Scatter Search for the Profile Minimization Problem

JESÚS SÁNCHEZ-ORO
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
jesus.sanchezoro@urjc.es

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Rafael.Marti@uv.es

ABSTRACT

We study the problem of minimizing the profile of a network and develop a solution method by following the tenets of scatter search. Our procedure exploits the structure of the problem and includes strategies that produce and agile search with computational efficiencies. Among several mechanisms, our search includes path relinking as the basis for combining solutions to generate new ones. The profile minimization problem (PMP) is NP-Hard and has relevant applications in numerical analysis techniques that rely on manipulating large sparse matrices. The problem was proposed in the early 1970s but the state-of-the-art does not include a method that may be considered powerful by today's computing standards. Our extensive computational experiments show that we have accomplished our goal of pushing the envelope and establishing a new standard in the solution of the PMP.

Keywords: Profile minimization, metaheuristics, scatter search.

Original version: May 27, 2012
Revised version: May 23, 2013

1. Introduction

Given a graph or network $G(V, E)$ —where V is a set of n vertices and E is a set of edges— an ordering (or permutation) φ of the vertices is a one-to-one mapping between the set $\{1,2,\dots,n\}$ and V. An ordering in a network can be conceptualized as locating its vertices in a line, as shown in Figure 1. For a given ordering φ, the profile $\delta_{\varphi(i)}$ of vertex $\varphi(i)$ —that is, the vertex in position i— is given by $\delta_{\varphi(i)} = i - f_{\varphi(i)}$, where $f_{\varphi(i)}$ is the position of the left-most vertex adjacent to $\varphi(i)$. If $\varphi(i)$ has no adjacent vertex to its left then $f_{\varphi(i)} = i$. The profile of G with the ordering φ, $P(G, \varphi)$, is the sum of the profiles of all its vertices. The Profile Minimization Problem (PMP) consists of finding the ordering φ^* of V such that the profile is minimized. Mathematically, the PMP seeks φ^*, over the set Φ of all possible permutations, such that:

$$P(G, \varphi^*) = \min_{\varphi \in \Phi} \sum_{i=1}^{n} \delta_{\varphi(i)} = \min_{\varphi \in \Phi} \sum_{i=1}^{n} \left(i - f_{\varphi(i)}\right)$$

Figure 1 shows an example of a network with six vertices ordered in a line, where the first one is labeled as A, the second one as B, and so on. We observe that $f_1 = 1$ because the left-most vertex adjacent to A is C, located in position 3, larger than 1, the position of A. As a consequence, $\delta_1 = 0$. Similarly, $f_2 = 2$ and $\delta_2 = 0$. On the other hand, $f_3 = 1$ and $\delta_3 = 3 - 1 = 2$, because the left-most vertex adjacent to C, in position 3, is A, in position 1. The rest of the calculations are shown in Figure 1 and they results in a profile value of $P(G, \varphi) = 11$.

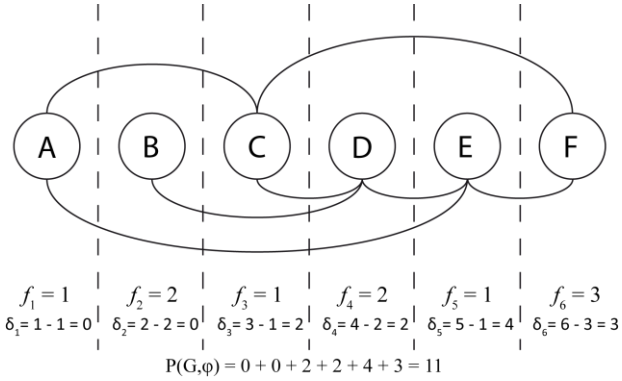

Figure 1. Sample network and profile calculation

It is possible to reduce the profile of the network in Figure 1 by permuting its vertices. Figure 2 shows the same network with the ordering $\varphi^* = (B, D, F, E, C, A)$, which results in an optimal profile value of $P(G, \varphi^*) = 8$.

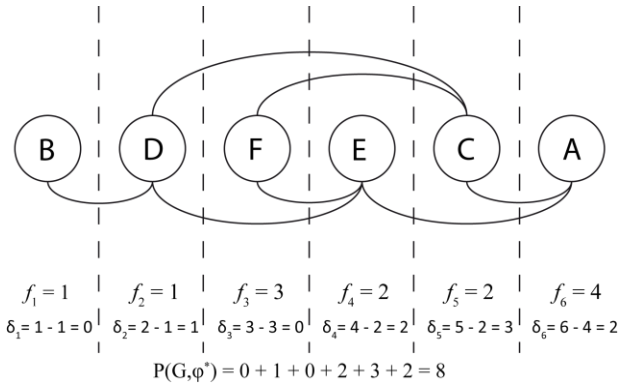

Figure 2. Sample network reordered

Another interpretation of the PMP on a network is based on the notion of *labeling*. Each vertex v in the graph is assigned a label $\pi(v)$ that is nothing else than the position that the vertex would occupy if the vertices were arranged in a line as in Figures 1 and 2. A solution is fully specified when all vertices have been labeled. By definition, the relationship between φ and π is such that if $\varphi(i) = v$ then $\pi(v) = i$. For a given π, the profile of vertex v is calculated as $\delta_v = \pi(v) - f_v$, where f_v is the smallest label of the vertices adjacent to v, as long as such label is smaller than $\pi(v)$. Otherwise, $f_v = \pi(v)$ and $\delta_v = 0$. The objective of the PMP is to find the labeling π^* for which the sum of all vertex profiles is minimized. To facilitate the description of our work, we will employ both interpretations.

The PMP was originally proposed as an approach to reducing the space requirements for storing sparse matrices (Tewarson, 1973). In this context, the PMP was shown to be equivalent to the SumCut problem (Agrawal et al., 1991). One of the main applications of the PMP continues to be the reduction of the space requirements to store systems of equations. Additionally, the PMP enhances the performance of operations on systems of nonlinear equations, such as the Cholesky factorization (Saad, 2003). Another interesting application of the PMP is described by Karp (1993) in the context of the Human Genome Project. The main goals of this project are to identify all genes in human DNA (between 20 and 25 thousand, approximately) and determine the sequences of the approximately 3 billion chemical base pairs associated with human DNA.

In archeology (Kendall, 1969), the PMP has been utilized in connection with the problem of organizing items such as fossils, tools and jewels according to a specific order. This process is known as “seriation” and consists of placing several items from the same culture in a chronological order determined by a method of establishing dates that are relative to each other. This results in a problem for which the reordering of the rows and columns of a matrix is required, which is equivalent to the reordering of a linear graph. Other applications of the PMP can be found in information retrieval (Botafogo, 1993) and fingerprinting (Karp, 1993).

