Designing low-cost access network topologies

Héctor Cancela [1]
cancela@fing.edu.uy

Franco Robledo [1,2]
frobledo@fing.edu.uy

Gerardo Rubino[2]
Gerardo.Rubino@irisa.fr

[1] Universidad de la República, J. Herrera y Reissig 565, Montevideo, Uruguay

[2] IRISA/INRIA, Campus de Beaulieu, Rennes 35042 CEDEX, France

Abstract

One of the problems in the design of a Wide Area Network consists in laying out the access network, that is, choosing the concentrator sites and the communication lines which will connect the terminal sites to the switches in the backbone network.

We model the problem of finding a minimum cost access network as a variant of the Steiner Problem in Graphs, which can be approximately solved by heuristics such as the Greedy Randomized Adaptive Search Procedure (GRASP), which comprises a construction phase and a local search phase. We consider two different alternatives for the construction phase. One method works iteratively selecting from a candidate list of the terminal nodes nearest to the current partial solution; the other chooses from among a list of k shortest paths from a single terminal node. Both methods are experimentally compared over 206 problem instances of different topological characteristics, generated using the problem classes in the SteinLib repository, and with known lower bounds for their optimal values.

Although both methods obtained good results, the method based on finding nearest nodes is less computationally expensive and gives lower cost (average 2% better) solutions over all problem classes; a significative amount in network design problems.

Keywords: metaheuristic, topological design, GRASP, RNN.

1 Introduction

A wide area network (WAN) can be seen as a set of sites and a set of communication lines that interconnect the sites. A typical WAN is organized as a hierarchical structure integrating two levels: the *backbone network* and the *access network* composed of a certain number of *local access subnetworks* [7]. Nodes in the local access subnetworks are either *terminals* or *concentrators*; nodes in the backbone are *switches*. Each local access subnetwork usually has a tree-like structure, rooted at a single site switch of the backbone network, and connects users (terminal sites) either directly to this switch or to a hierarchy of intermediate concentrator sites which are connected to the switch. The backbone network has usually a meshed topology, and its purpose is to allow efficient and reliable communication between its sites acting as connection points for the local access subnetworks.

Assume the backbone network fixed. Let S_C be the set of sites where concentrator equipment can be installed in order to diminish the cost of the access network and S_T the set of terminal sites (the

clients). Considering the network of feasible connections on the WAN as a cost-weighted, undirected graph, the Access Network Design Problem (ANDP) consists of finding a subgraph of minimum cost (costs are positive real numbers) such that $\forall s_t \in S_T$ there exists a path from s_t to the backbone network. To simplify the presentation and the algorithms, we collapse the backbone into a single fixed node z. We introduce the notation used to formalize the problem:

- $S = S_T \cup S_C \cup \{z\}$ is the set of all nodes,
- $C = \{c_{ij}\}_{i,j \in S}$ is the matrix which gives for any pair of sites of S, the cost of laying a line between them. When the direct connection between i and j is not possible, we take $c_{ij} = \infty$.
- $E = \{(i,j); \forall i,j \in S$ such that $c_{ij} < \infty\}$ is the set of feasible connections between sites of S.
- $G = (S, E)$ is the graph of feasible connections on the Access Network.

We define the Access Network Design Problem ANDP$(G(S,E), C)$ as the problem of finding a subgraph $\mathcal{T} \subset G$ of minimum cost such that $\forall s_t \in S_T$ there exists a unique path from s_t to node z and such that terminal sites can not be used as intermediate nodes (they must have degree 1 in the solution). This problem belongs to the NP-Hard class. Some references in this area and related problems are [7, 8].

In a previous paper [1], we have introduced a polynomial time heuristic based on the GRASP methodology and using a random neural network model in the GRASP local search phase for approximately solving the ANDP; this method is briefly presented in Section 2. As the performance of this method depends critically on the construction phase of the GRASP method, we present in Section 3 an alternative algorithm for this phase. In Section 4, the new algorithm is experimentally compared with the previous one, showing computational results obtained on a set of 206 problem instances, including topologies with hundreds of nodes. This section also discusses conclusions and future work.

