Sonet Network Design Problems

Marie Pelleau

Brown University
Computer Science
115 Waterman Street
Providence, RI 02912
`marie.pelleau@etu.univ-nantes.fr`

Abstract. This report presents two problems related to the design of optical telecommunication networks, namely the Synchronous Optical Network Ring Assignment Problem and the Intra-ring Synchronous Optical Network Design Problem. Theses network topology problems can be represented as a graph partitioning with capacity constraints. Previous works have described a mathematical model for these problems based on the graph partitioning. We have used this model to compare and implement, in Comet , several objective functions and metaheuristics.

1 Introduction

This report presents the work I have done during my internship at Brown University, as part of my Master 2 ORO. My advisors were Charlotte Truchet at Université de Nantes and Pascal Van Hentenryck at Brown University. This work dealt with real-world combinatorial optimization problems from the field of network design. Two particular problems have been considered, both consist in finding optical networks configurations in a particular network topology. The objective is to minimize the cost of the configuration, while respecting some constraints expressing that the customers demand is satisfied. These problems have been shown $\mathcal{NP}$-hard and have already been solved by combinatorial optimization techniques.

The goal of this internship was to re-implement the existing methods in Comet , an up-to-date programming language dedicated at constraint programming and combinatorial optimization, in order to reproduce the previous results. In addition, we have proposed several refinements to these algorithms. With these new components, we obtained similar results on the first problems and we managed to improve the results on the second one.

This report is organized as follows. In the sequel of this section we introduce the two problems we have worked on, and the local search techniques which have been used to solve them. Then, in section 2, we introduce the models in a constrained optimization format for the two problems. We then present the previous works on SRAP and IDP in section 3. Section 4 describes the key ingredients necessary to implement the local search algorithms. Finally, the results are shown in Section 5.

1.1 Optical networks topologies

During the last few years the number of internet based application users has exponentially increased, and so has the demand for bandwidth. To enable fast transmission of large quantities of data, the fiber optic technology in telecommunication is the current solution.

The Synchronous Optical NETwork (SONET) in North America and Synchronous Digital Hierarchy (SDH) in Europe and Japan are the standard designs for fiber optics networks. They have a ring-based topology, in other words, they are a collection of rings.

Rings Each customer is connected to one or more rings, and can send, receive and relay messages using an add-drop-multiplexer (ADM). There are two bidirectional links connecting each customer to his neighboring customers on the ring. In a bidirectional ring the traffic between two nodes can be sent clockwise or counterclockwise. This topology allows an enhanced survivability of the network, specifically if a failure occurs on a link, the traffic originally transmitted on this link will be sent on the surviving part of the ring. The volume traffic on any ring is limited by the link capacity, called B. The cost of this kind of network is defined by the cost of the different components used in it.

There are different ways to represent a network. In this report, we consider two network topologies described by R. Aringhieri and M. Dell'Amico in 2005 in [2]. In both topologies the goal is to minimize the cost of the network while guaranteeing that the customers' demands, in term of bandwidth, are satisfied.

First topology In the first topology, each customer is connected to exactly one ring. All of these *local rings* are connected with a device called digital cross connector (DXC) to a special ring, called the *federal ring*. The traffic between two rings is transmitted over this special ring. Like the other rings, the federal ring is limited by the capacity B. Because DXCs are so much more expensive than ADMs we want to have the smallest possible number of them. As there is a one-to-one relationship between the ring and the DXC, minimizing the number of rings is equivalent to minimizing the number of DXCs. The problem associated to this topology is called *SONET Ring Assignment Problem* (SRAP) *with capacity constraint*. Figure 1 shows an example of this topology.

Second topology In the second topology, customers can be connected to more than one ring. If two customers want to communicate, they have to be connected to the same ring. In this case, the DXC are no longer needed and neither is the federal ring. However there are more ADM used than in the first topology. In this case, the most expensive component is the ADM although its price has significantly dropped over the past few years. It is important, in this topology, to have the smallest numbers of ADMs. This problem is called *Intra-ring Synchronous Optical Network Design Problem* (IDP). The figure 2 illustrates this topology.

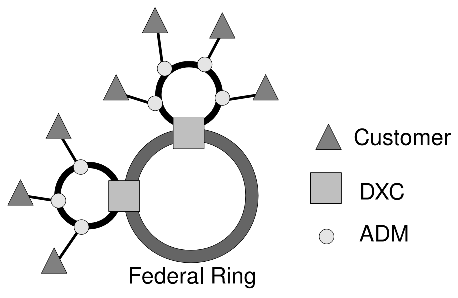

Fig. 1. A SONET network with DXC.

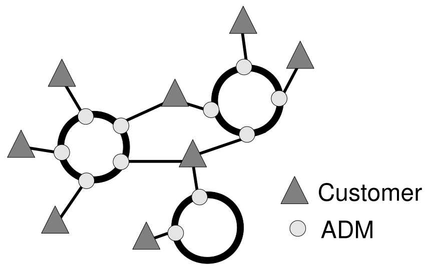

Fig. 2. A SONET network without DXC.

