GRASP for the SumCut Problem

Jesús Sánchez-Oro and Abraham Duarte

Dept. Ciencias de la Computación
Universidad Rey Juan Carlos.
E-mail: jesus.sanchezoro@urjc.es, abraham.duarte@urjc.es

Abstract

This paper proposes a GRASP algorithm to solve the SumCut minimization problem. In the SumCut problem one is given a graph with n nodes and must label the nodes in a way that each node receives a unique label from the set $\{1, 2, \dots, n\}$, in order to minimize the sum cut of the generated solution. The SumCut problem is really important in archeology (in seriation tasks) and in genetics, helping in the Human Genoma Project. This problem is equivalent to the Profile problem, but giving the reverse solution. Both problems are graph layout problems, and they are equivalent, because a solution for SumCut is reversal solution for Profile problem. This paper will solve only the SumCut problem, so in order to obtain a solution for the Profile problem it is only need to reverse the given solution.

Keywords: metaheuristic, GRASP, SumCut.

1. Introduction

Let $G = (V, E)$ be an undirected graph, where V denotes the set of vertices and E the set of edges. Let $n = |V|$ and $m = |E|$. A layout φ of the vertices of G is a one-to-one mapping from the set V onto the integers $\{1,2, \dots, n\}$ where each vertex $v \in V$ has a unique label $\varphi(v) \in \{1,2, \dots, n\}$. Given a graph G, and a vertex v placed in position $i = \varphi(v)$, we define the left set $L(i, \varphi, G)$ and the right set $R(i, \varphi, G)$ as:

$$L(i, \varphi, G) = \{u \in V: \varphi(u) \leq i\}$$
$$R(i, \varphi, G) = \{u \in V: \varphi(u) > i\}$$

Set $L(i, \varphi, G)$ contains vertices on the left side of position i, including itself. On the other hand, set $R(i, \varphi, G)$ stores vertices on the right side of position i. Taking into account the definition of these two set, the *vertex cut* at position i of φ, is computed as:

$$\delta(i, \varphi, G) = |\{u \in L(i, \varphi, G): \exists v \in R(i, \varphi, G): uv \in E\}|$$

The SumCut of a layout φ, denoted as $SC(\varphi, G)$, is calculated as the sum of vertices cut of each position. In mathematical terms:

$$SC(\varphi, G) = \sum_{i=1}^{n} \delta(i, \varphi, G)$$

The SumCut minimization problem then consists of minimizing the value of $SC(\varphi, G)$ over all possible layouts Π_n:

$$SC(G) = \min_{\varphi \in \Pi_n} SC(\varphi, G)$$

The Profile problem, and in the same way the SumCut, were proved to be NP-Complete for cobipartite graphs in [1], and they were also proved to be NP-Complete for general graphs in [2] [3] [4].

The SumCut is equivalent to the Profile minimization problem as it was stated in [5]. Specifically, the reverse solution of the SumCut corresponds to a solution of the Profile. Both optimization problems have been extensively studied. See for instance [6] [7] [8]. Practical applications of these problems appear in genetics. The goal of the Human Genome Project consist of sequencing the DNA of humans as well as other species with the target of elucidating the genetic information contained therein. In order to construct a physical map of a large DNA molecule it is necessary to extract clones from it. Then a fingerprint of each clone is obtained. Finally, DNA molecule is reassembled determining how the clones overlap among them. Each clone is a sequence of nucleotides drawn from the set {A,C,T,G}, so the reassembly process consists of permuting a linear layout of a graph.

In [9] is described an application in archeology, where it is necessary to serialize different artifacts (fossil, hardware, jewels, etc.). The serialization is known in archeology as "seriation" and consists of placing in chronological order different artifacts in the same culture using a relative dating method. Specifically, the practical application is based on the re-arrangement of a matrix, which can be translated on the reordering of a linear layout of a graph.

Reducing the profile of a matrix is relevant problem in mathematics since it leads to a reduction of the amount of space needed for some storage scheme. On the whole, it achieves an improvement of the performance of several operations such as Choleski factorization of non-singular systems of equations [10]. Recently the profile reduction has been used in new areas like information retrieval to browse hypertext [11].

The Profile problem was originally proposed as a way to reduce the storage space needed to save sparse matrix [12] but it was proved that it is equivalent to the SumCut problem [5]. An important application of the Profile problem arises in clone fingerprinting [13].

E.Cuthill and J.McKee proposed the Reverse Cuthill-McKee (RCM) algorithm [14] in order to get the minimum profile of a graph. If we want to get a solution for the SumCut problem it is only need to reverse the solution generated.

N.Gibbs et al. solve the SumCut problem using a new algorithm based on the RCM [15]. The paper describes three problems of the RCM and presents a new algorithm that solves the described problems.

