Scatter Search for the Cutwidth Minimization Problem

JUAN J. PANTRIGO

Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
juanjose.pantrigo@urjc.es

RAFAEL MARTÍ

Departamento de Estadística e Investigación Operativa
Universidad de Valencia, Spain
rafael.marti@uv.es

ABRAHAM DUARTE

Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
abraham.duarte@urjc.es

EDUARDO G. PARDO

Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
eduardo.pardo@urjc.es

July 28, 2010

Abstract — The cutwidth minimization problem consists of finding a linear layout of a graph so that the maximum linear cut of edges is minimized. This NP-hard problem has applications in network scheduling, automatic graph drawing and information retrieval. We propose a heuristic method based on the Scatter Search (SS) methodology for finding approximate solutions to this optimization problem. This metaheuristic explores solution space by evolving a set of reference points. Our SS method is based on a GRASP constructive algorithm, a local search strategy based on insertion moves and voting-based combination methods. We also introduce a new measure to control the diversity in the search process. Empirical results with 252 previously reported instances indicate that the proposed procedure compares favorably to previous metaheuristics for this problem, such as Simulated Annealing and Evolutionary Path Relinking.

KeyWords: Cutwidth, Metaheuristics, Scatter Search.

1. Introduction

The cutwidth minimization problem consists of finding a linear layout of a graph so that the maximum linear cut of edges (i.e., the number of edges that cut a line between consecutive vertices) is minimized. An interesting application of this NP-hard problem (Gavril 1977) appears in telecommunication to migrate an old network to a new one. Andrade and Resende (2007a) described in detail this application called *Network Migration Scheduling Problem*, in which all traffic originated or terminating at a given node in the old network is moved to a specific node in a new network. Other applications can be traced back 35 years (Adolphson and Hu 1973) to establish the number of channels in an optimal layout of a circuit (see also Makedon and Sudborough, 1989). Recent developments include network reliability (Karger 1999) and information retrieval (Botafogo, 1993).

The cutwidth minimization problem can be easily described in mathematical terms. Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, a labeling or linear arrangement f of G assigns the integers $\{1,2, \dots, n\}$ to the vertices in V, where each vertex receives a different label. The cutwidth of a vertex v with respect to f, $CW_f(v)$, is the number of edges $(u, w) \in E$ satisfying $f(u) \leq f(v) < f(w)$. The cutwidth of the graph, $CW_f(G)$, is the maximum of the cutwidth of its vertices:

$$CW_f(G) = max_{v \in V}\ CW_f(v)$$

The cutwidth minimization problem (CMP) consists of finding a labeling f that minimizes $CW_f(G)$. This optimization problem is NP-hard in even for graphs with a maximum degree of three (Makedon et al., 1985).

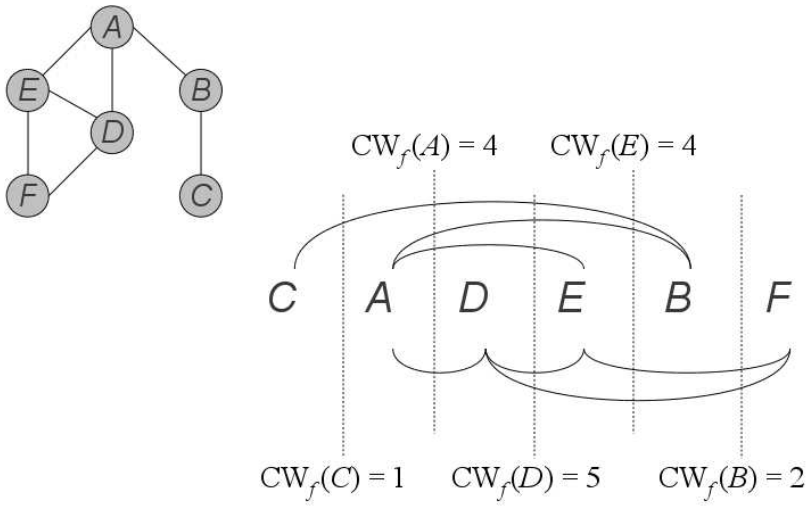

Figure 1: (a) Graph example, (b) Cutwidth of G for a labeling *f*.

Figure 1.a shows an example of an undirected graph with 6 vertices and 7 edges. Figure 1.b shows a labeling, f, of the graph in Figure 1.a, setting the vertices in a line with the order of the labeling, as commonly represented in the CMP. In this way, since $f(C) = 1$, vertex *C* comes first, followed by vertex *A* ($f(A) = 2$) and so on. We represent *f* with the ordering (*C, A, D, E, B, F*) meaning that vertex *C* is located in the first position (label 1), vertex *A* is located in the

second position (label 2) and so on. In Figure 1.b, the cutwidth of each vertex is represented as a dashed line with its corresponding value. For example, the cutwidth of vertex *C* is $CW_f(C) = 1$, because the edge (*C*,*B*) has and endpoint in *C* labeled with 1 and the other endpoint in a vertex labeled with a value larger than 1. In a similar way, we can compute the cutwidth of vertex *A*, $CW_f(A) = 4$, by counting the appropriate number of edges ((*C*,*B*), (*A*,*B*), (*A*,*E*), and (*A*,*D*)). Then, since the cutwidth of the graph *G*, $CW_f(G)$, is the maximum of the cutwidth of all vertices in V, in this particular example we obtain $CW_f(G) = CW_f(D) = 5$.

In spite of the practical applicability of the CMP, which is well-documented in a patent recently filed by Resende and Andrade (2009), researchers on heuristic optimization have paid little attention to it. We have only found three references concerning heuristic methods for this problem. Cohoon and Sahni (1987) tackled a generalization of the cutwidth called *Board Permutation Problem*. They proposed different constructive methods, local search procedures, and two metaheuristics based on Simulated Annealing. Extensive experimentation permitted them to conclude that the best procedure combines two constructive methods with Simulated Annealing.

Andrade and Resende (2007a) proposed a GRASP with Path Relinking for the CMP. The constructive procedure starts with an initial ordering of the vertices based on a randomized depth-first search. Then, according to this ordering, vertices are placed in the position where the increase in the objective function is minimized. The local search procedure is based on swaps, searching for the first improvement in the objective function. The local search stops when all swaps have been performed without improvement. The Path Relinking post-processing (PR) evaluates, at each step, all possible moves from the current solution to the guiding solution generating a set of intermediate solutions. The procedure stops when the current solution reaches the guiding solution, returning the best solution found in the path. The same authors presented an extension of this method (Andrade and Resende, 2007b) in which the main contribution is the introduction of a new methodology called Evolutionary Path Relinking, where a set of elite solutions populated with GRASP evolves by applying PR to its solutions.

In this paper we propose a Scatter Search procedure for the cutwidth minimization problem. Section 2 is devoted to describe the outline of our Scatter Search method, starting with an overview of the method. Then we describe in the following sections the main elements of the algorithm. Section 3 is devoted to the constructive method, Section 4 to the improvement methods and finally, Section 5 to the combination methods. Computational experiments are described in Section 6 and associated concluding remarks are made in Section 7.