Both of these problems are $\mathcal{NP}$-hard (see O. Goldschmidt, A. Laugier and E. Olinick in 2003, [6], and O. Goldschmidt, D. Hochbaum, A. Levin and E. Olinick in 2003, [7] for details). On such problems, the field of combinatorial optimization provide efficient solving tools.

1.2 Brief introduction of Local Search

Real-world combinatorial optimization problems are extremely computationally challenging. No single approach is likely to be effective on all problems, or even on all instances of a single problem. Designing efficient methods to solve such problems is a very active research area.

Principles Local search is a metaheuristic based on iterative improvement of an objective function. It has been proved very efficient on many combinatorial optimization problems. It can be used on problems which formulated either as mere optimization problems, or as constrained optimization problems where the goal is to optimize an objective function while respecting some constraints. Local search algorithms performs local moves in the space of candidate solutions, called the search space, trying to improve the objective function, until a solution deemed optimal is found or a time bound is reached. Defining the neighborhood graph and the method to explore it are two of the key ingredients of local search algorithms.

The approach for solving combinatorial optimization problems with local search is very different from the systematic tree search of constraint and integer programming. Local search belongs to the family of metaheuristic algorithms, which are incomplete by nature. Hence, it cannot prove optimality. However on many problems, it will isolate a optimal or high-quality solution in a very short time. Local search sacrifices optimality guarantees to performance.

Basic algorithm A local search algorithm starts from a candidate solution and then iteratively moves to a neighboring solution. This is only possible if a neighborhood relation is defined on the search space. Typically, for every candidate solution, we define a subset of the search space to be the neighborhood. Moves are performed from neighbors to neighbors, hence the name local search. The basic principle is then to choose among the neighbors the one with the best value for the objective function. The problem is then that the algorithm will be stuck in local optima. Metaheuristics, such as Tabu Search, are added to avoid this. In Tabu Search, the last t visited configurations are left out of the search (t being a parameter of the algorithm) : this ensures that the algorithm can escape local optima, at least at order t.

The Algortihm 1 illustrates the Tabu Search in pseudo-code.

Termination of local search can be based on a time bound. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of iterations. Local search algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by

```
Choose or construct an initial solution $S0$
$S \leftarrow S_0$
$S^* \leftarrow S_0$
$bestValue \leftarrow objValue(S0)$
$T \leftarrow \emptyset$
while Terminaison criterion not satisfied do
    Choose $S'$ in the neighborhood of $S$ which minimize the objective
    $S \leftarrow S'$
    if $objValue(S) < bestValue$ then
        $S^* \leftarrow S$
        $bestValue \leftarrow objValue(S)$
    end
    Record tabu for the current move in T (delete oldest entry if necessary)
end
```

Algorithm 1: Tabu Search

the algorithm is not optimal. This can happen even if termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithms.

Local search is particularly appropriate for large-scale problems involving thousands of decision variables. It is also efficient in many optimization applications where systematic search techniques are less effective.

COMET Comet is an object-oriented language created by Pascal Van Hentenryck and Laurent Michel. It has a constraint-based architecture that makes it easy to use when implementing local search algorithms, and more important, constraint-based local search algorithms (see [1] for details).

Moreover, it has a rich modeling language, including invariants, and a rich constraint language featuring numerical, logical and combinatorial constraints. Constraints and objective functions are differentiable objects maintaining the properties used to direct the graph exploration. The constraints maintain their violations and the objectives their evaluation. One of its most important particularity, is that differentiable objects can be queried to determine incrementally the impact of local moves on their properties.

2 Models

The models for these problems are based on graphs. Given an undirected graph $G = (V, E)$, $V = \{1, \ldots, n\}$, the set of nodes represent the customers and E, the set of edges, stand for the customers' traffic demands. A communication between two customers u and v correspond to the weighted edge (u, v) in the graph, where the weight d_{uv} is the fixed traffic demand. Note that $d_{uv} = d_{vu}$, and that $d_{uu} = 0$.

2.1 SRAP

The SRAP problem consists in assigning each customer to a ring. This is modeled by a decomposition of the set of nodes V into a partition, each subset of the partition representing a particular ring. Assigning a node to a subset of the partition in the model is then equivalent to assigning a customer to a ring.

Let $V_1, V_2, \ldots, V_k$ be a partitioning of V in k subsets. Each subset of the partition corresponds to a ring, in other words, each customer in the subset V_i is assigned to the i-th local ring. As each customer is connected with an ADM to one and only one ring, and each local ring is connected to the federal ring with a DXC, there are exactly $|V|$ AMD and k DXC used in the corresponding SRAP network.

Hence, minimizing the number of rings is equivalent to minimizing k subject to the following constraints :

$$\sum_{u \in V_i} \sum_{v \in V, v \neq u} d_{uv} \leq B, \qquad \forall i = 1, \ldots, k \tag{1}$$

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{u \in V_i} \sum_{v \in V_j} d_{uv} \leq B \tag{2}$$

Constraint (1) imposes that the total traffic routed on each ring does not exceed the capacity B. In other words, for a given ring i, it forces the total traffic demands of all the customers connected to this ring, to be lower or equal to the bandwidth. Constraint (2) forces the load of federal ring to be less than or equal to B. To do so, it computes the sum of the traffic demands between all the pairs of customers connected to different rings.

