An Improved Evaluation Function for the Bandwidth Minimization Problem*

Eduardo Rodriguez-Tello[1], Jin-Kao Hao[1], and Jose Torres-Jimenez[2]

[1] LERIA, Université d'Angers.
2 Boulevard Lavoisier, 49045 Angers, France
{ertello, hao}@info.univ-angers.fr
[2] ITESM Campus Cuernavaca, Computer Science Department.
Av. Paseo de la Reforma 182-A. 62589 Temixco Morelos, Mexico
jtj@itesm.mx

Abstract. This paper introduces a new evaluation function, called δ, for the Bandwidth Minimization Problem for Graphs (BMPG). Compared with the classical β evaluation function used, our δ function is much more discriminating and leads to smoother landscapes. The main characteristics of δ are analyzed and its practical usefulness is assessed within a Simulated Annealing algorithm. Experiments show that thanks to the use of the δ function, we are able to improve on some previous best results of a set of well-known benchmarks.

Key words: Bandwidth Evaluation Function, Bandwidth Minimization Problem, Heuristics, Simulated Annealing.

1 Introduction

The Matrix Bandwidth Minimization Problem (MBMP) seems to be originated in the 1950's when structural engineers first analyzed steel frameworks by computer manipulation of their structural matrices [1]. In order that operations like inversion and finding determinants take the least time as possible, many efforts were made to discover an equivalent matrix in which all the nonzero entries would lay within a narrow band near the main diagonal (hence the term "bandwidth") [2]. On the other hand the Bandwidth Minimization Problem for Graphs (BMPG) was proposed independently by Harper [3] and Harary [4]. The MBMP is equivalent to the BMPG, given that a graph can be transformed into an incidence matrix. The BMPG has a large number of applications including for instance circuit design and information retrieval in hypertext.

The BMPG can be defined formally as follows. Let $G = (V, E)$ be a finite undirected graph, where $V = \{1, 2, .., n\}$ defines the set of vertices and $E \subseteq V \times V = \{\{i, j\} \mid i,\ j \in V\}$ is the set of edges. Let $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ be a permutation of V. The bandwidth β of G for τ is defined by:

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

* This work was partially supported by the CONACyT Mexico.

Then the BMPG consists in finding a permutation τ for which $\beta_\tau(G)$ is minimum.

Since there are $n!$ possible permutations for a graph with n vertices, the BMPG is a highly combinatorial problem. Papadimitriou has shown that finding the bandwidth of a graph is NP-Complete [5]. Later, it was demonstrated that the BMPG is NP-Complete even for trees with a maximum degree of three [6].

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. Two examples are proposed in [7]. Both methods solve problems up to 100 vertices, for some classes of matrices. 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. Some examples are: the Cuthill–McKee algorithm [8] and the Gibbs Poole and Stockmeyer algorithm (GPS) [9]. More recently, metaheuristics have been applied to the BMPG: Simulated Annealing (SA) [10], Tabu Search [11] and Genetic Algorithms [12].

The most common practice in all these algorithms is to evaluate the quality of a configuration as the change in the objective function $\beta_\tau(G)$. This provides little or no information during the search process because $\beta_\tau(G)$ only takes into consideration the maximum absolute difference between labels of adjacent vertices in the graph (see Equation 1).

Three exceptions are reported in the literature. In [10], the authors take into account the five maximum absolute differences to evaluate a configuration. In [13], 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 used in graphs with less than 150 vertices. In [11], the value of a move between two vertices u and v is defined as the number of vertices adjacent to v or u whose bandwidth increases due to the move.

Following these ideas, and given that one of the most important elements in heuristic search is how the quality of a configuration is evaluated, a new evaluation function (namely δ) is proposed in this paper. This new evaluation function is able to capture even the smallest improvement that orients the searching of better solutions and permits to find configurations in which all the absolute differences are minimized.

The rest of this paper is organized as follows: Section 2 concentrates on an analysis of the β evaluation function, the cardinality of its equivalence classes and some possible drawbacks of it. In Section 3 a new evaluation function called δ is proposed. It takes into account all the edges of the graph. Section 4 makes a comparison between β and δ. Section 5 shows the implementation details of the SA algorithm used to study the performance obtained when it uses either δ or β. Section 6 explains the computational results obtained with the SA algorithm for both evaluation functions and a comparison with the best heuristics reported in the literature. Finally, in Section 7 some conclusions are presented.