2. Scatter Search

Scatter Search (SS) is a metaheuristic that explores solution spaces by evolving a set of reference points (Laguna and Martí, 2003). To turn this principle approach into an effective algorithm five methods and their associated strategies have to be defined. The search starts by applying the **diversification generation method**, obtaining as a result a population of points from which a subset is selected as the initial reference set (*RefSet*). The *RefSet* is evolved

through the application of four additional methods: **subset generation, combination, improvement** and **update**. Figure 2, taken from Laguna and Martí (2003), depicts the basic scheme of SS, showing how the five methods interact.

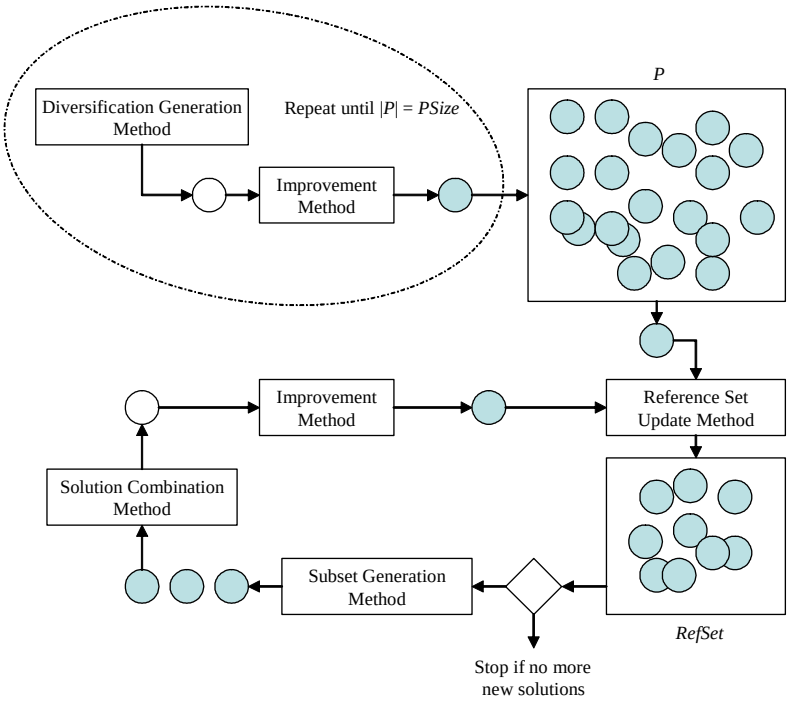

Figure 2. Illustration of the SS design.

In Figure 2, the diversification generation and improvement methods are initially applied until the cardinality of *P* reaches *PSize* solutions that are different from each other. The darker circles represent improved solutions resulting from the application of the improvement method. The main search loop appears to the left of the box containing the reference solutions and labeled *RefSet*. The subset generation method takes reference solutions as input to produce solution subsets to be combined. Solution subsets contain two or more solutions. The new trial solutions resulting from the application of the combination method are subjected to the improvement method and handed to the reference set update method. This method applies rules regarding the admission to the reference set of solutions coming from *P* or from the application of the combination and improvement methods.

We refer the reader to Laguna and Martí (2003) for a detailed description of the method. We follow the standard SS design in our implementation for the cutwidth minimization problem with the exception of the reference set update method. Solutions in *RefSet* are ordered according to quality, where the best solution x^1 is the first one in the list and x^b is the worst one (both in terms of the value of objective function). Given a new solution x generated by combination, let y^x be the closest solution to x in *RefSet*. The solution x is admitted to *RefSet* if it improves upon the best solution in it, x^1, or alternatively, if it improves upon the worst solution, x^b, and its distance with y^x is larger than a pre-established threshold $dthresh$. If solution x qualifies to enter *RefSet* (i.e., if it satisfies this criteria), then it replaces y^x.

As shown above, the definition of distance between solutions is a key design issue in Scatter Search. In our implementation for the CMP, given two solutions $f1$ and $f2$, their distance, $d(f1, f2)$, is computed as follows:

$$d(f1, f2) = min\left\{\sum_{v \in V} |f1(v) - f2(v)|, \sum_{v \in V} |f1(v) - f_R2(v)|\right\}$$

where f_R2 is the reverse permutation of $f2$ computed as $f_R2(v) = n - f2(v) + 1$. As it is customary in SS, the distance from a solution to a set of solutions is computed as the minimum distance to any solution in the set.

3. Diversification Generation Methods

In this section we propose four different solution generation methods based on the GRASP methodology (Feo and Resende, 1989; 1995 and Resende and Ribeiro, 2003). Figure 3 shows the pseudocode of the constructive procedure C1. It starts by creating a list of unlabeled vertices U (initially $U = \mathrm{V}$). The constructive algorithm selects the vertex u with minimum degree in G (step 3) and labels it with 1 (step 4). In case of ties (several vertices with minimum degree), one of them is randomly selected. In subsequent construction steps, the rest of labels are assigned in lexicographical order to the rest of vertices in U. The key point of the method is therefore the order in which the vertices are selected for label assignment.

PROCEDURE C1

1. Let S and U be the sets of labeled and unlabeled vertices of the graph respectively
2. Initially $S = \emptyset$ and $U = V$
3. $u = arg\ min_{v \in V}\{|N(v)|\}$
4. Assign the label $k = 1$ to u $(f(u) = 1)$
5. $S = \{u\}, U = U \setminus \{u\}$

WHILE $(U \neq \emptyset)$

6. $k = k + 1$
7. Construct $CL = \{u \in U\ /\ (w, u) \in E\ \forall\ w \in S\}$
8. Calculate $CW_f(u)\ \forall\ u \in CL$ considering that u is labeled with k (i.e. $f(u) = k$)
9. Compute $CW_{min} = min_{u \in CL}\ CW_f(u)$ and $CW_{max} = max_{u \in CL}\ CW_f(u)$
10. Construct $RCL = \{u \in CL\ /\ CW_f(u) \leq CW_{min} + \alpha(CW_{max} - CW_{min})\}$
11. Select a vertex u^* randomly in *RCL*
12. $f(u^*) = k$
13. $U = U \setminus \{u^*\},\ S = S \cup \{u^*\}$

Figure 3. Pseudocode of the constructive method C1.

The candidate list *CL* is formed with all the vertices in U that are adjacent to one or more labeled vertices. For each vertex u in *CL* the evaluation of u is directly computed as its cutwidth $CW_f(u)$ considering that u is labeled with the next label k; i.e. $f(u) = k$. In mathematical terms:

$$CW_f(u) = CW_f(v) - |N_{labeled}(u)| + |N_{unlabeled}(u)|$$

where $CW_f(v)$ is the cutwidth of the last labeled vertex v in f, $N_{labeled}(u)$ is the set of labeled adjacent vertices to u, and $N_{unlabeled}(u)$ is the set of unlabeled adjacent vertices to u. Note that a greedy selection would label in this step the vertex u with minimum $CW_f(u)$ value. Instead of that, C1 computes in step 10 a restricted candidate list, *RCL*, with good candidates (according to a search parameter α). As it is customary in GRASP, it is based on a threshold on the objective function value, $CW_{min} + \alpha(CW_{max} - CW_{min})$. Then C1 selects in step 11 one element at random in *RCL* and labels it in step 12. The method continues as long as the set of unlabeled elements *U* is not empty. The value of α is randomly selected in [0,1] each time C1 is call.

