A Relative Neighbourhood GRASP for the SONET Ring Assignment Problem

Lucas de Oliveira Bastos [1], Luiz Satoru Ochi [1], Elder M. Macambira [2]
[1] Instituto de Computação, Universidade Federal Fluminense
Address: Rua Passo da Pátria, 156, 24210-240, Niterói – RJ – Brasil
Email: {lbastos, satoru}@ic.uff.br
[2] Departamento de Estatística, Universidade Federal da Paraíba
Address: Cidade Universitária, 58051-900, João Pessoa – PB – Brasil
Email: elder@de.ufpb.br

January 2005

Abstract

We consider the SONET ring assignment problem, a combinatorial optimization problem that arises in telecommunications networks design. In this problem, each customer has to be assigned to exactly one SONET ring and a special ring interconnects the others rings together. A capacity constraint on each ring is also imposed. The problem is to find a feasible assignment of the customers minimizing the total number of rings used. We describe a greedy randomized adaptive search procedure (GRASP) for the SONET ring assignment problem. The procedure is based on a relative neighbourhood structure and the use of a variable objective function. We show extensive empirical evidences to the effectiveness of our algorithm in practice. These results indicate that the proposed GRASP implementation compares favorably to greedy methods and implementations of tabu search, scatter search and path relinking previously proposed for the problem. For most benchmarks instances in the literature, we obtained solutions that are either optimal or very close to it.

keywords SONET Ring Assignment Problem, Graph Partitioning, Metaheuristics, GRASP.

1 Introduction

Consider the following node-partitioning problem. We are given an undirected graph $G = (V, E)$ and a positive integer B. Associated to each edge (u, v) in E there is a non negative integer d_{uv}. A feasible solution of the problem is a partition of the vertices in G into disjoint sets $V_1, V_2, \ldots, V_k$ such that:

$$(i) \sum_{(u,v)\in G[V_j]} d_{uv} \le B \text{, for all } j \in \{1,\ldots,k\}\text{, and} \quad (1)$$

$$(ii) \sum_{j=1}^{k} \sum_{(u,v)\in \delta(V_j)\,|\,u<v} d_{uv} \le B\,, \quad (2)$$

where $G[V_j]$ is the graph induced by V_j in G and $\delta(V_j)$ is the set of edges with exactly one extremity in V_j. The goal is to find a feasible solution that minimizes the size of the partition, i.e., the value of k.

This graph partitioning problem is also known as the SONET (or SDH) ring assignment problem, or SRAP for short. The SRAP was investigated recently by Goldschmidt *et al.* in [3]. In their paper the authors motivated the study of the problem by showing its relevance for the design of fiber-optics telecommunication networks using the SONET (Synchronous Optical NETwork) or SDH (Synchronous Digital Hierarchy) technology.

In this context, the vertices of graph $G = (V,E)$, with $|V| = n$, are associated to client sites while the edges are associated to the existence of traffic demand between pairs of clients. The edge weights measure the actual demand of communication among clients. Given a feasible solution to the graph partitioning problem stated above, the vertices in each subset of the partition form a *local ring* or simply a *ring*. Due to physical limitations on the equipments used in building the network, the total demand in each ring must not exceed a constant B. Besides, the total demand between clients in different clusters is handled by an additional ring called the *federal ring*. The connection of a local ring to the federal ring is made possible through a device known as a digital cross-connect (DCS). Since the capacity of the federal ring is also bounded by B, the sum of the weights of the edges in the multicut corresponding any feasible solution cannot be larger than that amount. Finally, because the DCS's are by far the most expensive equipments needed to implement a network, a basic problem in the design of low-cost SONET networks asks for a solution that minimizes the number of local rings. One can easily check that the graph partitioning problem discussed at the beginning of this section correctly models the latter problem.

We refer to Goldschmidt *et al.* [3] for a more detailed discussion on the architecture of SONET networks and a review of the literature on the SRAP. In their work the authors proved that SRAP is NP-hard. They also proposed three different greedy heuristics to solve it. Another work on heuristics for SRAP can be found in [1] where tabu search, path relinking and scatter search metaheuristics are proposed. Recently, Macambira [4] proposed many integer programming formulations and exact methods to solve SRAP.

This paper focus on the solution of SRAP via metaheuristics. We present a GRASP for this problem, which makes use a relative neighbourhood structure. In Section 2, we describe the construction and local search phases, and in Section 3, we present computational results for benchmark instances of the problem. Concluding remarks are made in Section 4.

2 GRASP heuristic

GRASP [2] is a multi-start procedure, where different points in the search space are probed with local search for high-quality solutions. Each iteration of GRASP consists of the construction of a randomized greedy solution, followed by a local search, starting from the constructed solution. The best solution from all iterations is returned as result.

In the remainder of this section, we describe in detail the phases of the GRASP for the SRAP, i.e., the GRASP construction and local search phases. To describe the construction phase, one needs to provide a candidate definition (for the restricted candidate list) and an adaptive greedy function, and specify the candidate restriction mechanism. For the local search phase, one must define the neighbourhood and specify a local search algorithm.