2 The β Evaluation Function

Given that an undirected graph G could potentially have n reflexive edges and $\frac{n(n-1)}{2}$ non-reflexive edges, then the total number of possible edges is given by: $n+(\frac{n(n-1)}{2}) = \frac{n(n+1)}{2}$, and given that a particular edge can be present or absent (please note that we are counting even the graph with no edges), the number of possible graphs is: $2^{\frac{n(n+1)}{2}}$.

β is the most used evaluation function in the BMP algorithms [9, 14, 8, 12]. Next, some features of β are analyzed.

β can only take n different values ($0 \leq \beta \leq n-1$), $\beta = 0$ implies that the graph G has either no edges or only reflexive edges. Consequently the search space of $n!$ possible configurations (permutations of $\{1, 2, ..., n\}$) can be partitioned into n different equivalence classes[1] under β. Additionally, let ω_i be the cardinality of the equivalence class under $\beta = i$. Then, it is easy to show that $\omega_0 = 2^n$, and $\omega_1 = 2^n\left(2^{(n-1)} - 1\right)$ then:

$$\omega_i = (2^{(n-i)} - 1)\prod_{j=1}^{i-1} 2^{(n-j)} \tag{2}$$

Now, in order to verify that all the possible graphs are taken into account, the summation of all cardinalities of the equivalence classes is given in Equation 3, as can be verified the summation equals the total number of graphs.

$$\omega_0 + \sum_{i=1}^{n-1}\omega_i = 2^n + \sum_{i=1}^{n-1}(2^{(n-i)} - 1)\prod_{j=1}^{i-1} 2^{(n-j)} = 2^{\frac{n(n+1)}{2}} \tag{3}$$

It is very important to remark that the β evaluation function does not take into account all the absolute differences between labels of adjacent vertices, but the maximum absolute difference. In this sense there is no way to make distinctions between elements that belong to the same β equivalence class. For example, the two permutations for the graph showed in Fig. 1 belong to the same β equivalence class ($\beta = 3$), which can be a potential drawback when searching a solution, since the permutation in Fig. 1(b) is better than the one in Fig. 1(a) (it is better because it has only one absolute difference with value two).

The β evaluation function is very gross with very few equivalence classes, in consequence, each equivalence class has high cardinality.

3 The δ Evaluation Function

Given the negative features of β it has been developed a new evaluation function, called δ, which takes into account all the edges of the graph. The proposed

[1] Let S be a set with an equivalence relation R. An equivalence class of S under R is a subset $T \subset S$ such that: If $x \in T$ and $y \in S$, then $x\ R\ y$ if and only if $y \in T$. And if $S \neq \emptyset$, then $T \neq \emptyset$.

(a) (b)

Fig. 1. (a) Permutation τ with β=3. (b) Permutation τ' with β=3.

evaluation function for a permutation τ is defined by Equation 4 where d_x refers to the number of absolute differences with value x between adjacent vertices, and β_τ the bandwidth for the permutation τ.

$$\delta(\tau) = \beta_\tau + \sum_{x=0}^{\beta_\tau} \left(\frac{d_x}{\prod_{y=x}^{\beta_\tau} (n + (\beta_\tau - y) + 1)} \right) \tag{4}$$

To illustrate the computation of this new evaluation function, let us consider the graph in Fig. 1(a). For this particular graph: $d_0 = 2$, $d_1 = 1$, $d_2 = 2$, $d_3 = 2$, $d_4 = 0$; additionally it is easy to observe that $\beta = 3$ and $n = 5$.

Then, by making the substitution of these values in the Formula 4 and simplifying we obtain: $\delta(\tau) = 3 + \frac{2}{3024} + \frac{1}{336} + \frac{2}{42} + \frac{2}{6} = 3.3846$.

In contrast if δ is computed for the permutation τ' showed in Fig. 1(b) a different and smaller value is obtained $\delta(\tau') = 3.3638$.