We also consider an alternative construction procedure introduced in Resende and Werneck (2004). In this procedure, called C2, we first choose candidates randomly and then evaluate them according to a greedy function to make the greedy choice. Figure 4 shows a pseudocode of C2. In each iteration, the algorithm randomly selects *size* elements from the candidate list CL (step 6), where the value of *size* is computed as a percentage α of the number of elements in CL. Then, the element, u^*, with the smallest CW_f-value is selected (step 11) to become part of the partial solution under construction. The evaluation of the elements *u* in CL, $CW_f(u)$, is computed in the same way as in C1. Similarly, the value of α is randomly selected in [0,1] each time C2 is call.

PROCEDURE C2
1. Let S and U be the sets of labeled and unlabeled vertices of the graph respectively
2. Initially $S = \emptyset$ and $U = V$
3. $u = arg\ min_{v \in V}\{|N(v)|\}$
4. Assign the label $k = 1$ to u. $S = \{u\}$, $U = U \setminus \{u\}$

WHILE ($U \neq \emptyset$)
5. $k = k + 1$
6. Construct $CL = \{u \in U\ /\ (w,u) \in E\ \forall\ w \in S\}$
7. $Size = \alpha_2 |CL|$
8. Construct *RCL* by randomly selecting *size* vertices from *CL*
9. Compute $CW_f(u)\ \forall\ u \in RCL$
10. Select a vertex $u^* \in RCL\ /\ u^* = \text{argmin}_{u\ \in\ RCL}\{CW_f(u)\}$
11. Label u^* with the label k
12. $U = U \setminus \{u^*\}, S = S \cup \{u^*\}$

Figure 4. Pseudocode of the constructive method C2.

The third constructive procedure, C3, is similar to C1, but replacing the evaluation $CW_f(u)$ based on the objective function with $|N_{labeled}(u)|$, the number of adjacent labeled vertices. This strategy is based on the fact that if $f(u)$ takes a value close to $f(v)$ for a given edge (u,v), then we are reducing the number of vertices *w* such that $f(u) < f(w) < f(v)$. In other words, C3 tries to reduce the number of vertices affected by the edge (u,v) labeling adjacent vertices with close labels. Finally, the constructive procedure C4 is a variant of C3 in which the random and greedy selections are alternated (as we did with C1 and C2). C4 randomly chooses candidates and then it evaluates each candidate according to the same greedy function in C3, selecting finally the best evaluated candidate.

4. Improvement Method

We propose a local search strategy, LS_Insert, for the CMP based on insertion moves. Specifically, given a labeling f, we define $Move(f, j, v_i)$ consisting of deleting v_i from its current position $i = f(v_i)$ and inserting it in position j. This operation results in the ordering f', as follows:

- If $i > j$, then $f = (\dots, v_{j-1}, v_j, v_{j+1}, \dots, v_{i-1}, v_i, v_{i+1}, \dots)$ and the vertex v_i is inserted just before the vertex v_j , obtaining $f' = (\dots, v_{j-1}, v_i, v_j, v_{j+1}, \dots, v_{i-1}, v_{i+1}, \dots)$.
- If $i < j$ then $f = (\dots, v_{i-1}, v_i, v_{i+1}, \dots, v_{j-1}, v_j, v_{j+1}, \dots)$ and the vertex v_i is inserted just after the vertex v_j , obtaining $f' = (\dots, v_{i-1}, v_{i+1}, \dots, v_{j-1}, v_j, v_i, v_{j+1}, \dots)$.

The objective of the CMP consists in minimizing a maximum value. Consequently, there may be many different solutions with the same objective function value. In other words, the solution space presents a "flat landscape", which usually is a problem for local search based methods, where the search is based on movements and most of them have a null value. Considering that for a given labeling f there may be multiple vertices with cutwidth value equal to $CW_f(G)$, changing a labeling to decrease the cutwidth $CW_f(u)$ of a particular vertex u does not necessarily imply that $CW_f(G)$ also decreases. Additionally, vertices with cutwidth values close to $CW_f(G)$ do not determine the value of the objective function in the current labeling, but they are considered likely to do so in subsequent iterations. Therefore, following the candidate list strategy introduced in Piñana et al. (2004), we define the set of critical vertices CV as those with a cutwidth value equal or close to the cutwidth of the graph. In mathematical terms, the set of critical vertices is defined as:

$$CV = \bigcup_{i \geq \lceil \beta CW_f(G) \rceil} S_i$$

where $S_i = \{v \in V \,/\, CW_f(v) = i\}$ is the set of vertices with a cutwidth value equal to i and the threshold value is computed as a percentage β ($0 \leq \beta \leq 1$) of the current objective function value $CW_f(G)$, $\lceil \beta CW_f(G) \rceil$. In our computational experimentation we study the impact of β on the search procedure.

The local search method first constructs the set of critical vertices CV. Then, the S_i sets are scanned in descending order of i, starting with $i = CW_f(G)$, and the first improvement move is performed. Specifically, in each iteration, it selects a vertex v in S_i to evaluate $Move(f, pos, v)$ where the position pos in which v is inserted is computed as the median of the positions of its adjacent vertices. If this results in an improving move, it is performed. Otherwise, we consider $Move(f, j, v)$ with $j \in [pos - w,\ pos + w\ ]$, where w is a search parameter, and perform the first improving move. To overcome the lack of information provided by the move value in terms of the objective function, we extend here the meaning of "improving". Given a vertex v, we consider that a move improves the current solution if it is able to reduce the cardinality of any set $S_i \subseteq CV$ with $i \geq CW_f(v)$.

In order to improve the efficiency of the procedure, CV is not updated after performing a single move. As suggested in Glover and Laguna (1997), we do not update the candidate list

CV every iteration but only when all the vertices have been scanned. This strategy is particularly useful when move value updates are computationally expensive, as in our context.

In a straightforward implementation, the complexity of computing the cutwidth of each vertex $CW_f(v)$ in the graph is $\Theta(m)$ because the method should scan all the edges in the graph. Additionally, the computation of $CW_f(G)$ is $\Theta(n)$ since it requires to compute the maximum of $CW_f(v)$ for each vertex. However, it is clear that the cutwidth of some vertices does not change when we perform a move and therefore we can save their computation. The following example illustrates it on the graph depicted in Figure 1.a. Let $f = (C, A, D, E, B, F)$ be a solution of the cutwidth problem. Suppose that we perform $Move(f, 2, B)$, obtaining solution $f' = (C, B, A, D, E, F)$. Figure 5 graphically shows the solutions before and after the move.