2 GRASP description

GRASP is a well known metaheuristic, which has been applied for solving many hard combinatorial optimization problems with very good results [5, 6]. A GRASP is an iterative process, where each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result. Figure 1 illustrates a generic GRASP implementation pseudo-code. The GRASP takes as input parameters the candidate list size, the maximum number of GRASP iterations and the seed for the random number generator. After reading the instance data (line 1), the GRASP iterations are carried out in lines 2-6. Each GRASP iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (lines 5 and 6).

```
Procedure GRASP(ListSize,MaxIter,RandomSeed);

1  InputInstance();
2  for k = 1 to MaxIter do
3    InitialSolution = ConstructGreedyRandomizedSolution(ListSize, RandomSize);
4    LocalSearchSolution = LocalSearch(InitialSolution);
5    if cost(LocalSearchSolution) < cost(BestSolutionFound) then
6      UpdateSolution(BestSolutionFound, LocalSearchSolution);
7  end_for;
8  return BestSolutionFound;
```

Figure 1: GRASP pseudo-code.

In the construction phase, a feasible solution is built, one element at a time (the definition of what is an element is problem dependent; for example, in the ANDP, it could be a node, a link, or even a whole path in the network). At each step of the construction phase, a candidate list is determined by ordering all non already selected elements with respect to a greedy function that measures the (myopic)

benefit of including them in the solution. The heuristic is adaptive because the benefits associated with every element are updated at each step to reflect the changes brought on by the selection of the previous elements. Then one element is randomly chosen from the best candidates list and added into the solution. This is the probabilistic component of GRASP, which allows for different solutions to be obtained at each GRASP iteration, but does not necessarily jeopardize the power of the adaptive greedy component.

The solutions generated by the construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution from its neighborhood. The local search algorithm depends on the suitable choice of a neighborhood structure, on the availability of an efficient neighborhood search techniques, and the starting solution. Through the use of customized data structures and a careful implementation, an efficient construction phase that produces good initial solutions for the local search can be created. For further detail of this metaheuristic the reader may consult the references [5, 6].

In a previous work [1] we have presented a customization of the GRASP procedure for solving the ANDP. The construction phase was based on a customized and randomized version of the Takahashi-Matsuyama algorithm [10]; the elements which are added at each step of this method are the k shortest paths connecting one terminal node to the existing partial solution. The local search phase used a random neural network model (RNN, introduced by Gelenbe [2, 3, 4]) which helps to determine in which order to consider additional concentrator sites; this method helps to explore a reduced neighborhood of most promising solutions, improving the computing times of the method. The combined results of the two phases were very good. Nevertheless, an improvement in the construction phase could lead to still better results.

In the next section, we present an alternative construction method, which is compared experimentally with the previously presented one.

3 Construction phase based on nearest nodes

The new construction phase is based on the idea of finding the nearest nodes to the current partial solution and randomly selecting one of them. This node is connected by means of a shortest path to the solution, and the process is iterated, until all terminal nodes are included.

The algorithm (called ANDP_Construction_Phase_NearestNodes, and shown in Figure 2), uses an auxiliary structure $\mathcal{P}$ in which are stored the shortest paths from nodes of $S_T \cup \{z\}$ to nodes in S without using terminal nodes as intermediate nodes; this matrix is preprocessed before running the GRASP algorithm, by applying suitably the Dijkstra algorithm on the induced subgraph $G(\{i,j\} \cup S_C \cup \{z\})$, $\forall i,j \in S_T \cup \{z\}$.

```
Procedure ANDP_Construction_Phase_NearestNodes;
Input: G = (S, E), C, k, P;

1  v ← Select_Random(S_T ∪ {z});
2  T_sol ← {v}; Y ← {v};
3  while Y \ (S_T ∪ {z}) ≠ ∅ do
4     L_p ← the shortest paths from the k nearest sites of (S_T ∪ {z}) \ Y to T_sol using P;
5     p ← Select_Random(L_p);
6     u ← the endpoint of p non-belonging to T_sol;
7     T_sol ← T_sol ∪ {p}; Y ← Y ∪ {u};
8  end_while;
9  return T_sol;
end ANDP_Construction_Phase_NearestNodes;
```

Figure 2: Pseudocode for construction phase based on k nearest nodes.