Next the analysis of the equivalence classes of δ is presented. Two permutations belong to the same equivalence class if they have the same set of counters d_x, for instance the set of counters for the permutation in Fig. 2(a) is: $\{d_0 = 2,\ d_1 = 1,\ d_2 = 2\}$ and for the permutation in Fig. 2(b) is: $\{d_0 = 2,\ d_1 = 1,\ d_2 = 2\}$. In this sense both permutations, in Fig. 2, belong to the same equivalence class. Now, given that d_x takes values between 0 and $n - x$, the cardinality of possible values for d_x is $(n - x + 1)$, and the total number of equivalence classes is described by the equation:

$$\prod_{x=0}^{n-1} (n - x + 1) = (n + 1)!$$

The number of graphs that belong to the same equivalence class is:

$$\binom{1}{d_{n-1}} \binom{2}{d_{n-2}} \cdots \binom{n-1}{d_1} \binom{n}{d_0} = \prod_{x=1}^{n} \binom{x}{d_{n-x}} \tag{5}$$

To demonstrate that the summation of the cardinalities of all equivalency classes equals the number of possible graphs, it is necessary to use the Formula

5 instantiated with all possible values for the d_{n-x} counters and compute the sum. This can be expressed as:

$$\prod_{x=1}^{n}\left(\sum_{d_{n-x}}^{x}\binom{x}{d_{n-x}}\right)=\prod_{x=1}^{n}2^{x}=2^{\sum_{x=1}^{n}x}=2^{\frac{n(n+1)}{2}} \tag{6}$$

Then given that the Equation 6 gives the same value as the total number of graphs, we conclude that all the $(n+1)!$ equivalence classes capture all the possible graphs.

(a) (b)

Fig. 2. Two permutations of a graph which belong to the same equivalence class.

4 Comparing β and δ

In this section the differences between the evaluation functions β and δ are contrasted. First we compare the total number of equivalence classes (ω_β and ω_δ) for each evaluation function and then the average values of the cardinalities for these equivalence classes. This is presented in the Table 1 for some graphs with different number of vertices n. The second and third columns show the number of equivalence classes for β and δ respectively, while the fourth and fifth columns show the average of their cardinalities.

It is important to emphasize that ω_β has a linear increment while ω_δ has an exponential one. Thanks to the data presented in Table 1 it is possible to conclude that δ is finer than β since it 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.

5 A Simulated Annealing Approach to Solve the BMPG

To evaluate the practical usefulness of the δ evaluation function, a Simulated Annealing (SA) algorithm was developed. Next some details of the implementation proposed are presented:

Table 1. Comparison between the total number of equivalence classes and the average of their cardinalities.

n	ω_β	ω_δ	$2^{\frac{n(n+1)}{2}}/\omega_\beta$	$2^{\frac{n(n+1)}{2}}/\omega_\delta$
5	5	$7.20E+2$	$6.55E+3$	$4.55E+1$
10	10	$3.99E+7$	$3.60E+15$	$9.03E+8$
70	70	$8.50E+101$	$1.64E+746$	$1.35E+646$
150	150	$8.63E+264$	$9.73E+3406$	$1.70E+3144$
300	300	$9.21E+616$	$1.06E+13589$	$3.47E+12974$

Internal Representation. Let τ be a potential solution of the problem, that is a permutation of V. Then τ is represented as an array of integers of length n, in which the *i-th* element denotes the label assigned to the vertex i of the graph. The solution space is obviously the set of all the permutations of order n, where n is the number of vertices in the graph.

Evaluation Function. The choice of the evaluation function is an 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. All these characteristics are present in the new δ evaluation function whose formal definition is presented in Formula 4.

Neighborhood Function. The neighborhood of a solution $N(\tau)$ in our implementation contains all the permutations τ' obtained by swapping two adjacent vertices of the current permutation τ.

Initial Solution. The initial solution is the starting configuration used for the algorithm to begin searching better configurations using the neighborhood function. In this implementation the initial permutation is generated randomly.

Cooling Schedule. In a SA algorithm the way in which the temperature is decreased is known as the cooling schedule, in our implementation the proportional cooling schedule is used ($T_n = T_{n-1} * 0.92$). The initial temperature was fixed at $1.0E$-03 and the final temperature (T_f) at $1.0E$-09.

Termination Condition. The algorithm stops either if the current temperature reaches T_f, or the number of accepted configurations at each temperature falls below the limit of 25. The maximum number of accepted configurations at each temperature ($maxConfigurations$), depends directly on the number of edges ($|E|$) of the graph, because more moves are required for denser graphs ($maxConfigurations = 15 * |E|$).