Lin and Yuan (1994) proved that the PMP of an arbitrary graph is equivalent to the interval graph completion problem, which was shown to be NP-complete by Garey and Johnson (1979). However, special classes of graphs can be solved optimally in polynomial time. For example, Lin and Yuan (1994) proposed several polynomial-time algorithms to find the optimal solution of the PMP for paths, wheels,

complete bipartite graphs and D4-trees (trees with diameter 4). Likewise, Guan and Williams (2003) developed an algorithm to find the optimal solution of the PMP for triangulated graphs.

The SumCut problem (Gibbons et al., 1991; Penrose et al., 2001; Petit et al., 2002) and the PMP are equivalent in the sense that a solution to one problem provides a solution to the other problem by reversing the corresponding permutation. Consequently, the optimum of one problem corresponds to the optimum of the other (Agrawal et al., 1991). The profile minimization problem is also related to the Bandwidth Reduction Problem (BRP), which consists of finding a permutation of the rows and the columns in a matrix, such that all the non-zero elements are confined to a band that is the closest to the main diagonal.

Algorithmically, the PMP has been tackled since the late sixties. To the best of our knowledge, the first heuristic in the literature consists of a constructive procedure proposed by Cuthill and McKee (1969) known as the Reverse Cuthill-McKee algorithm (RCM). Their procedure is based on constructing a level structure of the vertices. Poole, et al. (1976) improved the RCM algorithm by changing the selection of the root node within a more general level structure. The method is known in the literature as GPS. Gibbs (1976) developed GK, a procedure that uses a pseudo-diameter to produce a new level structure. GK outperforms GPS in solution quality but it requires more computational time. Lewis (1982) introduces new implementations that improve the performance of both GK and GPS. His experimental results confirm the superiority of GK over GPS in terms of solution quality.

The best heuristic for the PMP in the literature belongs to Lewis (1994). It uses the Simulated Annealing (SA) methodology in combination with existing constructive procedures. The SA search starts from a solution constructed with RCM or GK, whichever is better according to the objective function value. In typical SA fashion, neighborhood exploration is performed with moves that are randomly generated. Improving moves are always executed and non-improving moves are only executed with a probability that depends on the current temperature. A so-called cooling schedule controls the systematic reductions of the temperature and the search ends when the temperature reaches a pre-specified minimum level.

2. Scatter Search

Scatter search (Laguna and Martí, 2003) is a metaheuristic whose framework includes five methods that are designed to build, maintain and transform a population of solutions (see Figure 3). Three of these methods, the Diversification Generation, the Improvement and the Combination Methods, are problem dependent and therefore their strategies take advantage of information that is context-specific. On the other hand, the Reference Set Update and the Subset Generation Methods have standard implementations that are context-independent. The scatter search (SS) literature keeps growing with examples of successful applications, such as those documented in Gallego, et al. (2009), Martí et al. (2009) and Duarte, et al. (2010 and 2011).

```
1. Diversification generation
2. Improvement
3. Reference set update
while (termination criteria not satisfied)
   4. Subset generation
   5. Combination
   6. Improvement
   7. Reference set update
```

Figure 3. Scatter search framework

The procedure starts with the application of the Diversification Generation method (which we describe in Section 3 in the context of the PMP) and the Improvement method (described in Section 4), obtaining as a result a population P of solutions from which the initial reference set ($RefSet$) of size b is constructed. The initial $RefSet$ must balance solution quality and diversity and therefore the standard update in step 3 (Figure 3) selects the best $b/2$ solutions from P and then the $b/2$ solutions in $P \backslash RefSet$ that are most diverse with respect to those solutions already in the reference set. We point out that selecting the most diverse solutions from a set is an NP-hard problem (see for instance Duarte and Martí, 2007) and hence we perform this step heuristically. In particular, after selecting the $b/2$ best solutions (i.e., the ones with the best objective function value), the remaining solutions are added one at a time. The first one to be added is the solution in $P \backslash RefSet$ that is the most diverse with respect to the solutions currently in $RefSet$. Diversity is typically measured with a function that maximizes the minimum distance between the solution under consideration and the set of solutions where the solution will be added (in this case $RefSet$). As discussed above, a solution s to the PMP is fully characterized by its ordering φ_s and equivalently by the labeling π_s of the vertices in G. The distance between s and $RefSet$ is given by:

$$d(s, RefSet) = \min_{r \in RefSet} \left(\sum_{v=1}^{n} |\pi_s(v) - \pi_r(v)| \right)$$

The distance is the sum of the absolute differences of the labels assigned to each vertex in the candidate solution s and all the reference solutions r. Since the labels represent positions, the calculation is equivalent to the so-called *positional distance* (Das and Roberts, 2004). Once a solution s is selected from $P \backslash RefSet$, it is added to $RefSet$ and the process is repeated $b/2$ times, choosing at each step the solution s^* with the maximum distance to the solutions currently in the reference set, that is:

$$s^* = \arg \max_{s \in P \backslash \text{RefSet}} d(s, RefSet)$$

In our implementation, step 4 in Figure 3 consists of generating all pairs of reference solutions that have not been combined before. Details about the combination method (based on the path relinking methodology) are presented in Section 5. The reference set update in step 7 is different from the updating performed in step 3. In order to maintain the diversity among the solutions in $RefSet$, a

solution φ is admitted if it improves the best solution in it, or alternatively, if it improves the worst solution and its distance with the closest solution to φ in $RefSet$ is larger than a pre-established threshold $dthresh$. If solution φ qualifies to enter in *RefSet*, then it replaces the closest solution with objective function value lower than or equal to φ.

The process terminates when no new solutions become part of $RefSet$. That is, if at a given iteration, $RefSet$ does not change after step 7, then the search ends.