The algorithm has the following inputs: G, the network of feasible connections; C, the link connection

costs; k, the candidate list size for the GRASP method; and $\mathcal{P}$, containing the shortest paths without terminals as intermediate nodes. In the pseudocode shown in Figure 2, lines 1-2 select randomly and uniformly a node from $S_T \cup \{z\}$ to initialize the constructed solution. The solution in construction is denoted by $\mathcal{T}_{sol}$ and the auxiliary set $Y \subseteq (S_T \cup \{z\})$ denotes the nodes already included in $\mathcal{T}_{sol}$. Let us notice that all the nodes of $S_T \cup \{z\}$ necessarily must be in the solution. Iteratively the construction phase adds new nodes of $S_T \cup \{z\}$ to the current solution $\mathcal{T}_{sol}$. Each iteration works in the following way. In line 4 the algorithm searchs for the k nodes of $(S_T \cup \{z\}) \setminus Y$ which are nearest to the current solution $\mathcal{T}_{sol}$; the corresponding shortest paths are extracted from $\mathcal{P}$ and stored in a restricted candidate list $\mathcal{L}_p$. A path p is randomly and uniformly selected from $\mathcal{L}_p$ in line 6. Let u be the endpoint of p such that $u \notin \mathcal{T}_{sol}$. In line 7, we add p to the current solution $\mathcal{T}_{sol}$ and the set Y is updated by adding u to it. This process is repeated until all the nodes of $S_T \cup \{z\}$ have been added to $\mathcal{T}_{sol}$. The built feasible solution $\mathcal{T}_{sol}$ is returned in line 9. Note that z has the same "role" that a terminal node until it is added to $\mathcal{T}_{sol}$.

In Figure 3 we can see the previous construction phase (which was introduced in [1]). That method basically builds a feasible solution based on choosing randomly a single not yet considered terminal node, computing the k shortest paths from this node to the current solution (using any standard k shortest path algorithm), and selecting randomly one of them to be added to the solution in construction, finalizing this process once all the terminal nodes have been added.

```
Procedure ANDP_Construction_Phase_ShortestPaths;
Input: G = (S, E), C, k;

1  ∀s_t ∈ S_T a unique identifier n_t is assigned;
2  T_sol ← {z}; Y ← ∅;
3  while (Y \ S_T) ≠ ∅ do
4     s̄_t ← ArgMax{n_t | s_t ∈ (S_T \ Y)};
5     L_p ← the k shortest valid paths from s̄_t to T_sol;
6     p ← Select_Random(L_p);
7     T_sol ← T_sol ∪ {p}; Y ← Y ∪ {s̄_t};
8  end_while;
9  return T_sol;
end ANDP_Construction_Phase_ShortestPaths;
```

Figure 3: Pseudocode for construction method based on k shortest paths.

4 Performance Tests and Discussion

We present here some experimental results obtained with the GRASP-RNN algorithm. The algorithm was implemented in ANSI C. The experiments were done on a Pentium IV computer with 1.7 GHz, and 1 Gbytes of RAM, running under Windows XP. All instances were solved with identical GRASP parameter settings. The candidate list size was $ListSize = 10$, and the maximum number of iterations $MaxIter = 100$. These values were chosen from the GRASP reference literature.

We used a large test set, by modifying the Steiner Problem instances from the classes C, MC, X, PUC, I080, I160, I320, I640, P6E, P6Z, WRP3, and WRP4 in the SteinLib library [9]. This library contains many problem classes of widely different graph topologies. For each problem, we selected the terminal node of the original problem with greatest degree as the z node; the Steiner nodes as concentrator sites, and the terminal nodes as terminal sites. Also, all the edges between terminal sites were deleted (as they are not allowed in feasible ANDP solutions). If the resulting topology was unconnected, the problem instance was discarded. By this process, we generated 206 ANDP instances. Notice that, since in the ANDP the terminals cannot be used as intermediate nodes the cost of an SPG optimum is a lower bound for the optimal value of the corresponding ANDP.