All the parameters of the SA algorithm were chosen experimentally, and taking into account some related work reported in [15, 13, 10].

6 Computational Experiments

In this section, we present the experiments accomplished to evaluate the performance of δ over a set of 125 benchmark instances. For these experiments the

above SA algorithm is used. The code, programmed in C, was compiled with *gcc* using the optimization flag -*O*2 and ran into a Pentium 4@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. All the results reported here are data averaged over the 20 corresponding runs.

6.1 Benchmark Instances and Comparison Criteria

Two sets of problem instances were used. The first set has 12 structured instances, randomly generated according to the model proposed by [10]. It consists of six different classes of graphs of increasing sizes including: grids, paths, cycles, binary trees, ternary trees and quaternary trees.

The second set of instances is the same test data used by Martí *et al.* [11] and Piñana *et al.* [16]. It has 113 problem instances from the Harwell-Boeing Sparse Matrix Collection[2], divided into two subsets. The first is composed of 33 instances with 30 to 199 vertices. The second consists of 80 large instances whose sizes vary from 200 to 1000.

The criteria used for evaluating the performance of δ are the same as those used in the literature: the average bandwidth over each instance set and the average CPU time in seconds.

6.2 Comparison Between β and δ

The purpose of the first experiment is to compare the new δ evaluation function and the classical β evaluation function. To do this, we use δ and β within the SA algorithm presented in Section 5 (call these SA algorithms SA-δ and SA-β) and test them on the first set of 12 random instances. Both SA-δ and SA-β were run 20 times on each instance and the results are presented in Table 2. In this table columns 1 to 3 show the name of the graph, 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 metric 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 metric was used.

The results presented in Table 2 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. 3 the bandwidth reduction versus the annealing temperature 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 lower than the bandwidth reached by SA-β.

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

Table 2. Results obtained with two SA for the BMPG using δ and β.

Graph	n	Edges	SA-β	SA-δ	Best SA-β	Best 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%

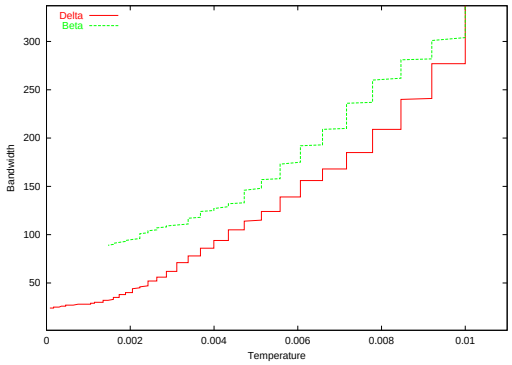

Fig. 3. SA-β and SA-δ behavior comparison on the instance dwt_361.mtx.rnd from the Harwell-Boeing Sparse Matrix Collection.

6.3 Comparison Between SA-δ and the Best Known Results

In this experiment a performance comparison of our SA-δ procedure with the following heuristics was carried out: GPS [9], Dueck and Jeffs' Simulated Annealing (SA-DJ) [10], Tabu Search (TS) [11], GRASP with Path Relinking (GRASP-PR) [16] and Genetic Algorithm with Hill Climbing (GA-HC) [12].

Table 3 shows for each algorithm the average bandwidth over each instance set along with the average CPU time in seconds and the average deviation from the best solutions found by applying all the heuristics to the same instance. It is important to remark that the CPU times for the algorithms GPS, TS, GRASP-PR and GA-HC are taken from [12] where a Pentium 4@1.6 GHz. was used for the experiments. Note that SA-DJ has a cooling schedule that is very slow. Our

SA-δ uses a set of parameters, presented in Section 5, that speeds up the cooling and allows to reach better configurations.

We can observe from Table 3 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 SA-δ is 0.45%, while GRASP-PR (the second best heuristic) obtains a 2.47%. For the large instances it is important to note the remarkable improvement in the average bandwidth obtained with our algorithm (*i.e.,* 94.80 for SA-δ versus 97.05 for GA-HC). In summary, the best solution quality, for the both experiments, is obtained by SA-δ. It was able to match 101 out of the 113 best known solutions, which outperforms the GA-HC (the best heuristic here) that was only able to find 48 of the best solutions.