3. Diversification Generation Method

We develop four diversification generation methods (labeled C1 to C4) to build a population P of solutions. These methods are based on the GRASP methodology (Feo and Resende, 1989; Resende, Smith and Feo, 1994; Resende, et al. 2010). To describe these methods we define U as the set of vertices that have not been labeled and $L = V \setminus U$ as the set of the vertices that have already been labeled. Initially, all vertices are in U (i.e., $U = V$). C1 starts by selecting the vertex $v \in U$ with the smallest degree (i.e., the vertex with the least number of adjacent vertices). In this step, ties are broken arbitrarily. The chosen vertex v is given the first label and therefore $\pi(v) = 1$ and $\varphi(1) = v$. Then $U = U \setminus \{v\}$ and $L = L \cup \{v\}$ and a candidate list CL consisting of the set of vertices adjacent to v is constructed:

$$CL = \{u: (v, u) \in E\}$$

A greedy function value is calculated for each vertex v in CL as follows:

$$g_1(v) = |N_L(v)| - |N_U(v)|$$

where $N_L(v) = \{u \in L: (v, u) \in E\}$ and $N_U(v) = \{u \in U: (v, u) \in E\}$. The greedy function $g_1(v)$ measures the level of "urgency" of labeling vertex v next. The function considers that it is more urgent to label a vertex for which most of its adjacent vertices have already been labeled. A greedy procedure would choose, at each step, the vertex v with the largest $g_1(v)$ value. However, in the GRASP framework, constructions are semi-greedy and the implementation includes a so-called restricted candidate list (RCL). The RCL includes top candidates that are equally preferred and hence equally likely to be chosen:

$$RCL = \{v \in CL: g(v) > g_1^{min} + \alpha_1(g_1^{max} - g_1^{min})\}$$

where g_1^{min} (g_1^{max}) is the minimum (maximum) value of $g_1(v)$ for all v in CL. A vertex v from RCL is chosen randomly, then the next available label is assigned to it, and the U and L sets are updated. CL is also updated by adding the unlabeled vertices that are neighbors of the selected vertex v (i.e. $CL = CL \cup N_U(v)$). The construction ends when all vertices have been labeled.

The second construction procedure (C2) is similar to C1 but implements in a different way the semi-greedy selection. Instead of calculating the greedy function value first and then randomly selecting from a restricted candidate list, C2 first takes from CL a random sample of vertices. Then, the vertex with the largest $g_1(v)$ value in the sample is selected. The size of the random sample is controlled by the

parameter α_2, which represents a fraction of the size of the candidate list (i.e., the random sample includes $\alpha_2|CL|$ vertices). As before, once the vertex is selected, the next available label is assigned to it and the U and L sets are updated.

The $g_1(v)$ greedy function does not take into consideration the values of the labels assigned to the vertices adjacent to v. For instance, if i is the next available label, $g_1(v)$ does not include in its calculation the labels that vertices in $N_L(v)$ have received (which may be any between 1 and $i-1$). Clearly, the "urgency" of vertex v is greater if its adjacent vertices have received labels that are much smaller than i. The following greedy function takes this into consideration:

$$g_2(v) = (|N_L(v)| - |N_U(v)|) \sum_{u \in N_L(v)} |\pi(v) - \pi(u)|$$

This greedy function is used to formulate two additional construction methods. C3 is the same as C1 but with g_2 as the greedy function. C4 is the same as C2 but with g_2 as the greedy function.

4. Improvement Method

For our improvement method we tested both a search neighborhood defined by swap moves and one defined by insert moves. The representation of a solution as an ordering of the vertices is useful to conceptualize these search neighborhoods. A $swap(i,j)$ is the exchange of positions of vertices v, u that are currently in positions i and j, respectively. That is, $\varphi(i) = v$ and $\varphi(j) = u$. An $insert(i,j)$ is the movement of vertex $v = \varphi(i)$ to position j. After the move, v precedes u if $i > j$ and v follows u if $i < j$. Since both of these moves produce a neighborhood of size $\mathcal{O}(n^2)$, a fast calculation of the move value is critical in order to search such a neighborhood efficiently and identify the best move to make.

For instance, to evaluate the move $insert(i,j)$ for $j > i$, it is easy to see that only the profile of the vertices v for which $i \leq \pi(v) \leq n$ needs to be recalculated. Therefore, if the profile values for all vertices in the current solutions are stored, then the calculation of the move value may be done by updating only those relevant profile-values and adding them to the values that the move does not affect. Additionally, the updates to the profile values may be accumulated. For instance if we first evaluate the $insert(i,j)$, then $n-i+1$ values must be recalculated, i.e., those corresponding to the vertices with labels between i and n. If we then evaluate $insert(i,j+1)$ without storing any information from the previous evaluation, we would need to recalculate once again $n-i+1$ values. However, observing the change of the profile values from one trial move to the other, it can be determined that the only changes occur in the vertices with labels $\pi(v) \geq j+1$.

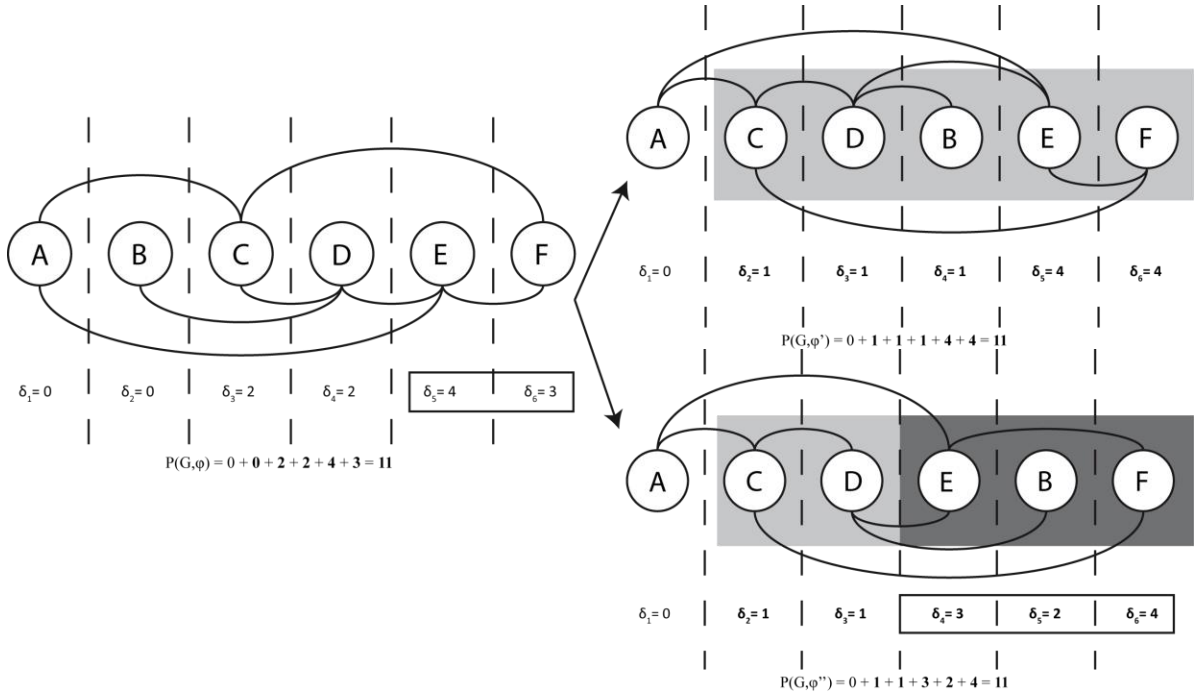

Figure 4. Illustrative representation of a move

Figure 4 illustrates two insertion moves applied to solution $\varphi = \{A, B, C, D, E, F\}$, on the left side of the figure. The first move inserts vertex B in position 4, obtaining the solution $\varphi' = \{A, C, D, B, E, F\}$ at the top right side of Figure 5, where vertices that changed their δ_i value are highlighted (i.e., vertices C, D, B, E and F). The insertion of B in position 5 that yields a new solution φ'' is depicted at the bottom right of Figure 4. Considering that the improvement procedure evaluates insertions as a sequence of swap moves, evaluating the insertion of B in position 5 after evaluating the insertion of B in position 4 requires the updating of vertices E, B and F, only. Calculation savings are achieved because there is no need to update C and D again.