J.Lewis also describes a method to re-order sparse matrices in order to reduce their profile [16] using the Gibbs-King algorithm to improve the results from the RCM algorithm.

The previous algorithms are used in Profile and SumCut problems and also in Bandwidth problem, with differences only in the last step: the numbering of the nodes.

Until now, the best heuristic proposed to obtain better solutions in the SumCut problem is presented in [17]. It uses the Simulated Annealing to reduce the profile of a matrix. The algorithm starts with a previously calculated solution and it improves the solution by using the Simulated Annealing technique. The original solution is calculated using either the RCM algorithm or the Gibbs-King algorithm, and the instances used are a subset of the Harwell-Boeing graphs set.

2. GRASP

The GRASP metaheuristic was developed in the late 1980s [18] and the acronym was coined in [19]. GRASP can be divided into two phases. The first one consists of constructing a solution and then, in the second phase, improver the incumbent solution with a local search procedure. In this section we describe several two constructive procedures and two local search methods for the SumCut problem.

2.1 Construction procedures

We have designed two constructive algorithms *C*1 and *C*2 for the SumCut problem. *C*1 implements a typical GRASP construction where each candidate element is initially evaluated by a greedy function to construct a Restricted Candidate List (*RCL*) and one element is selected at random from the *RCL*. *C*1 assigns the smaller available label to the selected vertex.

The constructive procedure C1 starts by creating the set of unlabeled vertices U (initially $U = V$) and the set of labeled vertices $L = V \backslash U$. The constructive procedure labels one vertex in each iteration. The first node u_0 is selected according to its degree. Specifically, the vertex with the minimum degree is selected (ties are broken at random). The vertex u_0 is labeled $l_0 = 1$. Then sets U and L are properly updated (i.e, $U = U \backslash \{u_0\}$ and $L = L \cup \{u_0\}$)

Once the first label l_0 is assigned to vertex u_0, *C*1 constructs the candidate list *CL* with the vertices adjacent to u_0. Then, all the vertices in the *CL* are evaluated with a greedy function g. For this problem we propose the following equation:

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

where $|N_L(v)|$ indicates the number of vertices adjacent to v that has been already labeled and $|N_U(v)|$ is the number of vertices adjacent with v that has not been labeled yet. The constructive procedure stores in g_{min} and g_{max}, respectively, the minimum and maximum value calculated with the greedy function. The next step consists of constructing the Restricted Candidate List, *RCL* with all the unselected vertices in *CL* having a value of the greedy function greater than or equal to a specified cutoff value, th. In mathematical terms:

$$RCL = \{v \in V : g(v) > th\}$$

where

$$th = g_{min} + \alpha_1 \cdot (g_{max} - g_{min})$$

and

$$g_{min} = arg \min_{u \in CL} g(u) \text{ and } g_{max} = arg \max_{u \in CL} g(u)$$

The *C*1 procedure randomly selects an element *u* from the *RCL* and assigns the next label to it. After that, it updates the CL with the adjacent vertices of *u*. This method ends when all the vertices are labeled.

We now consider the constructive algorithm *C*2, based on another strategy where randomization and greedy selection are interchanged. This strategy was introduced in [20] and recently used in [21] [22] [23]. Specifically, in *C*2 we first randomly choose candidates from the *CL* (defined as above) and then evaluate them according to the same greedy function. More formally, RCL is constructed by selecting at random $\alpha_2 * |CL|$ elements from *CL*. Then, the vertex $u \in RCL$ with the largest value of g is selected to become part of the solution under construction. *C*1 and *C*2 have two search parameters, α_1 and α_2. We study the impact of these two parameters in the Computational Results Section.

2.2 Local search procedures

A solution to the SumCut problem can be represented with a permutation. This kind of solution representation has associated two different types of moves: interchange and insertion. The first one consists of swapping the labels of two different vertices. That is, given two vertices u and v ($u, v \in V$), the operator $interchange(u, v)$ assigns the label $\varphi(u)$ to vertex v and the label $\varphi(v)$ to vertex u obtaining a new labeling φ'.

On the other hand, the insertion move consists of inserting a vertex in a different position. That is, given a vertex v and a position i, $insertion(v, i)$ assigns the label i to vertex v (i.e., after the move $\varphi(v) = i$) and all the nodes between u and v will change their labels as:

$$\varphi(v_j) = \begin{cases} \varphi(v_{j+1}) & \text{for } i < j \leq \varphi(v) \\ \varphi(v_{i-1}) & \text{for } i > j \geq \varphi(v) \end{cases}$$

The local search procedure based on interchanges, `LS_Interchange`, has two input arguments (i.e., the constructed solution, φ, and the graph, G). This procedure is illustrated in Figure 1. The algorithm performs moves while improving (see steps 2 to 13). Vertices are scanned at random. Specifically, the local search procedure selects at random a vertex v (step 4) placed in a position $\varphi(v)$. Then, the vertex u (step 5) is the one placed in position $\varphi(v) + 1$. The remaining vertices are explored in order taking into account that after exploring the vertex in position n, it is explored the vertex in position 1.