Table 3. Performance comparison according to problem size.

	33 instances with $n = 30, ..., 199$					
	GPS	SA-DJ	TS	GRASP-PR	GA-HC	SA-δ
Average β	31.42	29.36	23.33	22.52	22.67	22.03
Deviation	35.49%	56.50%	9.63%	2.47%	5.66%	0.45%
CPU sec.	0.003	1434.97	2.36	4.21	2.54	11.18
	80 instances with $n = 200, ..., 1000$					
	GPS	SA-DJ	TS	GRASP-PR	GA-HC	SA-δ
Average β	156.38	164.59	100.78	99.43	97.05	94.80
Deviation	46.96%	222.32%	11.77%	6.59%	6.22%	1.14%
CPU sec.	0.11	1800.00	121.66	323.19	85.22	199.25

7 Conclusions

In this paper, we have introduced the δ evaluation function for the BMPG. It has two important features: a) It considers all the edges of the graph and b) It produces more equivalence classes with lower cardinality. δ 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.

To validate the practical usefulness of δ, two versions of a basic SA (SA-δ and SA-β) was implemented. They were compared using a set of randomly generated structured graphs and the results showed that for many classes of graphs an average improvement of 42% can be achieved when δ is used.

On the other hand, the goodness of the SA-δ algorithm was also validated using a set of benchmarks from the Harwell-Boeing Sparse Matrix Collection. Our approach was able to match 101 out of the 113 best known solutions, and outperforms thus other state-of-the-art heuristics.

Finally, let us notice that the δ evaluation function proposed in this paper can be used by other metaheuristic algorithms (Genetic Algorithms, Tabu Search, Scatter Search) to boost their performance.

More generally, we think that the research of new evaluation functions for combinatorial problems is a very important topic, because it permits to improve the search power of metaheuristics.

Acknowledgments. The authors would like to thank Rafael Martí who have kindly provided us with his test data and detailed results.

References

1. Livesley, R.: The analysis of large structural systems. Computer Journal **3** (1960) 34–39
2. Chinn, P., Chvatalova, J., Dewdney, A., Gibbs, N.: The bandwidth problem for graphs and matrices - a survey. Journal of Graph Theory **6** (1982) 223–254
3. Harper, L.: Optimal assignment of numbers to vertices. Journal of SIAM **12** (1964) 131–135
4. Harary, F.: Theory of Graphs and its Aplications. Czechoslovak Academy of Science, Prague (1967) M. Fiedler.
5. Papadimitriou, C.: The NP-Completeness of the bandwidth minimization problem. Journal on Computing **16** (1976) 263–270
6. Garey, M., Graham, R., Johnson, D., Knuth, D.: Complexity results for bandwidth minimization. SIAM Journal of Applied Mathematics **34** (1978) 477–495
7. Corso, G.D., Manzini, G.: Finding exact solutions to the bandwidth minimization problem. Computing **62** (1999) 189–203
8. Cutchill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the ACM National Conference, New York (1969) 157–172
9. Gibbs, N., Poole, W., Stockmeyer, P.: An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM Journal on Numerical Analysis **13** (1976) 235–251
10. Dueck, G., Jeffs, J.: A heuristic bandwidth reduction algorithm. Journal of Combinatorial Mathematics and Computers **18** (1995) 97–108
11. Martí, R., Laguna, M., Glover, F., Campos, V.: Reducing the bandwidth of a sparse matrix with tabu search. European Journal of Operational Research **135** (2001) 211–220
12. Lim, A., Rodrigues, B., Xiao, F.: Integrated genetic algorithm with hill climbing for bandwidth minimization problem. Lecture Notes in Computer Science **2724** (2003) 1594–1595
13. Torres-Jimenez, J., Rodriguez-Tello, E.: A new measure for the bandwidth minimization problem. Lecture Notes in Computer Science **1952** (2000) 477–486
14. Esposito, A., Catallano, M.F., Malucelli, F., Tarricone, L.: A new matrix bandwidth reduction algorithm. Operations Research Letters **23** (1999) 99–107
15. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220** (1983) 671–680
16. Piñana, E., Plana, I., Campos, V., Martí, R.: GRASP and path relinking for the matrix bandwidth minimization. European Journal of Operational Research **153** (2004) 200–210