Therefore, an efficient way of searching the insert neighborhood is to evaluate the $insert(i, j)$ as a sequence of swaps of the vertices in positions $(i, i+1)$, then $(i+1, i+2)$ and so forth until $(j-1, j)$. These values are stored because the sequence is the same for $insert(i, j+1)$ with the addition of the $swap(j, j+1)$. We have implemented this strategy and the corresponding one for the case when $j < i$.

By implementing the search as a sequence of swaps of vertices in adjacent positions, the only possible moves values are -1, 0 and +1. Consider the $swap(i, i+1)$ that exchanges the labels of vertex v from $\pi(v) = i$ to $i+1$ and vertex u from $\pi(u) = i+1$ to i. Then, the move value is given by:

$$value(i, i+1) = \begin{cases} -1 & \text{if } f_v > i \text{ and } f_u < i+1 \\ 0 & \text{if } (f_v > i \text{ and } f_u \geq i+1) \text{ or } (f_v \leq i \text{ and } f_u < i+1) \\ +1 & \text{if } f_v \leq i \text{ and } f_u \geq i+1 \end{cases}$$

This characteristic renders a *first improving* strategy impractical. A first improving refers to the strategy of stopping the neighborhood search after finding the first move that improves the current solution. Since our neighborhood search is structured in such a way that complex moves (i.e., $insert(i, j)$ for a

$j > i + 1$) are achieved by a sequence of simple moves ((i.e., $swap(i, i + 1)$), stopping after a finding any improvement results in a process where improvements of more than one unit are not possible in a single move. Therefore, our implementation uses the *best improving* strategy in which the entire neighborhood is searched and the move with the best value is selected.

5. Combination Methods

We have developed one combination method (CM1) and a variant (CM2). The combination method follows the strategy known as path relinking (Glover and Laguna, 1997). The main idea behind path relinking (PR) is to construct a path between two elite solutions where the guide is not limited to the value of the objective function of the problem. PR was suggested in connection with tabu search because choice mechanisms in neighborhood-based searches often use the objective function as the only oracle to measure the quality of a move (just as we do in the improvement method described above). PR attempts to create search paths where the objective function is only one of the elements employed to determine the direction. PR also exploits the principle that the neighborhood of an elite solution might contain other high-quality solutions that could be found if the elite solution is approached from a direction that is different from the one that was used to find the elite solution in the first place. PR is implemented by choosing one or more elite solutions as the initiating solution and one or more as the guiding solutions. Strategies are then built around the notion of applying transformations to the initiating solution with the goal of moving it toward the guiding solution(s). Since all solutions that have been found during a search are connected by the path that the search followed to find them, building a new path between elite solutions may be conceptualized as a relinking exercise, and hence the name of the strategy.

Let s be the initiating (or starting) solution and t the guiding (or target) solution. CM1 starts by identifying the set D of vertices that have different labels in s and t, that is, $D = \{v: \pi_s(v) \neq \pi_t(v)\}$. A set of solutions is generated by swapping the labels of vertex v and vertex $\varphi_s(\pi_t(v))$ in the initiating solution, for all $v \in D$. Note that after these swaps, $\pi_s(v)$ equals $\pi_t(v)$ and the initiating solution moves closer to the guiding solution. This step generates $|D|$ trial solutions from which we select the best according to the objective function value. The chosen solution becomes the new initiating solution and the process continues until $D = \emptyset$, that is, until the labeling in s is the same as in t. The procedure, referred to as Greedy Path Relinking in (Resende, et al. 2010), returns the best trial solution found.

Figure 5 illustrates how CM1 operates on two solutions, where s is the initiating solution and t is the guiding solution. The relinking requires 4 steps and generates 12 intermediate solutions that are labeled with a digit and a letter, where the digit is the step number and the letter a solution identifier. Note that at each step, the best solution (according to the objective function value) becomes the initiating solution for the next step. Once the guiding solution is reached, the best intermediate solution (in this case solution 3A with a profile value of 8) is returned.

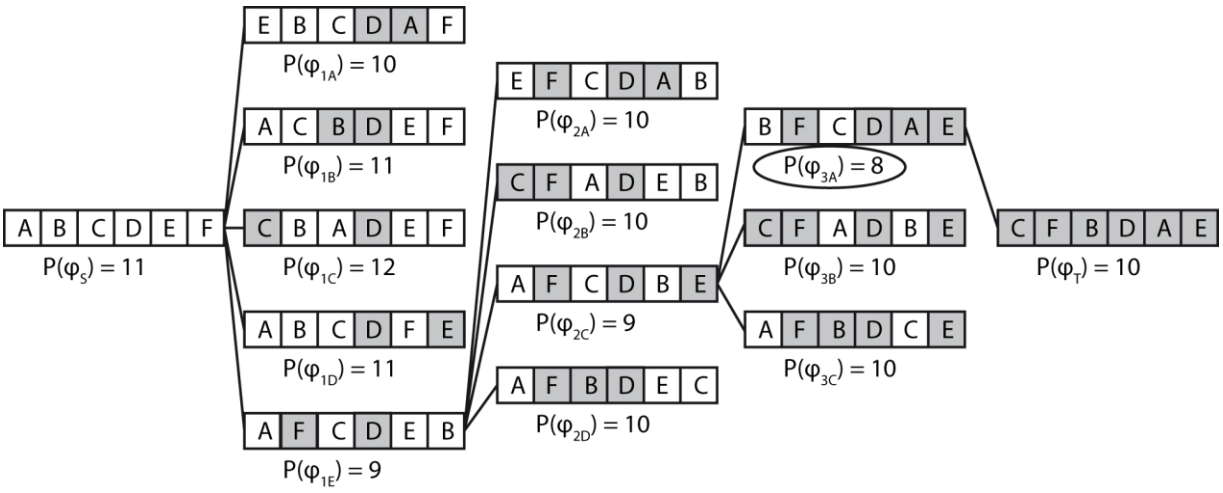

Figure 5. Illustration of the combination method CM1

We created a variant of CM1 that we refer to as CM2 and that consists of replacing the selection criterion of the trial solution generated during the relinking process. In particular, instead of selecting the best solution with respect to the objective function value from the set D of trial solutions, the next initiating solution is selected randomly. In this way, CM2 favors diversification over intensification. As before, CM2 returns the best solution encountered during the path relinking process.