Table 1 shows a summary of computational results. The first column contains the names of the original SteinLib classes and the entries from left to right are: the number of customized instances (NI), the size of

the selected instances in terms of number of nodes and edges respectively, the number of instances where the lower bound was obtained reaching therefore the optimum (NOPT), the average of the improvement of the results of the local search phase over the construction phase (Avg. LSI), the average running time per iteration (to which we added the preprocessing time of the matrix $\mathcal{P}$), and the average of the gap of the GRASP solution respect to the lower bound (Avg. LB_GAP).

The average improvement is computed as Avg. LSI= $\sum_{p \in Set} \text{LSI}(p)/\text{NI}$, where for problem p, $\text{LSI}(p) = 100 \times [(\sum_{i=1}^{MaxIter} (CC_i - LC_i)/CC_i)]/MaxIter$, CC_i and LC_i being the costs of the solutions delivered in iteration i by the Construction Phase and the Local Search Phase respectively. The average gap is Avg. LB_GAP= $\sum_{p \in Set} \text{LB\_GAP}(p)/\text{NI}$ (where for problem p, $\text{LB\_GAP}(p) = 100 \times (Best\_Cost\_Found - Lower\_Bound)/Lower\_Bound$).

We also include a last line with the summary results corresponding to the whole test set (the average values follow the same formulae as before, but now computed over the 206 instances).

Testset	**NI**	**Nodes**	**Edges**	**NOPT**	**Avg. LSI**	**Avg. secs/itr**	**Avg. LB_GAP**
C	6	500	625-2500	-	15.17%	10.12	0.27%
MC	3	97-150	4656-11175	1	19.33%	4.43	3.69%
X	2	52-58	1326-1653	-	8.12%	0.52	32.27%
PUC	4	64-128	192-750	2	17.32%	1.02	0.11%
I080	70	80	120-3160	17	13.21%	0.57	6.73%
I160	22	160	240-2544	7	17.14%	3.09	3.08%
I320	15	320	480-10208	3	16.12%	9.03	2.12%
I640	15	640	960-4135	2	17.69%	26.96	2.56%
P6E	10	100-200	180-370	2	16.11%	1.54	14.02%
P6Z	5	100-200	180-370	1	16.42%	1.03	19.12%
WRP3	25	84-886	149-1800	8	15.41%	15.07	0.00019%
WRP4	29	110-844	188-1691	5	17.75%	19.17	0.00101%
Total	206	-	-	48	15.46%	8.15	4.47%

Table 1: Computational results using ANDP_Construction_Phase_NearestNodes.

The results show that the algorithm finds in most cases good quality solutions. In 48 instances (out of 206) we reached the lower bound and therefore optimality. As in general only lower bounds and not true optima are known, it is natural that a gap persists in many cases; as shown in the table, with wide variations depending on the problem class. Even then, in most cases this gap is quite small (less than 5% gap average in 8 over 12 problem classes).

Another point of interest is that the RNN model in the local search phase was used with the aim of capturing global connectivity information about the graph and to determine the order in which the concentrator nodes non-present in the solution delivered by the construction phase are chosen to improve the solution delivered by the greedy construction phase. We observe that for all problem classed, the local search phase improved significantly the solutions built by the construction phase; over 15% average improvement for most problem classes (and always over 8% average improvement).

Testset	**NOPT_1**	**Avg_1. LSI**	**Avg_1. secs/itr**	**Avg_1. LB_GAP**	**DOPT**	**NB**	**Avg. DCOST**
C	-	19.95%	12.13	0.41%	-	2	0.09%
MC	1	23.34%	3.12	6.64%	0	1	2.03%
X	-	11.00%	0.73	39.56%	-	1	5.23%
PUC	2	21.04%	1.27	0.14%	0	1	0.02%
I080	13	18.22%	1.49	10.71%	4	8	2.97%
I160	7	23.82%	4.03	3.86%	0	2	0.57%
I320	2	21.12%	10.14	2.89%	1	3	0.61%
I640	2	20.59%	29.63	4.67%	0	2	1.93%
P6E	2	23.75%	1.83	16.49%	0	2	2.11%
P6Z	1	22.01%	1.10	23.22%	0	1	3.07%
WRP3	7	22.14%	19.54	0.00253%	1	4	0.00113%
WRP4	3	27.94%	26.45	0.00405%	2	5	0.00234%
Total	40	21.51%	10.47	6.53%	8	32	1.51%

Table 2: Comparison between both heuristics.