Figure 3 illustrates the relation between the node partitioning model and the first topology SRAP. We can see that, because the nodes 1, 3, 5 and 6 are in the same partition, they are connected to the same ring. Similarly, the nodes 2, 4 and 7 are on the same ring.

For this problem we can easily compute a lower bound k_{lb}. In fact, we want to know the minimum number of partitions needed to route all the traffic. It seems logical to sum all the traffic demands of the graph and divided it by the bandwidth B. Moreover, we cannot have fractional part of partition, that is why we take the upper round of this fraction.

$$k_{lb} = \left\lceil \frac{\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} d_{uv}}{B} \right\rceil$$

2.2 IDP

Contrarily to the SRAP problem, there is no need to assign each customer to a particular ring because customers can be connected to several rings. Here the

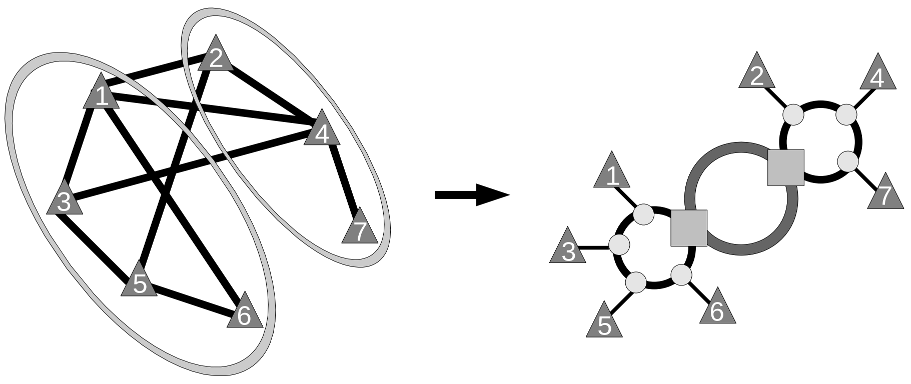

Fig. 3. Relation between the node partitioning and the network topology.

model is based on a partition of the edges of the graph, where a subset of the partition corresponds to a ring.

Let $E_1, E_2, \ldots, E_k$ be a partitioning of E in k subsets and $Nodes(E_i)$ be the set of terminal nodes of the edges in E_i. Each subset of the partition corresponds to a ring, in other words, each customer in $Nodes(E_i)$ is linked to the i-th ring. In the corresponding IDP network, there are $\sum_{i=1}^{k} |Nodes(E_i)|$ ADM and no DXC.

Hence, minimizing the number of ADMs is equivalent to minimizing $\sum_{i=1}^{k} |Nodes(E_i)|$ subject to,

$$\sum_{(u,v)\in E_i} d_{uv} \leq B, \qquad \forall i = 1, \ldots, k \tag{3}$$

Constraint (3) imposes that the traffic in each ring does not exceed the capacity B.

Figure 4 shows the relation between the edge partitioning and the second topology. If all the edges of a node are in the same partition, this node will only be connected to a ring. We can see, for example, the node 4 has all its edges in the same partition, because of that, the node 4 is connected to only one ring. On the opposite, the edges of the node 2 are in two different partitions, so it is logically, connected to two rings.

The SRAP problem can be seen as a node partitioning problem, whereas IDP, as an edge partitioning problem for the graph described above, subject to capacity constraints. These graph partitioning problems have been introduced in [6] and [7].

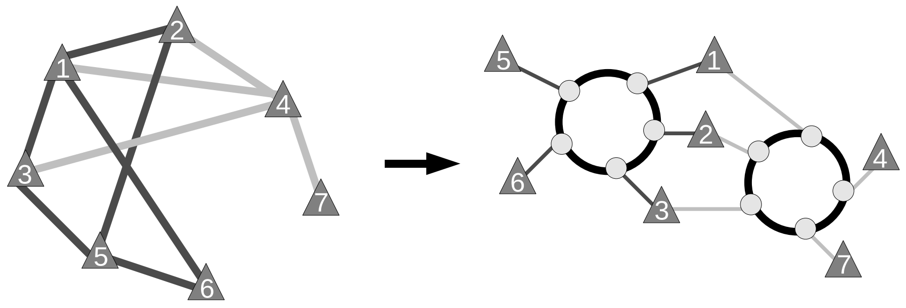

Fig. 4. Relation between the edge partitioning and the network topology.

3 Related work

These two problems have been well studied. It has been proven that they are both $\mathcal{NP}$-hard ([6], [7]).

3.1 Greedy algorithms for SRAP

In [6] the SRAP problem is considered. They propose three greedy algorithms with different heuristics, the *edge-based*, the *cut-based* and the *node-based*. The first two algorithms start by assigning each node to a different ring. At each iteration they reduce the number of rings by merging two rings V_i and V_j if $V_i \cup V_j$ is a feasible ring for the capacity constraint. In the edge-based heuristic, the two rings with the maximum weight edge are merged. While in the cut-based heuristic, the two rings with the maximum total weight of the edges with one endpoint in each of them, are merged. Algorithm 2 and 3 show the pseudo code for the edge-based and cut-based heuristics.