6. Computational Experiments

We now describe the computational experiments performed to test the SS approach that we developed for the PMP and then we discuss the associated results. The procedure was implemented in Java SE 6 and was compared against the best methods reported in the literature so far, namely, RCM (Cuthill and McKee, 1969), and SA (Lewis, 1994). The RCM and SA results were obtained running the original C implementations shared by the authors. We have implemented the Scatter Search (SS) procedure in Java SE 6. All tests were performed on an Intel Core i7 2600 machine running at 3.4 GHz and with 2 GB of RAM. The test set consists of 262 instances divided into three subsets. All instances are available at www.optsicom.es/pmp/.

- HB — This set is derived from 73 instances in the Harwell-Boeing Sparse Matrix Collection (Matrix Market, 2011). The data matrices in this set correspond to problems in linear systems, least squares and eigenvalue calculations in a wide variety of scientific and engineering disciplines. We selected the 73 problems with $n \leq 1000$ and therefore the number of vertices ranges from 24 to 960 and the number of edges ranges from 34 to 3721.
- K-Graphs — This set contains 98 bipartite graphs with number of vertices ranging from 4 to 142 and number of edges ranging from 3 to 5016. A bipartite graph is such that the set of vertices V can be divided into two subsets V_1 and V_2 in such a way there exists an edge between every pair of vertices, one belonging to V_1 and the other belonging to V_2. At the same time, there is no edge for which its endpoint vertices are in the same subset. Optimal solutions are known by

construction (Yixun and Jinjiang, 1994) and are given by $n_1 n_2 + \frac{1}{2} n_1(n_1 - 1)$ for $n_1 \leq n_2$, where $n_1 = |V_1|$ and $n_2 = |V_2|$.

- D4-Trees — This set consists of 91 instances that are based on trees with a diameter of 4 and a number of vertices ranging from 10 to 100 and a number of edges ranging from 9 to 99. These trees are built by choosing a root vertex v_0 and adjacent vertices v_1 to v_k, for $k \geq 2$. The vertices adjacent to v_0 form a star with v_0 in the center. If $\gamma(v_i)$ is the degree of vertex v_i then the v_i-branch has $\gamma(v_i) - 1$ leafs, because the other edge is the one connecting v_i to the root vertex v_0. The trees in the D4 set are built in such a way that $\gamma(v_1) \geq \gamma(v_2) \geq \cdots \geq \gamma(v_k) \geq 2$ and with a diameter of 4. This means that the tree consists of the root vertex, of all the adjacent vertices in a star configuration and of all the vertices corresponding to the leaves of the tree. The optimal solutions are known by construction and are given by $|E| + \sum_{i=3}^{k} (\gamma(v_i) - 1)$.

The experiments are divided into two main blocks. The first block has the goal of studying the behavior of the components of the solution procedure as well as determining the best values for the search parameters. The second block of experiments has the goal of comparing our procedure with the best in the literature. To be able to test the effectiveness of our strategies in the entire PMP class, the first set of experiments is performed on a subset of all problem instances. In this way, we can test how well our choices generalize to the entire set of problems. Specifically, the tuning experiments are performed on the 30 HB instances that have less than 250 vertices. We refer to these instances as the *training* set and to all other instances as the *testing* set.

A common practice in scatter search implementations is to arrive to the best design by choosing the components of the solution procedure sequentially. Since we are using standard implementations for the reference set update and the subset generation methods, the design process focuses on choosing a diversification generator, an improvement method and a combination method. As described in the previous sections, we have developed four diversification generation methods (C1 to C4), two improvement methods (one based on swaps and one based on inserts) and two combination methods (CM1 and CM2). This results in a full factorial design with 16 combinations. In the first block of experiments, we will contrast the sequential process that selects components one at a time and adds them to the solution procedure with the full factorial approach that identifies the best combination with a single experiment.

For the sequential design process, we begin by comparing the four construction methods described in Section 3. Since these methods are used for diversification purposes, the comparison metrics must include a measure of diversity as well as a measure of solution quality. Each method (C1 to C4) is executed to generate 100 solutions. The value of α is chosen randomly in each construction. For validation purposes, we also generate 100 solutions at random and refer to this set of solutions as C0. The objective function value of each solution is used to calculate an average quality of each set, which is then normalized between 0 and 1. The average diversity of a set of solutions is calculated using the distance measure described in Section 2. That is, for a particular set of solution, the distance between a solution and the rest in the set (i.e., the other 99 solutions) is computed and then the average of the 100 values obtained is used to represent the diversity of the set. We also normalize this diversity measure between 0 and 1. Figure 6 compares the diversity and quality of the construction methods.

Not surprisingly, C0 obtains the maximum diversity of all the sets. However, this set of solutions is of minimum quality when compared to the constructions based on GRASP. The C2 set has the highest quality but the lowest diversity. The sets generated by C3 and C4 are very similar, with C3 slightly more diverse and C4 with slightly higher quality. In order to test the effect of adding the improvement method to these construction procedures, we select C2 and C3 to perform additional experiments. Ignoring the purely random generator C0, the set generated by C2 and C3 are at the extremes of the frontier in which C1 is dominated and C4 is the most balanced non-dominated point.