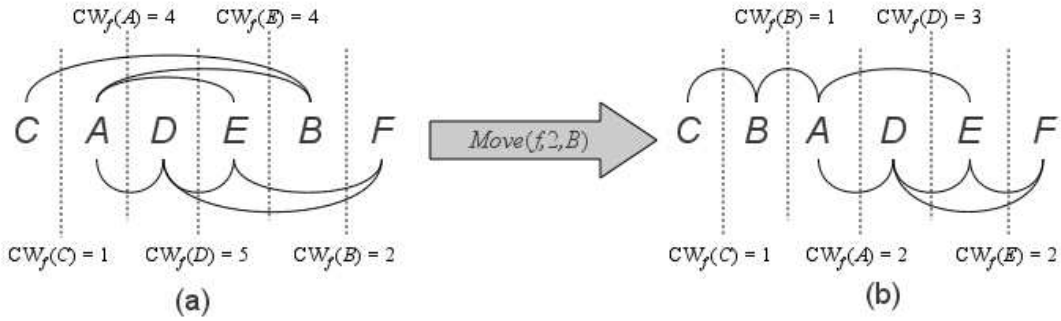

Figure 5. Insertion move example.

Figure 5 shows that vertices C and F are not affected by $Move(f, 2, B)$ and we do not need to re-compute their cutwidth.

When we perform $Move(f, j, v)$, where for example $j < f(v)$, it is clear that for any vertex w such that $f(w) < j$ or $f(w) > f(v)$, its cutwidth does not change. On the contrary, those vertices placed in positions between j and $f(v)$ change their cutwidth value. Let $N_i^L(w)$ be the set of vertices adjacent to w placed in any previous position to i and let $N_i^R(w)$ be the set of vertices adjacent to w placed in any subsequent position to i. In mathematical terms:

$$N_i^L(w) = \{u \in V, (u, w) \in E \,/\, f(u) < i\}$$

$$N_i^R(w) = \{u \in V, (u, w) \in E \,/\, f(u) > i\}$$

For example, in Figure 5.a the set $N_4^L(D) = \{A, E\}$ because both vertices $\{A, E\}$ are adjacent to D and both are placed in a previous position to 4. Analogously $N_4^R(D) = \{F\}$ since F is the only adjacent to D placed in a subsequent position to 4.

When we perform $Move(f, j, v)$ the resulting cutwidth value of a vertex w with $f(w) = i$ and $j < i < f(v)$ can be computed as:

$$CW_{f'}(w) - CW_f(w) = \begin{cases} \left|N_i^L(v)\right| - \left|N_i^R(v)\right|, & \text{if} \quad j < f(w) < f(v) \\ \left|N_i^R(v)\right| - \left|N_i^L(v)\right|, & \text{if} \quad j > f(w) > f(v) \end{cases}$$

We therefore compute the cutwidth of the vertices after a move in an incremental way. Moreover, to compute the value of the objective function, we store in a set all the vertices w

satisfying $CW_f(w) = CW_f(G)$. After a move, if this set is not empty the objective function does not change. Otherwise, we need to update $CW_f(G)$ scanning the vertices in the set of critical vertices CV, selecting the maximum cutwidth among them. Then, the update of $CW_f(G)$ is $O(n)$ instead of $\Theta(n)$ as in a straightforward implementation.

5. Combination Methods

We introduce now three different combination methods for the CMP. They are applied on the subsets created with the *subset generation method* (see Figure 2) to combine the two solutions in each subset to create a new trial solution. These methods are based on the "voting strategy" described in Laguna and Martí (2003) where each solution votes for its first component not included in the combined solution. Each method considers a different selection criterion to determine the element to be assigned to the next "free position" in the combined solution. Specifically, the combination method CM1 selects in each step the vertex with the lowest label between those voted for each solution. Figure 6 shows the first three steps of CM1, where solutions f_1 and f_2 are combined in order to produce f_{12}.

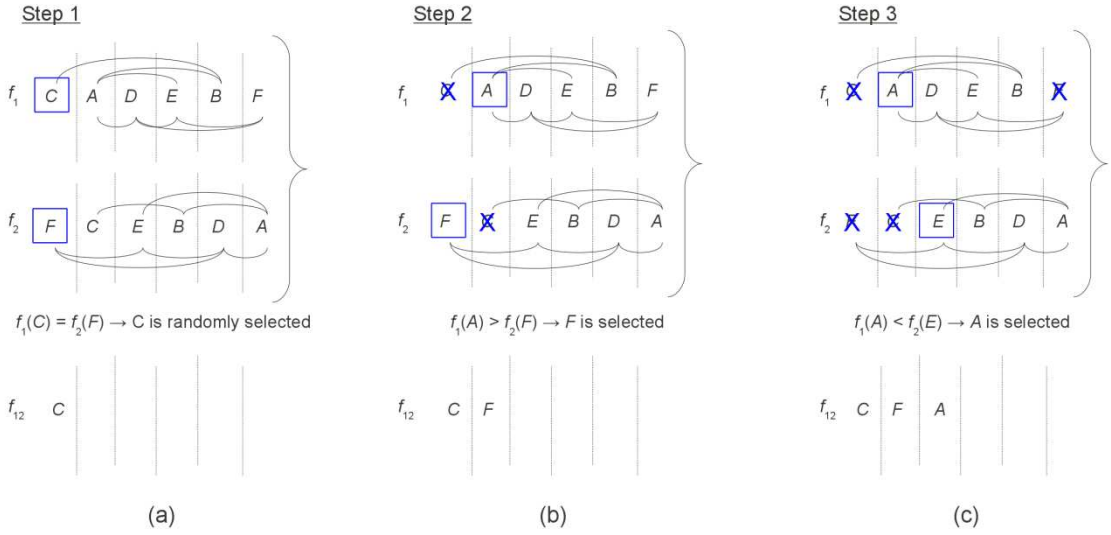

Figure 6. Combination method CM1.

Figure 6.a represents the first step of the algorithm. In this case, each solution votes for including its first vertex in the ordering. Considering that the selection criterion is given by the position in the ordering, we are facing a tie in this situation. The tie-breaking rule is implemented as a random selection. Let us consider that vertex *C* is selected (the bottom-right on the figure illustrates this selection). Then, this vertex is no longer available in subsequent iterations. In the second step (see Figure 6.b), solution f_1 votes for vertex *A* and solution f_2 votes for vertex *F*. Vertex *F* is selected to be included in the combined solution because it has a lower label than *A* (and therefore *F* is no longer available for selection). Figure 6.c shows the third step, where solution f_1 votes for vertex *A* (with label 2) and solution f_2 votes for vertex *E* (with label 3), being selected vertex *A*. This procedure ends when all vertices have been scanned.