Given a value k, the node-based heuristic, starts by randomly assigning a node to each of the k rings. At each iteration it first chooses the ring V_i with the largest unused capacity, then the unassigned node u with the largest traffic with the nodes in V_i. Finally it adds u to the ring V_i disregarding the capacity constraint. The pseudo-code for this heuristic is shown on algorithm 4.
The node-based heuristic is run ten times. At each run, if a feasible solution is found, the corresponding value for k is kept and the next run takes $k - 1$ as an input. The idea behind this is to try and improve the objective at each run.

To test these heuristics, the authors have randomly generated 160 instances[1]. The edge-based, and the cut-based are run first. If they have found a feasible solution and obtained a value for k, the node-based is then run with as input

[1] These instances are available at `www.seas.smu.edu/~olinick/srap/GRAPHS.tar`.

```
$F \leftarrow E$
$\forall v \in N\ ring(v) \leftarrow v$
while $F \neq \emptyset$ do
    Choose a maximum capacity edge $(u, v) \in F$
    $i \leftarrow ring(u), j \leftarrow ring(v)$
    if $V_i \cup V_j$ is a feasible ring then
        $\forall v \in V_j\ ring(v) \leftarrow i$
        $F \leftarrow F \backslash \{(x, y) | ring(x) = i, ring(y) = j\}$
    else
        $F \leftarrow F \backslash \{(u, v)\}$
    end
end
```

Algorithm 2: Edge-Based Heuristic

```
$L \leftarrow \emptyset$
$\forall v \in N\ ring(v) \leftarrow v, L \leftarrow L \cup \{ring(v)\}$
repeat
    $max \leftarrow , M \leftarrow \emptyset$
    for $\{s, t | s \neq t\} \in L$ do
        if $V_s \cup V_t$ is a feasible ring then
            $E_{st} \leftarrow \{\{(u, v)\} | ring(u) = s, ring(v) = t\}$
            $cut \leftarrow \sum_{(u,v) \in E_{st}} d_{uv}$
            if $cut > max$ then
                $max \leftarrow cut, M \leftarrow \{st\}$
            end
        end
    end
    if $M \neq \emptyset$ then
        Merge $M$
    end
until No ring can be merged ($M = \emptyset$) ;
```

Algorithm 3: Cut-Based Heuristic

```
$U \leftarrow V$
for $i = 1$ to $k$ do
    Choose $u \in U, V_i \leftarrow u, U \leftarrow U \backslash \{u\}$
end
while $U \neq \emptyset$ do
    Choose a minimum capacity ring $V_i$
    Choose $u \in U$ to maximize $\sum_{\{v \in V_i\}} d_{uv}$
    $ring(u) \leftarrow V_i, U \leftarrow U \backslash \{u\}$
end
```

Algorithm 4: Node-Based Heuristic

the smallest value obtained for k. If they have not, the node-based heuristic has for input a random value from the range $[k_{lb}, |V|]$ where k_{lb} is the *lower bound*, described previously.

3.2 MIP and Branch and Cut for IDP

A special case of the IDP problem where all the edges have the same weight, is studied in [7]. This special case is called the *K-Edge-Partitioning* problem. Given a simple undirected graph $G = (V, E)$ and a value $k < |E|$, we want to find a partitioning of E, $\{E_1, E_2, \dots E_l\}$ such that $\forall i \in \{1, \dots, l\}, |E_i| \leq k$. The authors present two linear-time-approximation algorithms with fixed performance guarantee.

Y. Lee, H. Sherali, J. Han and S. Kim in 2000 ([8]), have studied the IDP problem with an additional constraint such that for each ring i, $|Nodes(E_i)| \leq R$. The authors present a mixed-integer programming model for the problem, and develop a branch-and-cut algorithm. They also introduce a heuristic to generate an initial feasible solution, and another one to improve the initial solution. To initialize a ring, the heuristic first, adds the node u with the maximum graph degree, with respect to unassigned edges, and then adds to the partition the edge $[u, v]$ such that the graph degree of v is maximum. It iteratively increases the partition by choosing a node such that the total traffic does not exceed the limit B. A set of 40 instances is generated to test these heuristics and the branch-and-cut.

3.3 Local Search for SRAP and IDP

More recently, in [2], these two problems have been studied. The authors have developed different metaheuristic algorithms, all based on the Tabu Search. The metaheuristics are the *Basic Tabu Search* (BTS), two versions of the *Path Relinking* (PR1, PR2), the *eXploring Tabu Search* (XTS), the *Scatter Search* (SS), and the *Diversification by Multiple Neighborhoods* (DMN). These local search algorithms are detailed in section 4.4.

Previously, we saw that with local search it is necessary to define a neighborhood to choose the next solution. The one they used, is the same for all of their metaheuristics. It tries to assign an item x from a partition, P_1, to another partition, P_2. The authors also consider the neighborhood obtained by swapping two items, x and y, from two different partitions, P_1 and P_2. But instead of trying all the pairs of items, it will only tries to swap the two items if the resulting solution of the assignment of x to the partition P_2 is unfeasible.