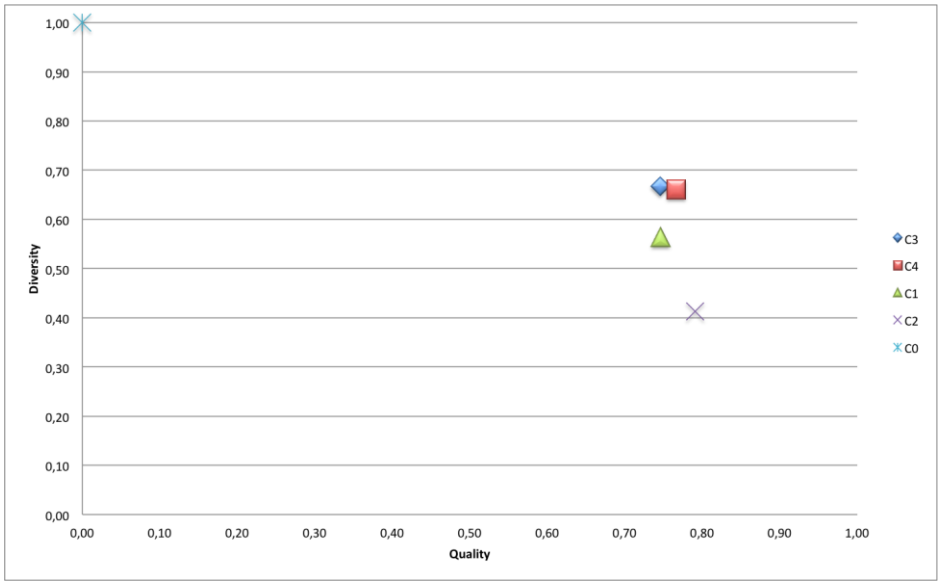

Figure 6. Comparison of construction methods

The second experiment couples the construction methods C2 and C3 with the improvement methods based on swap and insert neighborhoods, resulting in 4 different variants. Once again, the α values associated with C2 and C3 are chosen randomly in each construction. Table 1 shows results obtained when applying these variants to the training set. For each variant, the table reports the average objective function value (Obj), the percent deviation of this value from the average obtained with the best known solutions (Dev), the number of best solutions found out of 30 (#Best) and the CPU time in seconds.

Variant	Obj	Dev (%)	#Best	CPU Time
C2+Swap	1235.94	4.61	7	68.76
C3+Swap	1209.72	2.05	10	90.84
C2+Insert	1209.34	1.30	17	2.56
C3+Insert	1194.78	0.67	18	3.90

Table 1. Construction procedures coupled with improvement methods

The improvement method based on inserts seems to be more effective than the one based on swaps according to the results in Table 1. The table also shows, however, that inserts are more computationally expensive than swaps. The best combination of construction and improvement is given by C3+Insert, CPU time notwithstanding. Therefore, in this sequential process, we have determined, so

far, that our final scatter search should have C3 as the diversification generator and Insert as the improvement method.

Finally, we must choose a combination method that works well with the C3+Insert. For this experiment, we also need to add the subset generation method and the reference set update method. The subset generation method is a standard implementation and has no parameters. The behavior of the reference set update method depends on the values of b and the $dthresh$ parameter. While we set b to the standard value of 10 used in the SS literature, in this experiment, we try 4 values for $dthresh$ (0.01, 0.05, 0.10 y 0.30). As indicated at the end of Section 2, the search terminates when no solution generated by the application of the combination and improvement methods is admitted to the reference set. Table 2 summarizes the results of this experiment, where the column labels have the same meaning as in Table 1 with the addition of the $dthresh$ column containing the value of the parameter that was utilized to produce the results, where MD is the maximum distance between the labeling of two solutions. Considering the distance function formulated in Section 2, MD is computed as follows:

$$MD = \sum_{i=1}^{n} |i - (n - i + 1)|$$

According to the results shown in Table 2, the best SS configuration should include CM1 as the combination method. Within that, the best value for $dthresh$ seems to be 0.05, which results in the smallest deviation from the best solutions known. The results indicate that while there is a significant difference in performance between employing CM1 and employing CM2, the procedure is robust with respect to the value of $dthresh$. In particular, there is no significant difference between values in the range between 0.01 and 0.10 for CM1.

Variant	$dthresh$	Obj	Dev(%)	#Best	CPU Time
C3+Insert+CM1	0.01*MD	1187.97	0.10	25	6.54
	0.05*MD	1187.38	0.08	25	6.37
	0.10*MD	1187.63	0.09	24	6.71
	0.30*MD	1188.16	0.12	24	6.73
C3+Insert+CM2	0.01*MD	1191.69	0.43	21	6.08
	0.05*MD	1191.56	0.43	21	5.74
	0.10*MD	1191.78	0.45	21	5.90
	0.30*MD	1191.56	0.43	20	5.43

Table 2. Performance of combination methods CM1 and CM2

The final experiment of this block consists of choosing the best SS configuration by running a single full-factorial experiment. To perform this experiment, we set $b = 10$, $dthresh = 0.05$ and α is chosen randomly in each construction. Table 3 summarizes the results obtained with the 16 variants on the training set.

Variant	Obj	Dev(%)	#Best	CPU Time
C1+Insert+CM1	1189.44	0.80	20	6.59
C1+Insert+CM2	1195.38	0.96	18	5.30
C1+Swap+CM1	1195.94	1.35	13	92.68
C1+Swap+CM2	1200.91	1.74	12	91.24
C2+Insert+CM1	1218.31	3.30	11	3.79
C2+Insert+CM2	1219.41	3.51	12	3.48
C2+Swap+CM1	1228.91	4.42	9	76.20
C2+Swap+CM2	1232.91	4.74	7	75.36
C3+Insert+CM1	1188.06	0.75	17	6.78
C3+Insert+CM2	1191.53	1.04	14	6.58
C3+Swap+CM1	1201.91	1.79	10	101.27
C3+Swap+CM2	1202.78	1.95	12	96.13
C4+Insert+CM1	1195.28	1.27	14	6.91
C4+Insert+CM2	1200.97	1.50	15	6.14
C4+Swap+CM1	1205.44	2.25	10	94.01
C4+Swap+CM2	1205.66	2.44	10	93.00

Table 3. Full factorial experiment

The output of the full factorial design that Table 3 summarizes indicates that there are two configurations that dominate all others when considering percent deviation and number of best solutions: C1+Insert+CM1 and C3+Insert+CM1. The C3+Insert+CM1 configuration was the one identified as the best by the sequential design process. It achieves the lowest deviation of 0.75% and the third highest number of best solutions of 17. The C1+Insert+CM1 configuration achieves the highest number of best solutions of 20 and an average deviation of 0.8% that is second lowest. The configuration uses a diversification generator that is different from the one that we selected during the sequential design process. In fact, in our analysis, C1 was dominated by the other three construction procedures in terms of both quality and diversity (see Figure 6). Hence, the full factorial design is able to identify a configuration that performs well when all elements are put together at once but that does not seem attractive when the merit of each element is assessed separately. Nonetheless, the fact that the C3+Insert+CM1 configuration is in the non-dominated set seems to indicate that in situations where running a full factorial design is not practical, a sequential process is an approach for which it is reasonable to expect that it will yield an effective combination of SS components. This analysis concludes the first block of experiments.