The procedure tries to perform the corresponding interchange (step 6). If this move reduces the objective function (step 7), the original solution is updated (step 9). Otherwise, the method performs moves until no further improvement is reached.

The local search procedure based on insertions is quite similar. Specifically, instruction in step 6 is substituted by $\varphi' = interchange(u, \varphi(v))$.

```
begin LS_Interchange(φ, G)
   1    improve = true;
   2    while (improve) do
   3       improve = false;
   4       for each v ∈ V do
   5            for each u ∈ V do
   6                 φ' = interchange(u, v)
   7                 if (SC(φ', G) < SC(φ, G) then
   8                      improve = true;
   9                      φ = φ'
   10                end
   11           end
   12      end
   13   end
end
```

Figure 1. Pseudo-code for Local Search

The SumCut and Profile are problems in which a simple movement in a solution generated may change the objective function value, because as the solution is based on the sum of the cut of all vertices, each vertex will contribute to the objective function. For that reason the selection of a vertex that will improve the global objective function value is a challenge for solution methods based on heuristic optimization.

3. COMPUTATIONAL RESULTS

This section describes the computational experiments performed to test the efficiency of the GRASP heuristic. We implemented the methods in Java and the experiments where carried out on an Intel Core 2 Duo E4800 computer running at 3.00 Ghz with 3 Gb of RAM.

We used one set of test problems in our experiments. A total of 22 instances were considered. The set of instances are derived from the Harwell-Boeing Sparse Matrix Collection [24]. This collection consists of a set of standard test matrices arising from problems in lineal systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large matrices arising in applications. Graphs are derived from these matrices as follows. Let M_{ij} denote the element of the i-th row and j-th column of the $n \times n$ sparse matrix M. The corresponding graph has n vertices. Edge (i, j) exists in the graph if and only if $M_{ij} \neq 0$.

The computational experiments are divided into two parts. In the first one, we evaluate the performance of the proposed strategies. We use a subset of the set of instances of 10 representative examples (i.e., different sizes and densities). Then, one time we have identify the best combination of search strategies we compare it with the state-of-the-art.

In the first experiment we compare our 2 proposed constructive methods in order to determine the best constructive procedure. Both procedures construct 100 different solutions, recovering the best one of all constructions. Parameters α_1 and α_2 have been set randomly, in order to boost the diversity of the generated solutions. Table 1 shows the comparison between our constructive methods ($C1$ and $C2$), and the Cuthill-McKee algorithm (RCM), described in [14], which is one of the most used algorithms in the SumCut problem, and the Gibbs-King algorithm (GK), described in [16] [15]. We report for each method the average percentage deviation with respect to the best know solution (Dev.), the number of times that each method matches the best known solution (#Best) and the CPU time (Time) in seconds. Regarding RCM and GK we directly use the values published by the authors.

	Dev	**#Best**	**Time**
C1	25.64%	1	0.33
C2	13.32%	2	0.36
RCM	19.73%	2	*
GK	15.08%	1	*

Table 1. Comparison of constructive methods

Table 1 clearly shows that the best constructive procedure is C2 (13.32%) followed by GK (15.08%) method. It is important to realize that that the CPU time of GK and RCM were not published in the corresponding papers.

In our second experiment we study how the local search procedures interact with the constructive methods. Specifically, we compare constructive C1 coupled with local search based on Interchanges, C1+LS1, C1 with local search based on insertions, C1+LS2 and the same for the C2 procedure (i.e., C2+LS1 and C2+LS2). In order to prevent really long running time, all the algorithms are stopped after a maximum time of 1000 seconds. Table 2 shows the performance of the four considered methods considering the same statistics than above.

	Dev	**#Best**	**Time**
C1+LS1	13.21%	2	228.52
C1+LS2	13.42%	1	291.87
C2+LS1	8.05%	1	205.97
C2+LS2	8.46%	2	264.33

Table 2. Comparison between local search methods

Table 2 shows that the best algorithm is C2+LS1 in terms of the objective function and CPU time. However, C2+LS2 obtains a larger number of best solutions.

In our last experiment we compare our best GRASP procedures (i.e., GRASP1(C2+LS1) and GRASP2(C2+LS2)) with the best method identified in related literature, using all the 22 instances. Specifically, they are compared with the Simulated Annealing due to [17]. As we said before, Table 2 shows that the best algorithm is C2+LS1 in terms of the objective function and CPU time, but we have choose to include C2+LS2 because it finds more best solutions than the C2+LS1.

