.W. Dueck, J. Jeffs, A heuristic bandwidth reduction algorithm, J. Combinat. Math. Comp. 18 (1995) 97–108.

[12] A. Esposito, M. Catalano, F. Malucelli, L. Tarricone, Sparse matrix bandwidth reduction: algorithms, applications and real industrial cases in electromagnetics, in: P. Arbenz, M. Paprzycki, A. Sameh, V. Sarin (Eds.), High Performance Algorithms for Structured Matrix Problems, Nova Science Publisher, Inc., 1998, pp. 27–45.
[13] N.E. Gibbs, W.G.P. Jr., P.K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. Numer. Anal. 13 (1976) 236–250.
[14] C.M. Grinstead, L.J. Snell, Grinstead and Snell's Introduction to Probability, version dated 4 july 2006 ed., American Mathematical Society, 2006.
[15] F. Harary, Theory of Graphs and its Applications, Czechoslovak Academy of Science, Prague, 1967.
[16] L.H. Harper, Optimal assignments of numbers to vertices, J. Soc. Indust. Appl. Math. 12 (1) (1964) 131–135.
[17] B. Koohestani, R. Poli, A genetic programming approach to the matrix bandwidth-minimization problem, in: Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part II, PPSN'10, Springer-Verlag, Berlin, Heidelberg, 2010.
[18] A. Lim, J. Lin, B. Rodrigues, F. Xiao, Ant colony optimization with hill climbing for the bandwidth minimization problem, Appl. Soft Comput. 6 (2) (2006) 180–188.
[19] A. Lim, B. Rodrigues, F. Xiao, Heuristics for matrix bandwidth reduction, Euro. J. Operat. Res. 174 (1) (2006) 69–91.
[20] R. Martí, V. Campos, E. Piñana, A branch and bound algorithm for the matrix bandwidth minimization, Euro. J. Operat. Res. 186 (2) (2008) 513–528.
[21] R. Martí, M. Laguna, F. Glover, V. Campos, Reducing the bandwidth of a sparse matrix with tabu search, Euro. J. Operat. Res. 135 (2) (2001) 450–459.
[22] Matrix Market maintained by the Mathematical and Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology (NIST), Matrix Market, 2014 <http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/> (accessed 17.12.14).
[23] N. Mladenović, D. Urošević, D. Pérez-Brito, C.G. García-González, Variable neighbourhood search for bandwidth reduction, Euro. J. Operat. Res. 200 (1) (2010) 14–27.
[24] B. Monien, I.H. Sudborough, Bandwidth constrained np-complete problems, in: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, STOC '81, ACM, New York, NY, USA, 1981.
[25] C. Papadimitriou, The np-completeness of the bandwidth minimization problem, J. Comput. 16 (1976) 263–270.
[26] E. Piñana, I. Plana, V. Campos, R. Martí, Grasp and path relinking for the matrix bandwidth minimization, Euro. J. Operat. Res. 153 (1) (2004) 200–210.
[27] E. Rodriguez-Tello, J.K. Hao, J. Torres-Jimenez, An improved simulated annealing algorithm for bandwidth minimization, Euro. J. Operat. Res. 185 (3) (2008) 1319–1335.
[28] J. Torres-Jimenez, BMPG Permutations Obtained by Dual Representation Simulated Annealing (DRSA) algorithm, 2014 <http://www.tamps.cinvestav.mx/~jtj/bmp/> (accessed 17.12.14).