In the second block of experiments we compare the performance of the SS procedure configured as C1+Insert+CM1 against the best methods in the literature. In particular, the comparison includes RCM (Cuthill and McKee, 1969) and SA (Lewis, 1994). Tables 4 and 5 show the results associated with the K-Graphs and D4-Trees data sets, respectively. The optimal solutions to these problems are known by construction and therefore the deviations are calculated against the optimal objective function values. Also, the tables report the number of optimal solutions found (#Opt) instead of the number of best solutions.

Procedure	Obj	Dev(%)	#Opt	CPU Time
RCM	1165.64	0.00	98	1.02
SA	1167.06	0.02	97	5.87
SS	1165.64	0.00	98	0.66

Table 4. Experiments with 98 K-Graphs

Clearly, the K-Graphs do not represent a challenge to any of the procedures in our test. Only SA fails to find the optimal solutions to one of the instances, even when employing considerably more time than its counterparts. The situation changes when the methods are applied to the D4-Tree set. The results are shown in Table 5 where the difficulty of these problems becomes evident. The best performance is achieved by SS, which is able to find the optimal solution to 97.8% (i.e., 89 out of 91) of the problems. The solution time is negligible for all procedures.

Procedure	Obj	Dev(%)	#Opt	CPU Time
RCM	282.75	173.57	2	1.01
SA	126.08	30.39	31	0.68
SS	86.12	0.01	89	0.28

Table 5. Experiments with 91 D4-Trees

We perform non-parametric tests to provide additional support to our conclusions about the performance of the scatter search implementation. First, we apply the *Friedman test* for multiple correlated samples to the best solutions obtained by each of the 3 methods in Table 5. This test computes, for each instance, the rank value of each method according to solution quality (where rank 1 is assigned to the best method and rank 3 to the worst). Then, it calculates the average rank values for each method across all instances. If the averages differ greatly, the associated *p*-value or level of significance is small. The resulting *p*-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the 3 methods. The rank values produced by this test are 1.18 (SS), 1.85 (SA), and 2.97 (RCM).

Next, we employed the *Wilcoxon test* and *Sign test* to make a pairwise comparison of SS and SA, which consistently provide the best solutions reported in our experiments. The results of the *Wilcoxon* test (with a *p*-value of 0.000) determined that the solutions obtained by the two methods indeed represent two different populations. The *Sign* test (with a *p*-value of 0.000) indicated that the objective function values of the solutions obtained with SS tend to be better (i.e., smaller) than those obtained with SA.

In the final experiment of this block, we add a procedure to our comparison set. We refer to the additional procedure as TS-BMP because it was developed by Campos et al. (2011) for the solution of the bandwidth minimization problem (BMP). The BMP is related to the PMP because they both have the goal of transforming a sparse symmetric matrix into one for which the nonzero elements are close to the main diagonal. The PMP achieves this goal with an additive objective function that penalizes a distance measure from the main diagonal while the BMP uses a criterion that minimizes the maximum deviation. Specifically, in the BMP, the objective is to find an ordering of the rows and columns of a matrix M in such a way that the nonzero elements are in a band that is as close as possible to the main

diagonal. In other words, the goal is to find a labeling π of the vertices of the corresponding $G(V,E)$ graph such that bandwidth, $B(G,\pi)$, is minimized, where:

$$B(G,\pi) = max(B(v,\pi): v \in V) \text{ and } B(v,\pi) = max(|\pi(v) - \pi(u)|: (v,u) \in E).$$

That is, $B(v,\pi)$ is the bandwidth of vertex v for labeling π and it is calculated as the maximum absolute difference between label of v and the labels of its adjacent vertices. The bandwidth of the graph is the maximum vertex bandwidth. Given these similarities, it is reasonable to believe that a solution procedure design for the BMP might find high quality solutions to the PMP. To test this, we applied TS-BAND without any modifications to the HB instances. TS-BMP operates with the objective of minimizing $B(G,\pi)$ and upon termination the procedure returns the solution π^* with the best value according to this objective function. We then calculate $P(G,\varphi^*)$ by applying the transformation $\varphi^*(\pi^*(v)) = v$ for all vertices in the graph. The results associated with this experiment are in Table 6. Since the optimal solutions to the HB instances are not known, the deviations are calculated against the best-known solutions and #Best refers to the number of best-known solutions found by each procedure.

Procedure	Obj	Dev(%)	#Best	CPU Time
TS-BMP	10046.10	55.52	1	240.60
RCM	8441.78	31.31	7	250.00
SA	8458.41	28.93	7	390.61
SS	6749.56	0.38	67	214.90

Table 6. Experiments with 73 HB instances

When the results shown in Table 6 are added to the results of our previous experiments, they support for the position of SS as the best procedure for the PMP of those in the comparison set. However, the following statistical tests provide additional evidence.

We applied the three statistical tests described above to the results in Table 6. The *p*-value of 0.000 obtained for this experiment with the Friedman test clearly indicates that there are statistically significant differences among the 4 methods. The rank values produced by this test are 1.19 (SS), 2.52 (RCM), 2.62 (SA), and 3.67 (TS-BMP). Finally, we employed the *Wilcoxon test* and *Sign test* to make a pairwise comparison of SS and SA for the results reported in Table 6. Both obtained a *p*-value of 0.000, indicating that the solutions obtained with SS are consistently better than those obtained with SA. Table 7 in the Appendix shows the best values and associated CPU times obtained with SS on the HB instances.

7. Conclusions

Our goal was to develop a state-of-the-art solution method for the profile minimization problem. We accomplished this goal with an implementation of a scatter search procedure that computational experiments and statistical analysis show to be superior to the solution methods reported in the literature. In the process of choosing the best scatter search design, we discovered that a sequential approach is capable of producing a highly competitive design and thus validating what has been the typical process published in the SS literature.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Refs. TIN2009-07516 and) and the Government of the Community of Madrid, grant S2009/TIC-1542. We would like to thank Professor Robert R. Lewis for sharing the source code of the RCM and SA methods.

References

Agrawal, A., P. Klein and R. Ravi, "Ordering problems approximated: single-processor scheduling and interval graph completion," *Automata, Languages and Programming*, 510: 751-762, 1991.

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

Campos, V., E. Piñana and R. Martí , "Adaptive Memory Programming for Matrix Bandwidth Minimization" *Annals of Operations Research* vol. 183: 7-23, 2011.

Cuthill, E. and J. Mckee, "Reducing the bandwidth of sparse symmetric matrices," *ACM '69 Proceedings of the 1969 24th national conference*, pp. 157-172, 1969.