The combination method CM2 follows the same logic, but the selection criterion is based on the cutwidth of each vertex, solving ties at random as CM1. Figures 7 shows the first three steps of the combination of f_1 and f_2 using CM2 to produce f_{12}.

Figure 7.a depicts the first step of CM2 where each solution votes for including their first vertex in the ordering. In this example, f_1 votes for vertex *C* with $CW_{f1}(C) = 1$ and f_2 votes for vertex *F* with $CW_{f2}(F) = 2$. Consequently, vertex *C* is selected and no longer considered in the combination process. Figure 7.b shows step 2, where vertex *F* is selected since $CW_{f1}(A) = 4$ and $CW_{f2}(F) = 2$. In the third step (Figure 7.c), a tie is produced, selecting vertex *E* at random.

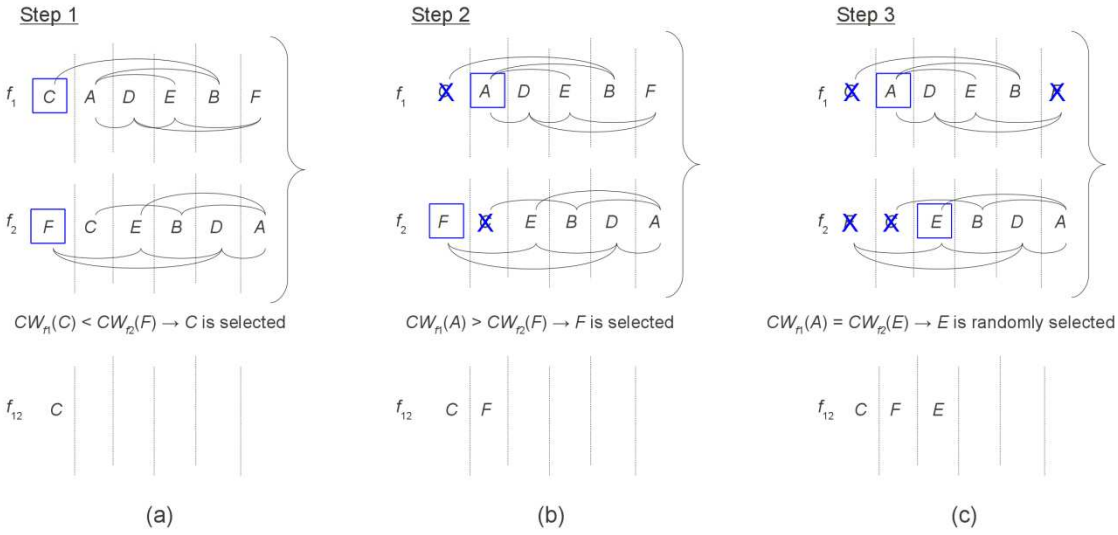

Figure 7. Combination method CM2.

The combination method CM3 considers as the selection criterion the value of the objective function of the combined solution. As in previous combination methods, f_1 and f_2 vote for its first vertex not included in the combined solution f_{12}. Let u_1 and u_2 be the voted vertices of f_1 and f_2, respectively. CM3 computes $CW_{f12}(u_1)$ and $CW_{f12}(u_2)$, selecting the one with the minimum value. As pointed above, ties are broken at random.

6. Computational Experiments

This section describes the computational experiments that we performed to test the efficiency of our Scatter Search procedure as well as to compare it to state-of-the-art methods for solving the CMP. We have implemented the Scatter Search procedure in Java SE 6 and all the experiments were conducted on an Intel Core 2 Quad CPU and 6 GB RAM.

We have employed three sets of instances in our experimentation, totalizing 252 instances. The first one, Small, was introduced in Martí et al. (2008), the second one, Grids, was introduced in Rolim et al. (1995) and the third one, Harwell-Boeing, is a subset of the public-domain Matrix Market library (available at http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/). All these instances are available at http://heur.uv.es/optsicom/cutwidth.

Small: This data set consists of 84 graphs introduced in the context of the bandwidth reduction problem. The number of vertices ranges from 16 to 24, and the number of edges ranges from 18 to 49.

Grids: This data set consists of 81 matrices constructed as the Cartesian product of two paths (Raspaud et al., 2009). They are also called two dimensional meshes and, as documented in Raspaud et al. (2009), the optimal solution of the cutwidth problem for these types of instances is known by construction. For this set of instances, the vertices are arranged on a grid with a dimension *width* × *height* where *width, height* are selected from the set {3, 6, 9, 12, 15, 18, 21, 24, 27}.

HB: We derived 87 instances from the Harwell-Boeing Sparse Matrix Collection. This collection consists of a set of standard test matrices $M = (M_{ij})$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. Graphs are derived from these matrices by considering an edge (i, j) for every element $M_{ij} \neq 0$. From the original set we have selected the 87 graphs with $n \leq 700$. Their number of vertices ranges from 30 to 700 and the number of edges from 46 to 41686.

In the initial set of preliminary experimentation we study the performance of the constructive, improvement and combination methods by experimentally varying their key search parameters, to establish the best configuration of the Scatter Search procedure. We have selected 10 Grid and 10 HB representative instances, with different sizes and densities, to perform the preliminary experimentation. Specifically, we consider *bcsstm07, dwt__361, dwt__592, fs_680_1, lund_a, lund_b, pores_3, saylr3, steam1, steam2, Grid6x21, Grid9x21, Grid9x24, Grid9x27, Grid12x24, Grid12x27, Grid21x9, Grid24x12, Grid27x9, Grid27x12*. We then compare the entire Scatter Search procedure with previous algorithms in the complete set of 252 instances.

	Avg. Q.	**Best Q.**	**Avg. D.**	**CPU Time**
C1+LS_Insert	80.1	62.6	3746.4	287.34
C2+LS_Insert	84.3	61.8	3993.5	295.25
C3+LS_Insert	112.9	81.3	5555.8	435.41
C4+LS_Insert	114.1	79.0	5811.3	388.41

Table 1. Constructive methods.

In our first preliminary experiment we compare the four constructive methods for the CMP described in Section 2.1 (C1, C2, C3 and C4) coupled with the improvement method, LS_Insert, based on their quality and diversity. It is well documented (see for example Laguna and Martí, 2003) that initial solutions in Scatter Search must be of a reasonable quality and sufficiently scattered in the solution space to allow the local search and combination methods to reach different local optima. To test this point, we generate 100 solutions with each constructive plus local search method and compute their quality by simply calculating their objective function value. Then, we compute the average (Avg. Q.) and the minimum (Best Q.) of these values.

The diversity value of each method is computed as the average distance (Avg. D.) between all the pairs of solutions generated (where the distance between two solutions is computed with the expression given in Section 2). Table 2 shows these three values, Avg. Q., Best Q. and Avg. D. as well as the associated CPU Time in seconds for each method. Since we study in this experiment the combination of the constructive and the improvement methods, we set the parameters β to 0 and w to n, for a broader exploration (i.e., all vertices and positions are explored in each iteration).