In order to compute a starting solution for the IDP problem, the authors describe four different heuristics. The first heuristic introduced in [2] ordered the edges by decreasing weight, at each iteration it tries to assign the edge with the biggest weight which is not already assigned, to the ring with the smallest residual capacity regarding to capacity constraint. If no assignment is possible, the current edge is assigned to a new ring. The second one, sorts the edges by increasing weight, and tries to assign the current edge to the current ring if the capacity constraint is respected, otherwise the ring is no longer considered and a new ring is initialized with the current edge.

The two other methods described in [2] are based on the idea that to save ADMs a good solutions should have very dense rings. They are both constructive greedy. In the third one, *Clique-BF*, it iteratively selects a clique of unassigned edges with the total traffic less or equal to B. Then assigns it to the ring that minimizes the residual capacity and, if possible, preserves the feasibility. If both of them are impossible it places it to a new ring. Algorithm 5 shows the pseudo code associated to this heuristic. The last algorithm, *Cycle-BF*, is like the previous method, but instead of looking for a clique at each iteration it try to find a cycle with as many cords as possible.

```
$U \leftarrow E$
$r \leftarrow 0$
**while** $U \neq \emptyset$ **do**
    Heuristicaly find a clique $C \subset U$ such that $weight(C) \leq B$
    $j \leftarrow min\{B - weight(E_i) - weight(C) : i \in$
    $\{1, \ldots, k\}, B - weight(E_i) - weight(C) \geq 0\}$ **if** $j = null$ **then**
        $r++$
        $j \leftarrow r$
    **end**
    $E_j \leftarrow E_j \cup C$
    $U \leftarrow U \backslash C$
**end**
```

Algorithm 5: Clique-BF

They also introduce four objective functions, one of which depends on the current and the next status of the search. Let z_0 be the basic objective function counting the number of rings of a solution for SRAP, and the total number of ADMs for IDP, and let BN be the highest load of a ring in the current solution.

$$
\begin{aligned}
z_1 &= z_0 + max\{0, BN - B\}, \\
z_2 &= z_1 + \begin{cases} \alpha \cdot RingLoad(r) & \text{if the last move has created a new ring } r, \\ 0 & \text{otherwise} \end{cases} \\
z_3 &= z_0 \cdot B + BN \\
z_4 &= \begin{cases} z_{4a} = z_0 \cdot B + BN (= z_3) & \text{(a) : from feasible to feasible} \\ z_{4b} = (z_0 + 1)BN & \text{(b) : from feasible to unfeasible} \\ z_{4c} = z_0 B & \text{(c) : from unfeasible to feasible} \\ z_{4d} = \beta z_0 BN & \text{(d) : from unfeasible to unfeasible} \end{cases}
\end{aligned}
$$

with $\alpha \geq 1$ and $\beta \geq 2$, two fixed parameters, and where $RingLoad(r)$ is the load of the ring r.

The first function z_1 minimizes the basic function z_0. As $BN > B$, it also penalizes the unfeasible solutions. In addition to the penalty for the unfeasible solutions, z_2 penalizes the moves that increase the number of rings. Function z_3 encourages solutions with small z_0, while among all the solutions with the same value of z_0, it prefers the ones in which the rings have the same loads. The last objective function z_4 is an adapting technique that modifies the evaluation according to the status of the search. It is a *variable objective function* having different expressions for different transitions from the current status to the next one.

4 Our work

In this section we present the different tools needed to implement the Constraints Based Local Search algorithms for SRAP and IDP. First we introduce the starting solution, then the neighborhoods and the objectives functions. Finally we present the different local search algorithms.

4.1 Starting solution

The starting solution, for the SRAP, can be generated with one of the three heuristics presented in [6], or like in [2] where each node is assigned to a different ring. And for the IDP, it can be generated with the heuristic in [8] or with one of the four heuristics described in [2].

The starting solution can be a random solution, like in most of the generic local search. It corresponds, for these problems, to a solution where all the items, nodes for SRAP or edges for IDP, are randomly assigned to a partition. On the opposite, the starting solution can have all the items assigned to the same partition.

Another way to do it is to compute the lower bound k_{lb} (described in section 3) and randomly assign all the items to exactly k_{lb} partitions. The local search will then reduce the violations if the starting solution is not feasible.

We have implemented and tested all these starting solutions. Based on our experiments, the best starting solution for both of the problems is the one where all the items are assigned to the same partition. Note that this solution is certainly unfeasible as all the traffic is on only one ring.

4.2 Neighborhoods

The two neighborhoods used are basic. Given a solution for a partitioning problem, they can be obtained either by moving an item form a partition to another (including a new one) or by swapping two items assigned to two different partitions. For both of these moves, the capacity constraints is ignored, and among all the possible moves, the one minimizing the most the objective function is chosen. Like in [2], we allow to construct unfeasible solutions. We have discussed and tested different variants of the first neighborhood.

Based on the fact that for the SRAP problem we try to minimize the number of rings, the first variant tries to reduce the number of partition by emptying the partition with the lowest load. More precisely, it randomly picks a customer in the m partitions with the lowest load, with m a fixed parameter. We did not try this neighborhood because of the obvious problem that it will decrease the number of rings until it reaches the value 1.