	Dev	#Best	Time
GRASP1	8.02%	2	283.78
GRASP2	8.63%	4	357.36
SA	9.36%	15	307.95

Table 3. Comparison between Local Search and Simulated Annealing

We can see in Table 3 that although Simulated Annealing (SA) achieve better solutions more times, both C2+LS1 and C2+LS2 have a lower average percentage deviation. It is remarkable to say that GRASP1 also achieves a better CPU time than the other ones.

4. CONCLUSIONS

In this paper we present two different constructive methods for the SumCut problem as well as two local searches, deriving into four possible GRASP procedures. Experimental results show that our two best GRASP methods outperform in terms of average percentage deviation the previous Simulated Annealing, using shorter CPU time. On the other hand, Simulated Annealing finds more times the best solution but much of merit seems to be related to GK constructive method.

The main objective of future works is the factorization of the objective function so we can evaluate a solution faster, which means that we will be able to do more movements in the same time, which seems to end in a better solution. We are also trying to develop new local searches in order to move only the more suitable nodes based in the characteristics of the solution.

References

[1] Y.Lin, Y.Liu, S.Wang J.Yuan, "NP-completeness of the profile problem and the fill-in problem on cobipartite graphs," *J.Mathem.Study*, 1998.

[2] A.Gibbons, M.Paterson, J.Torán J.Diaz, "The Minsumcut problem," *Algorithms and Data Structures*, 1991.

[3] P.Golovach, "The total vertex separation number of a graph," *Diskretnaya Matematika*, 1997.

[4] J.Yuan Y.Lin, "Profile minimization problem for matrices and graphs," *Acta Mathematicae Applicatae Sinica*, 1994.

[5] A.Agrawal, P.Klein R.Ravi, "Ordering problems approximated: single-processor scheduling and interval graph completition," *Automata, Languages and Programming*, 1991.

[6] M.Penrose, J.Petit, M.Serna J.Diaz, "Convergence theorems for some layout measures on

random lattice and random geometric graphs," *Journal Combinatorics, Probability and Computing*, 2000.

[8] J.Petit, M.Serna J.Díaz, "A Survey of Graph Layout Problems," *ACM Computing Surveys*, 2002.

[7] J.Petit, M.Serna, L.Trevisan J.Díaz, "Approximating Layout Problems on Random Sparse Graphs," Universidad Politécnica de Valencia, Tech. Report 2001.

[9] R.Kendall, "Incidence Matrices, Interval Graphs and Seriation in archaeology," *Pacific Journal of Mathematics*, 1969.

[10] Y.Saad, "Iterative Methods for Sparse Linear Systems," *PWS Publishing Company*, 1996.

[11] R.A.Botafogo, "Cluster analysis for hypertext systems," *Proceedings of the 16th Annul International ACM-SIGIR Conference on Research and Development in Information Retrieval*, 1993.

[12] R.Tewarson, "Sparse Matrices," *Academic Press, New York*, 1973.

[13] R.Karp, "Mapping the Genome: Some Combinatorial Problems Arising in Molecular Biology," *STOC'93 Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993.

[14] E.Cuthill, J.Mckee, "Reducing the bandwidth of sparse symmetric matrices," *ACM '69 Proceedings of the 1969 24th national conference*, 1969.

[15] W.Poole,P.Stockmeyer N.Gibbs, "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix," *SIAM Journal on Numerical Analysis*, 1976.

[16] J.Lewis, "The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices," *ACM Transactions on Mathematical Software (TOMS)*, 1982.

[18] M.G.C. Resende, T.A.Feo, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, 1989.

[17] R.Lewis, "Simmulated Annealing for Profile and Fill Reduction," University of British Columbia, Tech. Report 1993.

[19] M.G.C.Resende, S.H.Smith T.A.Feo, "A greedy randomized adaptive search procedure for maximum independent set," *Operations Research*, 1994.

[20] R.F.Werneck, M.G.C.Resende, "A hybrid heuristic for the p-median problem," *Journal of heuristics*, 2004.

[21] Renato F. Werneck, Mauricio G. C. Resende, "A Hybrid Heuristic for the p-Median Problem," *Journal of Heuristics*, 2004.

[22] A. Duarte, R. Martí, E.G.Pardo J.J.Pantrigo, "Scatter Search for the cutwidth problem," *Annals of Operations Research*, 2010.

[23] M.G.C. Resende, R. Martí, M. Gallego, A. Duarte, "GRASP and Path Relinking for the Max-Min Diversity Problem," *Computers and Operations Research*, 2010.

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