The results in Table 1 clearly indicate that the C1 and C2 methods coupled with the local search obtain the best solutions with respect to the quality. On the other hand, C3 and C4 are the bests with respect to the diversity and they consume longer running times. C2+LS_Insert is quite balanced, with an average quality of 84.3 (better than C3+LS_insert and C4+LS_Insert) and an average diversity of 3993.5 (better than C1+LS_Insert). We therefore select C2+LS_Insert as the diversification generation method of our SS procedure.

The second preliminary experiment is devoted to adjust the local search parameters β (which determines the number of elements in the candidate list) and w (which determines the number of positions tested). We consider three levels of each parameter: low level of exploration ($\beta = 0.8, w = 0.1$), medium level of exploration ($\beta = 0.5, w = 0.3$) and high level of exploration ($\beta = 0.2, w = 0.5$). Table 2 reports for each level, the average of the objective function (Avg.) and the CPU Time in seconds (CPU).

w \ β		**low**	**medium**	**high**
low	Avg.	89.08	86.53	85.82
	CPU	28.65	58.67	66.06
medium	Avg.	88.88	86.01	85.31
	CPU	82.24	161.17	179.67
High	Avg.	88.88	85.98	85.28
	CPU	115.52	229.14	230.31

Table 2. Local search parameters.

As expected, the lower the β and the larger the w, the better the objective function and the larger the CPU Time. Table 2 clearly shows that we can save a significant amount of running time varying the values of these two parameters with a moderate degradation of the objective function value. Moreover, we can observe that w has a larger impact in the CPU Time than in the objective function, while the behavior of β is more complex. As a compromise selection, we set $w = 0.1$ (low) and $\beta = 0.5$ (medium) in the rest of our experimentation.

In the third experiment, we compare the three different combination methods described in Section 5. For this experiment, we execute the Scatter Search algorithm for 20 iterations ($RefSet$ update or reconstruction) over each instance using C2 as a Diversification Generation Method and LS_Insert with $\beta = 0.5$ and $w = 0.1$. Table 3 reports the average percent deviation from the best-known solution, Dev(%), the number of best known solutions (in HB instances) or optima (in Grid instances) found in this experiment, #Best, and CPU Time in seconds.

	CM1	CM2	CM3
Dev(%)	6.08%	5.58%	4.33%
#Best	8	9	10
CPU Time (s)	351.63	298.54	241.02

Table 3. Combination methods.

The results summarized in Table 3 show that, with respect to the average percentage deviation and number of best solutions, CM3 provides the best results, since it exhibits a value of 4.33% and 10 respectively, which compare favorably with the values obtained with the other methods. Moreover, this variant consumes slightly lower running times than the others.

The objective of next experiment is to test the selective application of the improvement method as well as the percentage of quality-diversity solutions in the $RefSet$. In Scatter Search every solution generated by the Diversification Generation Method or by the Combination Method is typically submitted to the Improvement Method. Since the execution of the improvement method is computationally expensive, applying it to every solution may prevent the search from visiting additional solutions during the allotted search time. Therefore, in this experiment, we test the selective application of the improvement method to a subset of the solutions that are generated by the diversification generation and combination methods. Specifically, we only apply the improvement method to the best b constructed or combined solutions (where $b = |RefSet|$). Table 4 summarizes the results of this experiment using different values of b.

	Improve All			Selective Improvement		
$\|b\|$	**Avg.Dev.**	**#Best**	**CPU Time**	**Avg.Dev.**	**#Best**	**CPU Time**
5	5.14%	9	100.45	8.07%	6	12.45
10	4.33%	10	241.02	7.65%	6	22.73
15	4.59%	10	502.33	7.35%	7	37.01

Table 4. Selective improvement method.

Table 4 reveals that, in the context of the cutwidth problem, the selective improvement is not a good strategy since the "Improve All" strategy systematically obtains better results (in both percentage deviation and number of best solutions) than the "Selective Improvement" strategy although, as expected, the former employs larger CPU Time. On the other hand, the standard value of the *RefSet* cardinality, $b = 10$, provides the best results in the "Improve All" strategy, although it is not conclusive in the "Selective Improvement".

In our next preliminary experiment we apply a full-factorial design to determine whether differences exist in the results of the algorithm for the values of the key search parameters. Specifically, we employ a multifactor ANOVA with the four parameters and the three levels shown in Table 5. The first three parameters, β, b and the combination method, CM, have been identified in the previous experiments. To complement them, we also investigate different percentages of quality versus diversity in the $RefSet$ initial composition. In the Scatter Search literature this percentage, q, is usually set to 50%. In this experiment we test 10%, 50% and 90% (where 10% means that the $0.1b$ solutions are included in the $RefSet$ because of their

quality while the rest of solutions in the $RefSet$ $(0.9b)$ are selected attending to their diversity).

	β	b	CM	q
Level 1	0.2	5	CM1	10%
Level 2	0.5	10	CM2	50%
Level 3	0.8	15	CM3	90%

Table 5. Design of experiment for search parameters.

Considering the average percentage deviation with respect to the best known solution as the dependent variable, and β, b, CM, and q the factors, we apply an ANOVA where the model is limited to the main effects and all two-way interactions. Applying this factorial design to the 20 instances in our set gives a total of 1620 runs. We obtain that β is the most influential parameter in our algorithm, followed by the combination method (both present a large F-value of 50.7 and 8.6 respectively, and a significance of 0.00 < 0.05). On the other hand, the influence of b and q in the results of the algorithm is much more limited, with a significance level larger than 0.05. The strongest interaction is given between β and b with an F-value of 8.7 and a significance level of 0.00, followed by the interaction between β and CM, with an F-value of 3.4 and a significance level of 0.01. Figure 8 shows both interactions.

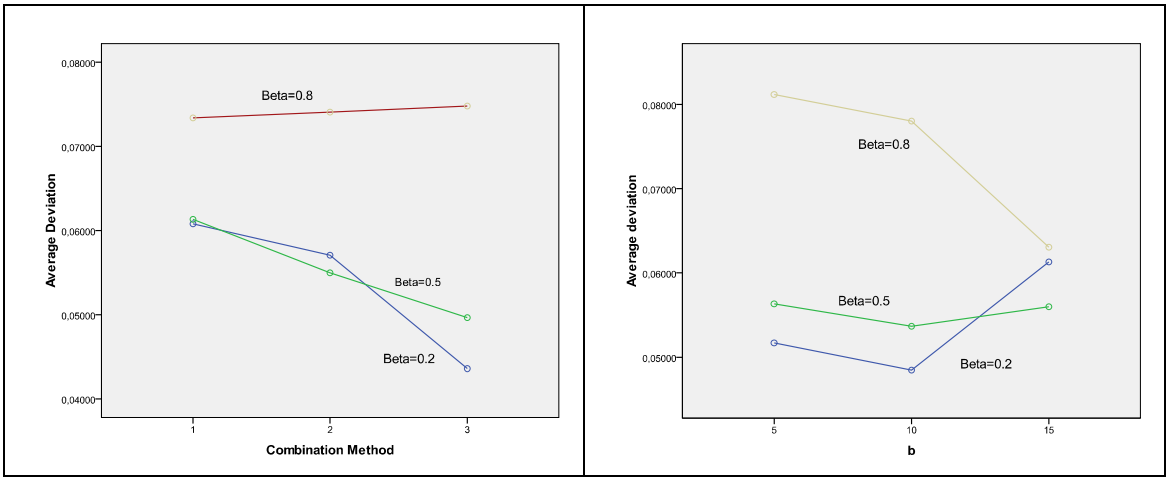

Figure 8. Interaction of factors.