To avoid this problem we decide to pick an item from the most violated partition and assign it to another one with no violations or to a new one. Unfortunately this neighborhood did not work, because it only considers the partitions and in the case of the SRAP problem, the federal ring is not a partition. In other words, it decreases the violations on all the local rings but not the ones on the federal ring.

So in addition to the previous neighborhood, if none of the partitions are violated, the new one considers all the items. This one works well, but takes a long time to compute.

We also designed a neighborhood dedicated to the IDP problem. In this case we try to minimize the number of ADMs, in other words, we want to minimize the number of rings on which a single customer is connected. In this variant, we pick the customer which is connected to the higher number of rings. Then among all the edges for which this customer is one of the endpoints, we choose the one minimizing the objective function. This neighborhood did not work well because we reason on the nodes while the partitioning is on the edges.

After all these attempts we decided to stop trying to be smart and just choose to assign the item to the partition that minimize the most the objective function. In the end, this method appeared to work better.

4.3 Objective function

We have compared the four objectives functions describe in [2] (see Section 3) to a new one we have defined : z_5.

$$z_5 = z_0 + \sum_{p\ \in\ partitions} violations(p)$$

where

partitions are all the rings (in the case of the SRAP problem the *federal ring* is also included),

$$violations(p) = \begin{cases} capacity(p) - B & \text{if the load of } p \text{ exceed } B \\ 0 & \text{otherwise.} \end{cases}$$

This objective function minimizes the basic function z_0 and penalizes the unfeasible solutions. As z_0 is much smaller than the load of a ring, a feasible solution with 4 rings will be preferred to an unfeasible solution with 3 rings.

4.4 Local Search

Beside of the Local Search algorithms introduced in [2], we have proposed a different Diversification by Multiple Neighborhoods (DMN2). In this section we briefly recall the algorithms described in [2], then we present our variant.

The Tabu Search introduced by F. Glover and M. Laguna in 1997 in [4], like all the local search moves from? one solution to another one in its neighborhood. To avoid being stuck in a local optimum, the last t moves are marked tabu and left out of the search, hence the name of Tabu search.

The Path Relinking PR1 has been introduced by Glover in 1997 in [5]. It keeps in memory, the best solution found called *elite solution*. After a certain number of non improving iteration, the local search tries to find a path in the search space between the current solution and the elite. In order to reduce the distance, d, between the current and the elite solution, it will applies M moves ; where M is a certain percentage of the distance d.

In the second version of the Path Relinking, PR2, instead of keeping just one solution, it keeps a set of elite solutions of length ES. Like in the first version, it tries to find a path between the current solution and each of the ES elite solutions. It generates ES new solutions and continues the search with the best of the ES new solutions.

The eXploring Tabu Search introduced by M. Dell'Amico and M. Trubian in 1998 ([3]), jumps in the solution space based on long term memory information. At each iteration, it applies the best move and keeps in memory the second best one. After a fixed number of non improving iterations, the search goes back in the previous configuration associate to the best second move and applies it.

In the Scatter Search algorithm (see [5]), a small population of solutions, called *Reference Set*, evolves through combination of its solutions. At each iteration, it combines different subsets of the Reference Set to construct a new solution. More precisely, it assigns, in the new solution, the item i to the partition p, if the score of the pair (i, r) is the biggest among all the solutions in the subset considered. Note that for the Scatter Search approach, the objective functions based on the concept of 'move' can not be used. So for this algorithm, we only consider the functions z_1, z_5 and a special version of z_4 :

$$z_4 = \begin{cases} z_{4a} = z_0 \cdot B + BN & \text{if the solution is feasible} \\ z_{4d} = \beta z_0 BN & \text{otherwise} \end{cases}$$

The Diversification by Multiple Neighborhoods metaheuristic proposed in [2], has some similarity with the Variable Neighborhood Search. After a series of consecutive non improving iterations, the search empties a partition by moving all its items to another partition, disregarding the capacity constraint and locally minimizing the objective function.

The metaheuristic we have proposed, called DM2, differs from DMN on the diversification method. When the search needs to be diversified, it randomly chooses among three diversification methods (d_1, d_2, d_3). The first method, d_1, is the diversification used in DMN. The second one, d_2, randomly assigns all the items to a ring. Finally, d_3 randomly chooses a number m in the range [1, k], where k is the number of rings, and applies m moves.

Our general algorithm starts with a solution where all the items are in the same partition. Then applies one of the local search algorithms described before. If the solution returned by the local search is feasible but with the objective value greater than the lower bound k_{lb}, it empties one partition by randomly assign all its items to another. Then run once again the local search until it founds a solution with the objective value equals to k_{lb} or until the time limit is exceeded.

5 Results

The objectives functions and the metaheuristics, respectively described in Section 4.3 and Section 4.4, have been coded in COMET and tested on Intel based, dual-core, dual processor, Dell Poweredge 1855 blade server, running under Linux. The instances used are from the literature.