2.1 Construction phase

We first notice that it is possible for GRASP proposed to return infeasible solutions. Of course this must be allowed since, sometimes, we cannot decide in reasonable computing time if the instance has a solution or not. Even when the construction phase discovers that the instance is feasible, infeasible solutions can be visited.

The goal here is to allow more flexibility to move along the search space. However, the objective function has been modified to turn feasible solutions more attractive than infeasible ones. To explain the changes in the objective function, we define the following value associated to a feasible solution S with vertex set $V_1, \ldots, V_r$:

$$BN = \max\left(0, \sum_{j=1}^{r} \sum_{(u,v)\in\delta(V_j)\,|\,u<v} d_{uv}\right), \tag{3}$$

where BN indicate the total traffic through the federal ring. Now, the cost of this solution in the modified objective function is computed by the formula:

$$z = rB \text{ (feasible solution) or,} \tag{4}$$

$$z = 2r + BN \text{ (unfeasible solution).} \tag{5}$$

We now turn our attention to the construction phase of GRASP. The algorithm is an adaptation of the three greedy heuristics presented in [3] to include randomization. It always starts with a solution containing n rings with only one vertex assigned to it. Those rings are feasible since otherwise the instance itself is infeasible. Therefore, the only constraint that can be violated is the one that imposes a limit on the demand on the federal ring.

At each iteration, the union of two rings into a single one is considered. Such an union is accepted only if the resulting ring is feasible. Clearly, this operation reduces the amount of demand on the federal ring. The greedy choice is guided by the demands on edges which are in the multicutset of the rings. The randomization will not force us to pick the best edge (edge-based heuristic), best cut (cut-based heuristic) or the best node (node-based heuristic). Instead, the selected edge, cut or node is chosen randomly among the best candidates in the restricted candidate list with smaller demand.

We developed a new algorithm for the construction phase trying to improve the results generated by greedy heuristic randomized below. This new proposal is described as follows.

The aim of the Relative Neighbourhood Heuristic (RNH), introduced here, is to establish sets of neighbour localities according to a certain criteria to maintain in same rings. The criteria used here establishes that two localities are considered relative neighbours if they directly share the bigger demand among all their common demands. After this step, all neighbour localities are gathered together in a ring. This procedure generates at least a ring and at most $(n/2)$ rings, being n the number of localities of the problem. Remaining locations are then placed in a ring with probability given by $p = f * dem(i, a)$ where f is a random factor and $dem(i, a)$ is the total traffic between locality i and the ring a. After this, RNH tries to merge rings the same way cut-based heuristic does. We refer to Goldschmidt *et al.* [3] for more details on cut-based heuristic.

The steps of the construction algorithm are summarized in Figure 1.

2.2 Local search phase

After a solution is constructed, a local search phase should be executed for attempting to improve the initial solution. The local search phase is based on two types of neighbourhoods.

The first neighbourhood considers the possibility of moving every vertex u from its ring to any of the existing rings different. The incumbent solution is initialized with the solution obtained by the construction phase. Each vertex u is moved from the ring s to the ring t. We observe that moving u from s to t modifies the traffic values and a new solution $z(.)$ is computed. If for each triple (u,s,t), the value $z(.)$ has been improved, then they are interchanged, and a new incumbent solution is created.

The second neighbourhood considers all possibilities of exchanging pairs of vertices among different rings. If no exchange improved the value $z(.)$, then the local search is terminated. Otherwise, after the elements that provides minimum $z(.)$ are interchanged, a new incumbent solution is created and the local search is performed again. The neighbourhoods are inspected in that order as it is suggested in the algorithm in Figure 2.

3 Computational results

In this section, we illustrate the use of GRASP on randomly generated test problems. All tests were run on a PC desktop equipped with a 1700 MHz Pentium IV processor and 256 Mbytes of RAM under Linux operating system. The GRASP code is written in C ++ language.

Before we describe the experimental results, we must comment about instances used. The set of instances is divided into two categories. Instances in category C1 are taken from [3] and correspond to instances type 1 and 2 in that paper. Category C2 is composed of all instances tested in [1] excluding those already in C1.

```
procedure RNH( )
//-- Generating neighbourhood information:
for each edge (i, j) do
  if weight(i, j) >= max(largerEdge(i), largerEdge(j))
    setAsNeighbors(i, j);
//-- Generating Rings from neighbourhood information:
for each pair (i, j) of localities do
  if areNeighbors(i, j)
    mergeInARing(i, j);
//-- Placing remaining locations:
for each i = nextDesalocatedLocality( ) do
  placeInARing(i);
//-- Merging rings:
for each pair (i, j) of rings do
  if canMerge(i, j)
    mergeRings(i, j);
end procedure
```

Figure 1. Pseudo-code for Relative Neighbourhood Heuristic.