On the other hand, in Table 2 we show a comparison of these results with the ones obtained when applying the heuristic introduced in [1] over the same test-set. Let us denote by $\mathcal{H}_1$ and $\mathcal{H}$ the GRASP heuristics introduced in [1] and in this paper respectively. We note that for all the problem classes the average values obtained by $\mathcal{H}$ were better than those obtained by $\mathcal{H}_1$. In Table 2 the first column contains the names of the original SteinLib classes; the other entries, from left to right, are: the same measures defined above and shown in Table 1 but for the heuristic $\mathcal{H}_1$ (they are differentiated by the index 1); the number of instances where the lower bound was attained by $\mathcal{H}$ but not by $\mathcal{H}_1$ (DOPT); the number of instances where $\mathcal{H}$ was better than $\mathcal{H}_1$ (NB); and the average of the difference between the solution cost obtained by $\mathcal{H}$ respect to the one obtained by $\mathcal{H}_1$ (Avg. DCOST=$100 \times [\sum_{p \in Set}(CH_1(p) - CH(p))/CH_1(p)]$/NI, where for problem p, $CH_1(p)$ and $CH(p)$ are the costs of the best solutions obtained by $\mathcal{H}_1$ and $\mathcal{H}$ respectively).

The comparison shows that the heuristic $\mathcal{H}$ improved the solution built by $\mathcal{H}_1$ in 32 ANDP instances, achieving in addition the lower bound (i.e. the optimum) in 8 of these. In average,$\mathcal{H}$ improves over 1.5% over $\mathcal{H}_1$; and in 5 over 12 problem classes, the average improvement of $\mathcal{H}$ respect to $\mathcal{H}_1$ is greater than 2%. These are good results considering that when designing a WAN access network, a cost decrease of few percentage points results in high economic interest due to the investment levels associated.

An additional point of interest is to observe the average improvement of the local search phase over the results of the construction phase (the columns marked "Avg. LSI"). Here we observe that the local search phase improves in average 21% over the results of the construction phase in method $\mathcal{H}_1$, and in average 15% in $\mathcal{H}$; however, the average gap respect to the lower bound is smaller in $\mathcal{H}$ than in $\mathcal{H}_1$. The reason of this is that in general the solution built by ANDP_Construction_Phase_NearestNodes has a better quality (i.e. with smaller cost) than the one built by ANDP_Construction_Phase_ShortestPaths providing thus a better starting solution for the local search phase.

We also observe that the weighted average running time per iteration is smaller in $\mathcal{H}$ than in $\mathcal{H}_1$; this is an additional criterion for selecting the new method as a component of the GRASP procedure.

References

[1] H. Cancela, F. Robledo, and G. Rubino, "A GRASP algorithm witn RNN based local search for designing a WAN access network", Proceeding of the Latin American Conference on Combinatorics, Graphs and Applications (LACGA), Santiago, Chile, August 16-20, 2004. To appear in Electronic Notes on Discrete Mathematics.

[2] E. Gelenbe, "Stability of the random neural network model", Neural Computation, vol. 5, no. 2, pp. 239-247 (1990).

[3] E. Gelenbe and F. Batty, "Minimum cost graph covering with the random neural network", in Computer Science and Operations Research. New York: Pergamon, pp. 139-147 (1992).

[4] E. Gelenbe, V. Koubi, and F. Pekergin, "Dynamical random neural network approach to the traveling salesman problem", in Proc. Symp. Syst., Man., Cybern., pp. 630-635 (1993).

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

[6] S.L. Martins, M.G.C. Resende, C.C. Ribeiro and P.M. Pardalos, "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", Journal of Global Optimization, vol. 17, pp. 267-283 (2000).

[7] M. Priem and F. Priem, "Ingénierie des WAN", ISBN 2-10-004510-5, Dunod InterEditions (1999).

[8] C. D. Randazzo, H. P. L. Luna and P. Mahey, "Benders decomposition for local access network design with two technologies", Discrete Math.& Theoretical Comp. Science, vol. 4 no. 2, pp. 235-246 (2001).

[9] http://elib.zib.de/steinlib/testset.php (last access: April 28, 2004).

[10] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs", Math. Jpn., 24:537-577 (1980).