5.1 Benchmark

To test the algorithms, we used two sets of instances. The first one has been introduced in [6]. They have generated 80 *geometric* instances, based on the fact that customers tend to communicate more with their close neighbors, and 80 *random* instances. These subsets have both 40 *low-demand* instances, with a ring capacity $B = 155$ Mbs, and 40 *high-demand* instances, where $B = 622$ Mbs. The traffic demand between two customers, u and v, is determined by a discrete uniform random variable corresponding to the the number of T1 lines required for the anticipated volume of traffic between u and v. A T1 line has an approximately capacity of 1.5 Mbs. The number of T1 lines is randomly pick in the interval $[3, 7]$, for low-demand cases, while it is selected from the range $[11, 17]$, for the high-demand cases. The graphs generated have $|V| \in \{15, 25, 30, 50\}$. In the 160 instances, generated by O. Goldschmidt, A. Laugier and E. Olinick in 2003, 42 have been proven to be un feasible by R. Aringhieri and M. Dell'Amico using CPLEX 8.0 (see [2]).

The second set of instances has been presented in [8]. They have generated 40 instances with a ring capacity $B = 48$ T1 lines and the number of T1 lines required for the traffic between two customers has been choose in the interval $[1, 30]$. The graphs considered have $|V| \in \{15, 20, 25\}$ and $|E| = \{30, 35\}$. Most of ([2]), the instances in this set are unfeasible.

Note that all the instances can be feasible for the IDP problem, we always could assign each demand to a different partition.

5.2 Computational Results

We now describe the results obtained for SRAP and IDP on the above two benchmark sets, by the algorithms Basic Tabu Search (BTS), Path Relinking (PR1, PR2), eXploring Tabu Search (XTS), Scatter Search (SS), Diversification by Multiple Neighborhoods (DMN, DMN2) (see Section 4.4 for details). For each algorithm we consider the five objective functions of Section 4.3, but for the SS we use the three functions described in Section 4.4.

We gave a time limit of 5 minutes to each run of an algorithm, even if the average time to find the best solution is less than 1 minute. Obviously, the algorithm terminates if the current best solution found is equal to the lower bound k_{lb}.

Figures 5 and 6 show the number of optimal or high-quality solutions found by the algorithms using the five objective functions, for the IDP and the SRAP problems. Remind that objective functions z_2 and z_3 cannot be applied with the Scatter Search.

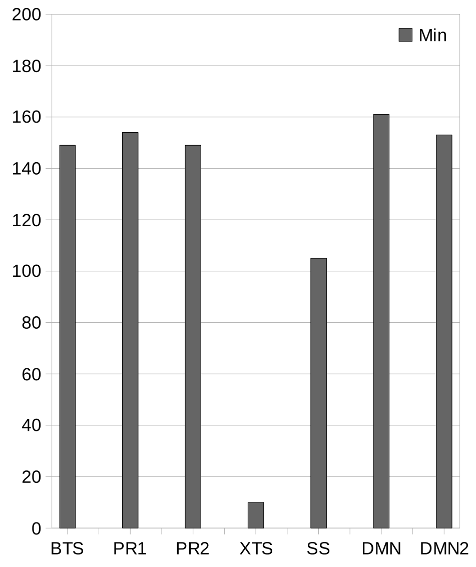

Fig. 5. Results for IDP.

The figure 5 only shows for each algorithm the number of optimal solutions found with the objective function z_5. With the other objectives, the number of optimal solutions found is zero, that is why we did not show them on the diagram. We discussed about it, and came to the conclusion that maybe the other functions do not enough discriminate the different solutions. With our implementation, it also takes more time to compute their value with the IDP problem, because of that, the search loses a lot of time at each iterations, and explores a much more smaller part of the search space than with the function z_5. For this problem, we can see that the eXploring Tabu Search does not give good results. This can be due to a too early "backtracking". As we saw previously, after a fixed number of consecutive non improving iterations the search goes back in a previous configuration and applies the second best move. In the case of the IDP problem, it could take much m ore iteration to improve the value of the objective function than for the SRAP problem. Indeed, the value of the objective function depends on the number of partition in which a customer belongs, while an iteration moves only one edge ; and to reduce its value by only one it could need to move several edges.