The best parameter setting in this experiment is $\beta = 0.2, b = 15$, CM=3, and $q = 0.9$, which obtains an average percentage deviation of 3.92%. However, if we consider $b = 10$ (and keep the values of the other parameters), the average percentage deviation slightly increments to 4.01% but the associated average running time drops from 446.5 to 238.9 seconds. We therefore consider these values in our final experiment.

We finish our preliminary experimentation studying the contribution of the different SS elements in the evolution of the best solution. Specifically, Figure 9 depicts, for each SS iteration, the value of the best solution in the *RefSet* (Best RefSet), the value of the best solution obtained with the combination of the solutions in the *RefSet* (Best Comb), and the value of the best solution resulting from the application of the improvement method to the combined solutions (Best Improv).

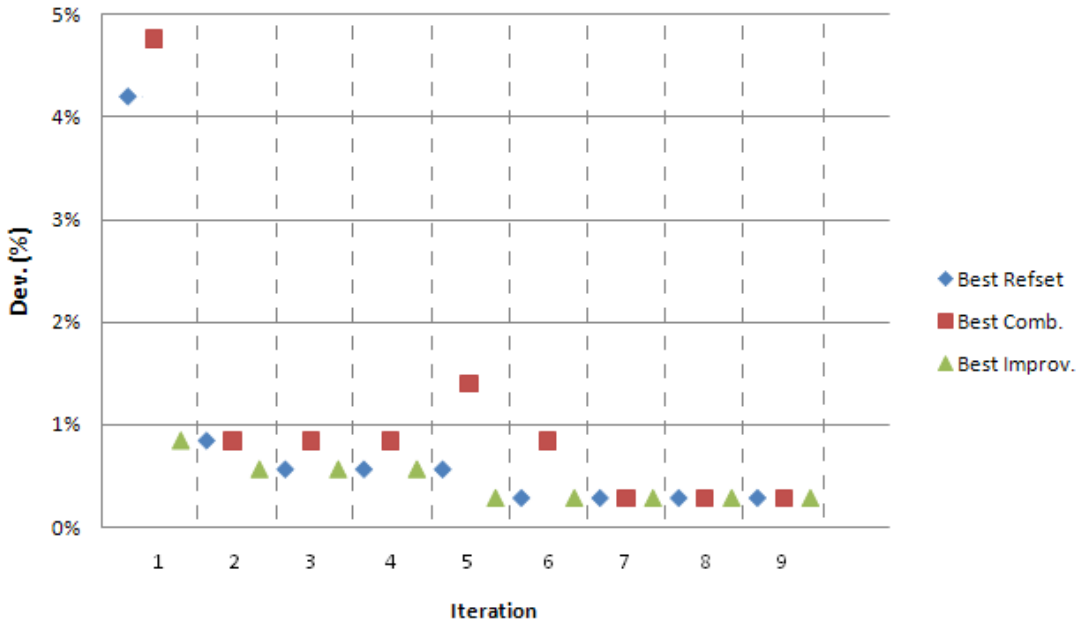

Figure 9. Evolution of the best SS solution

Figure 9 shows the contribution of the combination and improvement methods to the best solution found. This figure clearly shows that the solutions obtained by combination are not able to improve themselves the value of the best solution in the *RefSet*; however, they constitute good seeds for the application of the improvement method. This is especially true in iterations 1 and 5 in which the application of the improvement method to the combined solutions is able to improve the best solution in the *RefSet*, thus generating a new best solution overall.

In the last set of experiments we compare the final design of our Scatter Search with the Simulated Annealing, SA, by Cohoon and Sahni (1985) and the GRASP with Path Relinking, GPR, by Andrade and Resende (2007). The Scatter Search algorithm is set to $\beta = 0.2$, $b = 10$, CM = CM3, $w = 0.1$ and $q = 0.9$ according to the previous experiments, and it stops after 20 iterations. The other two methods are executed for a similar amount of time with the parameters recommended by their authors. Tables 6, 7 and 8 report for the three set of instances respectively, the average of the objective function value, Avg., the average percentage deviation from the best known solution (or the optimum when available), Dev(%), the number of best solutions (optima) that each method is able to match, #Best (#Opt.), and the CPU Time in seconds.

	SS	SA	GPR
Avg.	4.92	5.15	5.2
Dev(%)	0.00%	5.60%	6.54%
#Opt.	84	64	60
CPU Time	0.07	0.07	0.07

Table 6. Final comparison over the set *Small* (84 instances).

	SS	SA	GPR
Avg.	13	16.14	38.44
Dev(%)	7.76%	25.42%	201.81%
#Opt.	44	37	2
CPU Time	210.07	216.13	235.16

Table 7. Final comparison over the set *Grid* (81 instances).

	SS	SA	GPR
Avg.	315.22	346.21	364.83
Dev(%)	1.42%	48.97%	90.97%
#Best	59	8	2
CPU Time	430.56	435.41	557.48

Table 8. Final comparison over the set *HB* (87 instances).

Tables 6, 7 and 8 clearly show that our SS algorithm consistently produces the best solutions in the three types of instances considered. In the Small instances, SS is able to match the 84 optima while SA and GPR obtain 64 and 60 respectively. However, if we allow the methods to run for longer running times, all of them are able to obtain the 84 optima in less than 5 seconds. Grid instances are clearly more difficult to solve than the Small ones. In particular, our SS method obtains 44 optima out of the 81 instances in this set in 210.07 seconds, while SA and GPR obtain 37 and 2 respectively. Finally, in the HB set, where optimum solutions are not known, our method is able to match 59 best known solutions, while SA and GPR obtain 8 and 2 respectively. We refer the reader to the Appendix where these best known values are listed.

To complement this information, we apply a Friedman test for paired samples to the data used to generate these tables. The resulting p-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the three methods tested (we are using the typical significance level of 0.05 as the threshold between rejecting or not the null hypothesis). A typical post-test analysis consists of ranking the methods under comparison according to the average rank values computed with this test. According to this, the best method is the SS (with a rank value of 1.36), followed by the SA (2.05) and finally the GPR (with 2.59 rank value). We now compare SS and SA with two well-known nonparametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former one answers the question: Do the two samples (solutions obtained with both methods in our case) represent two different populations? The resulting p-value of 0.000 indicates that the values compared come from different methods. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting p-value of 0.000 indicates that the SS is the clear winner between both methods.