Das, A. and M. Roberts, "Metric distances of permutations," www.cra.org/Activities/craw_archive/dmp/awards/2004/Das/paper.ps, 2004

Duarte, A., R. Martí , "Tabu Search and GRASP for the Maximum Diversity Problem". *European Journal of Operational Research*, 178: 71-84, 2007.

Duarte, A., R. Martí, E. G. Pardo and J. J.Pantrigo, "Scatter search for the cut width minimization problem," *Annals of Operations Research*, In press, DOI: 10.1007/s10479-011-0907-2, 2010.

Duarte, A., F. Glover, R. Martí and F. Gortázar, "Hybrid scatter tabu search for unconstrained global optimization," *Annals of Operations Research*, 183:95-123, 2011.

Feo, T. A. and M. G. C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, 8(2): 67-71, 1989.

Gallego, M., A. Duarte, M. Laguna and R. Martí, "Hybrid heuristics for the maximum diversity problem," *Computational Optimization and Applications*, 44(3): 411-426, 2009.

Garey, M.R and D.S. Johnson, "*Computers and Intractability: A Guide to the Theory of NP-Completeness*", W. H. Freeman and Co., 1979.

Gibbs, N.E., "A hybrid profile reduction algorithm", *ACM Trans. Math. Softw.* 2(4):378-387, 1976.

Gibbons, A., M. Paterson, J. Torán and J. Diaz, "The minsumcut problem," *Algorithms and Data Structures*, 519:65-79, 1991.

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

Guan, G. and K.L.Williams, "Profile minimization of triangulated triangles", *Discrete Mathematics,* 260: 69-76, 2003.

Karp, R., “Mapping the genome: some combinatorial problems arising in molecular biology”, *STOC'93 Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pp. 278-285, 1993.

Kendall, R., “Incidence matrices, interval graphs and seriation in archaeology,” *Pacific Journal of Mathematics*, vol. 28, pp. 565-570, 1969.

Laguna, M and R.Martí, “*Scatter Search: methodology and implementations in C*”, Clubber Academic Publisher, 2003.

Lewis, J., “The Gibbs-Poole-Stockmeyer and Gibbs-King algorithms for reordering sparse matrices", *ACM Transactions on Mathematical Software (TOMS)*, 8: 190-194, 1982.

Lewis, R., “Simulated annealing for profile and fill reduction," *International Journal for Numerical Methods in Engineering*, 37(6): 905-925, 1994.

Lin, Y., and J.Yuan, “Profile minimization problem for matrices and graphs.”, *Acta Math. Appl. Sinica*, English-Series, Yingyong Shuxue-Xuebas, 10(1): 107-112, 1994.

Martí, R., A. Duarte and M. Laguna, “Advanced scatter search for the max-cut problem”, *INFORMS Journal on Computing*, 21(1): 26-38, 2009.

Matrix Market [Online]. http://math.nist.gov/MatrixMarket/collections/hb.html, 2011

Penrose, M., J. Petit, M. Serna and J. Diaz, “Convergence theorems for some layout measures on random lattice and random geometric graphs,” *Journal of Combinatorics, Probability and Computing*, 9: 489-511, 2000.

Petit, J., M. Serna and J. Díaz, “A survey of graph layout problems,” *ACM Computing Surveys*, 34: 313-356, 2002.

Poole, W., P. Stockmeyer and N. Gibbs “An algorithm for reducing the bandwidth and profile of a sparse matrix,”*SIAM Journal on Numerical Analysis*, 13:236-250, 1976.

Resende, M. G. C., R. Martí, M. Gallego and A. Duarte, “GRASP and path relinking for the max-min diversity problem," *Computers and Operations Research*, 37:498-508, 2010.

Resende,M. G. C.,S. H. Smith and T. A. Feo, “A greedy randomized adaptive search procedure for maximum independent set," *Operations Research*, 42: 860-878, 1994.

Saad, Y., *Iterative methods for sparse linear systems*, SIAM, ISBN-13: 978-0-898715-34-7, 2003.

Tewarson, R., *Sparse matrices*, Academic Press: New York, 1973.

Appendix

Table 7 shows the objective function value (profile) of the Harwell-Boeing instances and the associated running time of the SS method.

Instance	Value	CPU Time
494 BUS	3499	237.5
662 BUS	8962	318.5
685 BUS	8528	799.8
ASH292	2784	61.8
ASH85	490	1.4
BCSPWR01	82	0.1
BCSPWR02	113	0.3
BCSPWR03	434	3.2
BCSPWR04	1992	40.6
BCSPWR05	3354	90.3
BCSSTK01	466	0.4
BCSSTK02	2145	0.4
BCSSTK03	272	0.3
BCSSTK04	3159	4.9
BCSSTK05	2192	4.3
BCSSTK06	13437	123.9
BCSSTK07	13437	123.9
BCSSTK19	7638	855.7
BCSSTK20	3006	195.8
BCSSTK22	641	2.2
CAN 24	95	0.1
CAN 61	338	0.4
CAN 62	172	0.5
CAN 73	520	0.9
CAN 96	1080	2.1
CAN 144	969	2.3
CAN 161	2482	6.6
CAN 187	2195	9.7
CAN 229	4141	27.9
CAN 256	5049	40.8
CAN 268	5215	25.0
CAN 292	4718	41.7
CAN 445	15494	150.4
CAN 634	28493	499.9
CAN 715	24414	984.3

Instance	Value	CPU Time
DWT 59	223	0.4
DWT 66	127	0.2
DWT 72	151	0.7
DWT 87	434	1.2
DWT 162	1286	5.8
DWT 193	4388	15.3
DWT 198	1092	7.6
DWT 209	2621	25.1
DWT 221	1646	20.1
DWT 234	803	10.6
DWT 245	2053	29.7
DWT 307	6676	46.4
DWT 310	2630	25.0
DWT 346	6051	54.1
DWT 361	4635	64.8
DWT 419	6679	107.3
DWT 492	3361	151.0
DWT 503	13152	192.3
DWT 512	3975	144.6
DWT 592	9498	220.5
DWT 607	13278	419.3
DWT 758	6392	371.3
DWT 869	13107	1526.5
DWT 878	17259	1104.6
DWT 918	16502	1277.3
GR 30 30	24311	1190.4
LSHP 265	3162	25.7
LSHP 406	5964	83.2
LSHP 577	10045	222.1
LSHP 778	15719	586.2
NOS1	467	2.1
NOS2	1907	36.0
NOS3	45631	2222.8
NOS4	651	1.1
NOS5	20446	209.0
NOS6	9095	184.2
NOS7	34675	338.7
PLAT362	10620	105.8

Table 7. Best individual values on HB instances obtained with SS