In total, there are 111 instances in C1 and 230 in C2. Within each category, instances are divided into geometric and random ones and, then, further divided into those having low and high ring capacities, respectively, 155 Mbs and 622 Mbs. These characteristics of an instance can be deduced from its name. Instance names always start with two letters. The first letter is either "G" or "R" meaning that the instance belongs to the geometric or the random subdivision, respectively. The second letter is either "L" or "H", depending if the ring capacity is 155 Mbs or 622 Mbs, respectively. For more details on how those instances were generated and their properties, we refer the interested reader to the original papers where they were introduced.

```
procedure LS( )
//-- Applying First Neighbourhood:
do {
  for each node belonging to the lesser ring do
    sendItToTheBestRingThatCanHoldIt( );
} until couldNotMoveANode;
//-- Applying Second Neighbourhood:
for each ring i do
  for each ring j do
    for each node n belonging to i do {
      tryToInterchange(n, nodesOf( j));
      if solutionIsImprooved( )
        restartLocalSearch( );
    }
end procedure
```

Figure 2. Pseudo-code of the local search procedure.

The tests were processed using 1000 iterations for each instance and each one was executed 10 times with different seeds. Harder instances were executed until find optimal solution or up to 1 million iterations. Macambira [4] implemented an exact algorithm that provided optimal solutions for the two categories C1 and C2.

In Tables 1 and 2, we show the results obtained for first category of instances C1 and C2, respectively. In these tables, the first column shows instance type according to its characteristics as explained above. The second column tells the demand constraint for each ring. Third column brings the number of feasible instances in each category and type. Fourth and fifth columns show the absolute and percent number of feasible solutions found by GRASP. The next two columns show these numbers for optimal solutions found. Finally, last column shows average computational time in seconds.

In Table 1, we can see that the proposed GRASP found optimal solutions for all instances. The average execution times for GRASP were less than 0.08 second.

Table 1: Computational results for C1 class instances.

Instances			**Feasible solutions**		**Optimal solutions**		
Type	**B (Mbs)**	**# FS**	**# FS - G**	**# FS - G (%)**	**# FS - G**	**# FS - G (%)**	**Avg. time (s)**
GL	155	23	23	100.0	23	100.0	0.047
GH	622	27	27	100.0	27	100.0	0.054
RL	155	28	28	100.0	28	100.0	0.072
RH	622	33	33	100.0	33	100.0	0.046

In Table 2, we show the results obtained for second category of instances C2. We make the following observations about this category: GRASP found optimal solutions in 225 of the 230 test problems and the average execution times for GRASP were less than 2.6 seconds.

Table 2. Computational results for C2 class instances.

Instances			**Feasible solutions**		**Optimal solutions**		
Type	**B (Mbs)**	**# FS**	**# FS - G**	**# FS - G (%)**	**# FS - G**	**# FS - G (%)**	**Avg. time (s)**
GS	155	70	70	100.0	70	100.0	0.371
GL	622	70	70	100.0	67	95.7	0.636
RS	155	70	69	98.6	68	97.1	1.425
RL	622	20	20	100.0	20	100.0	2.519

Table 3. Test problems for which GRASP did not find optimal solutions.

Instances	z_{exact}	z_{GRASP}	**gap (%)**	**It. to best**	**Time to best**	**Total time**
new.GH_50_3.4	5	6	16.67	811	1.216	1.500
new.GH_50_9.4	5	6	16.67	595	0.536	0.921
new.GH_50_9.7	5	6	16.67	407	0.687	1.688
new.RL_15_2.7	3	4	25.00	1	0.001	0.140

Table 3 lists all test problems for which GRASP did not find optimal solutions. Besides iterations and CPU time to find the best solution, the table lists the total time to run the 1000 iterations, as well as the optimality gap ($100[z_{GRASP} - z_{exact} / z_{GRASP}$) in the GRASP solution. It should be noted that the optimality gap is small, i.e., notwithstanding the GRASP found solutions within 25% and 16.67% of the optimal solution this percents correspond to only one ring far from the optimal solution each. Furthermore, these solutions were obtained at most 1.7 seconds in average.

4 Concluding remarks

In this paper, we have studied the SONET ring assignment problem. A heuristic method for solving this problem was proposed. The method, a GRASP, was described in details. The quality of the heuristic solutions can be measured with a lower bound produced by exact method.

Computational results obtained for benchmark problems illustrated that the GRASP heuristic is quite efficient to find optimal solutions and good solutions for SRAP. So, the GRASP heuristic is competitive with the best heuristic in the literature in terms of solution quality. On these problems that were not possible to find optimal solutions, the average deviations in percent of the solution obtained by GRASP was less than 25%. For all instances, the best solution was found in less than 0.7 second in average.

References

[1] R. Aringhieri and M. Dell'Amico. Solutions for the SONET ring assignment with capacity constraints, in Proc. of the 4th Metaheuristic International Conference, Porto, Portugal, 2001.

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

[3] O. Goldshmidt, A. Laugier and E. V. Olinick, SONET/SDH ring assignment problem with capacity constraits, Discrete Applied Mathematics, 129:99-128, 2003.

[4] E. M. Macambira, Modelos e algoritmos de programação inteira no projeto de redes de telecomunicações, Tese de doutorado, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, maio, 2003.