In the last experiment we compare the performance of the three methods over the time on the instances employed in the preliminary computation. These methods were run for 250 seconds per instance and the best solution found was reported every 10 seconds. The results of this experiment are shown in Figure 10.

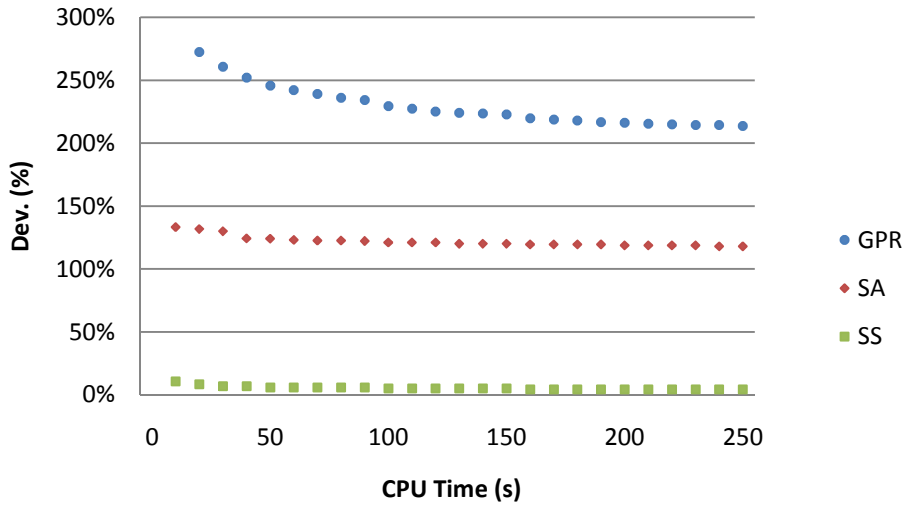

Figure 10. Average best solution value over time

Figure 10 shows that SS is capable of obtaining high quality solutions from the very beginning of the search. Specifically, in the first 10 seconds, it exhibits an average deviation of 10.68%, while GPR and SA show an average deviation of 332.14% and 133.31%, respectively. As it can be observed in Figure 10, the three methods are able to improve its corresponding deviation as the search progresses, with a final value of 213.64% (GPR), 117.98% (SA) and 4.08% (SS) when the time limit of 250 seconds is reached.

7. Conclusions

The Cutwitdth minimization is a computationally difficult optimization problem, which has served us well as test case for a few new strategies that we are proposing to embed in the standard Scatter Search framework. In particular, we tested a GRASP constructive algorithm, a local search strategy based on insertion moves and voting-based combination methods. Moreover, our SS algorithm is also based on a new measure to control the diversity in the search process.

We performed extensive computational experiments to first study the effect of changes in critical Scatter Search elements and then to compare the efficiency of our proposal with previous solution procedures. The comparison with two previous methods based on metaheuristic methodologies favors our proposal.

Acknowledgments

This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (Grant Ref. TIN2009-07516 and TIN2008-06890-C02-02) and by the Comunidad de Madrid (Ref. S2009/TIC-1542).

References

Adolphson D. and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403-423, 1973.

Andrade, D.V. and M.G.C. Resende. GRASP with path-relinking for network migration scheduling. *Proceedings of International Network Optimization Conference*, 2007a.

Andrade, D.V. and M.G.C. Resende. GRASP with evolutionary path-relinking. *Proceedings of Seventh Metaheuristics International Conference* (MIC), 2007b.

Botafogo R. A. Cluster analysis for hypertext systems. 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 116-125, 1993.

Cohoon, J. and S. Sahni. Heuristics for the Board Permutation Problem. *Journal of VLSI and Computer Systems*, 2, 37- 61, 1987.

Feo T.A., and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8:67–71, 1989.

Feo T.A., and M.G.C. Resende. Greedy randomized adaptive search procedures. Journal of Global Optimization, 6:109–133, 1995.

Gavril F. Some NP-complete problems on graphs. *In Proceedings of the 11th conference on information Sciences and Systems*, 91-95, 1977.

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

Karger D. R. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492-514, 1999.

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

Makedon F., C. Papadimitriou, I. H. Sudbourough. Topological bandwidth. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):418-444, 1985.

Makedon F. and I.H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243-265, 1989.

Martí, R., V. Campos and E. Piñana, Branch and Bound for the Matrix Bandwidth Minimization, *European Journal of Operational Research* 186:513-528, 2008.

Piñana E., I. Plana, V. Campos and Rafael Martí: GRASP and Path Relinking for the matrix bandwidth minimization. *European J. of Operational Research* 153(1):200-210, 2004.

Raspaud A., H. Schröder, O. Sýkora, L. Török, and I. Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics, 309:3541-3552, 2009.

Resende, M.G.C. and D. V. Andrade. Method and System for Network Migration Scheduling. *United States Patent Application Publication* US2009/0168665, 2009.

Resende, M.G.C. and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, pages 219–250. Kluwer Academic Publishers, 2003.

Resende, M.G.C. and R. F. Werneck. A hybrid heuristc for the p-median problem. *Journal of Heuristics*, vol. 10, pages 59-88, 2004.

Rolim J., O. Sýkora and I. Vrt'o. Cutwidth of the de Bruijn graph. *RAIRO Informatique Théorique et Applications*, 29(6):509-514, 1995.

Appendix

HB Set			
Instance	**Best value**	**Instance**	**Best value**
494_bus	17	impcol_a	47
662_bus	27	impcol_b	55
685_bus	35	impcol_c	46
arc130	202	impcol_d	64
ash292	36	impcol_e	170
ash85	16	lns__131	32
bcspwr01	5	lns__511	71
bcspwr02	5	lund_a	113
bcspwr03	10	lund_b	111
bcspwr04	29	mbeacxc	16329
bcspwr05	18	mcca	390
bcsstk01	32	nnc261	46
bcsstk02	1089	nnc666	80
bcsstk04	310	nos1	4
bcsstk05	115	nos2	4
bcsstk06	227	nos4	12
bcsstk20	20	nos5	193
bcsstk22	13	nos6	29
bcsstm07	199	plat362	155
can__144	25	plskz362	30
can__161	50	pores_1	17
can__292	96	pores_3	29
can__445	123	saylr1	16
curtis54	13	saylr3	45
dwt__209	58	sherman4	36
dwt__221	27	shl__200	387
dwt__234	12	shl__400	385
dwt__245	27	shl____0	357
dwt__310	26	steam1	182
dwt__361	39	steam2	308
dwt__419	55	steam3	20
dwt__503	138	str__200	560
dwt__592	70	str__600	606
fs_183_1	185	str____0	388
fs_541_1	296	west0132	71
fs_680_1	17	west0156	56
gent113	87	west0167	54
gre_216a	52	west0381	480
gre__115	36	west0479	287
gre__185	48	west0497	181
gre__343	72	west0655	466
gre__512	94	will199	131
hor__131	145	will57	11
ibm32	23		

Table 9. Best known values of HB instances.