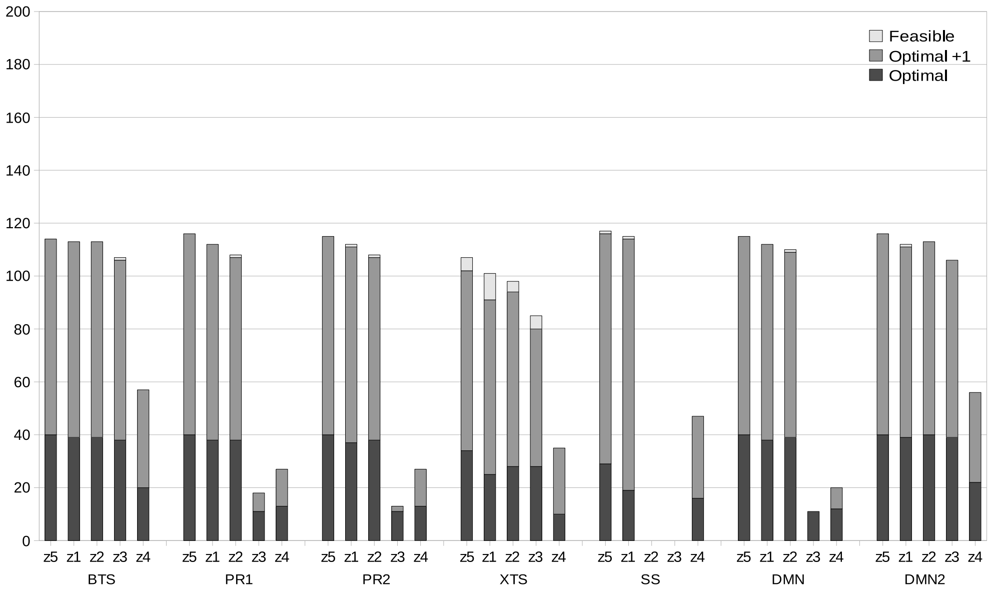

Fig. 6. Results for SRAP.

Figure 6 shows for each algorithm and each objective function, the number of instances for which the search has found an optimal solution, i.e. a solution with k_{lb} partitions (in dark gray on the diagram) ; the number of those for which the best feasible solution found has $k_{lb} + 1$ partitions (in gray) ; and, in light gray, the number of instances for which it has found a feasible solution with more than $k_{lb} + 1$ partitions. From the objective functions perspective, we can see that z_4, supposedly the most improve one, is not that good in the Comet implementation. However the one we add, z_5, is always slightly better than the other ones.

Against all odds, the Basic Tabu Search on all the objective functions, is as good as the other search algorithms. Still on the local search algorithms, we can see that the second version of the Diversification by Multiple Neighborhoods, is much more better than the first one with the objectives z_3 and z_4.

The details of our results are shown in the appendix B.

6 Conclusion

The purpose of this internship was to reproduce with Comet the results obtained, for the SONET Design Problems, by R. Aringhieri and M. DellAmico in 2005 (see [2] for details).

To do so, we have implemented in Comet the algorithms and the objective functions described in this report. We found relevant to add a variant of one of their local search algorithm and a new objective function. Unfortunately, we can not exactly compare our results to theirs because the set of 230 instances they have generated is not available. However, for the IDP problem, we obtained better results for 15 instances over the 160 compared, and similar results for the other instances. It would be interesting to have all the instances and the results to fully compare our results.

Based only on our results, we can say that our objective function implemented in Comet founds a bit more of good solutions than theirs. There is still some work to do in order to know if our results are really better than theirs on the complete set of instances.

Acknowledgments

I would like to thank all the persons that made this internship possible. First of all, Charlotte Truchet for helping me finding it, and Pascal Van Hentenryck for accepting me as an intern ; and for being both my advisors.

All my teachers, especially Xavier Gandibleux, Laurent Granvilliers, Anthony Przybylski, Irena Rusu, Christophe Jermann, Jean-Jacques Loiseau, Nicolas Beldiceanu.

All the administrative staff, Lori Agresti, Lauren Clarke, Janet Eager, JaneMcllmail, Dorinda Moulton, at Brown ; and Christine Brunet, Isabelle Condette, Martine Coignac, Catherine Fourny, Diana Gaudin, Brigitte Misser, Anne-Franoise Quin, at Nantes.

Carleton Coffrin, Justin Yip, Lucile Robin and Olya Ohrimenko the best optimization students, for all the fun and the optimization lab meetings.

Dana O'Malley, Eva Cohen and Jack Spencer for being great roommates.

Natalie McGarvey, Alice Costas, Alison Nguyen, Hilary Merzbacher, Jhon Clavijo, Carlos V, the Art people ; Aggeliki Tsoli, Wenjin Zhou, Fabio Vandin, Dan Grollman, Ariane Buffum, Anna Ritz, Yuri Malitsky, Stu Black, David McClosky, Micha Elsner, Serdar Kadioglu, Jan Bandouch, Laura Sevilla Lara, Eric Sodomka, Kevin Tierney, the CS guys for all the fun. And all the wonderful people that I met there.

A special thank to my parents, my siblings and my boyfriend for being around.

References

1. Van Hentenryck, Pascal and Michel, Laurent Constraint-Based Local Search The MIT Press (2005)
2. Aringhieri, Roberto, Dell'Amico, Mauro : Comparing Metaheuristic Algorithms for Sonet Network Design Problems Journal of Heuristics 11, 35–57 (2005)
3. Dell'Amico, Mauro, Trubian, Marco : Solution of Large Weighted Equicut Problems European J. Oper. Res 106(2/3), 500–521 (1998)
4. Glover, Fred, Laguna, Manuel : Tabu Search Boston. Kluwer Academic Publishers (1997)
5. Glover, Fred : A Template for Scatter Search and Path Relinking Lecture Notes in Computer Science 1363, 13–54 (1997)
6. Goldschmidt, Olivier, Laugier, Alexandre, Olinick, Eli V.: SONET/SDH Ring Assignment with Capacity Constraints Discrete Appl. Math. 129, 99–128 (2003)
7. Goldschmidt, Olivier, Hochbaum, Dorit S., Levin, Asaf, Olinick, Eli V.: The Sonet Edge-Partition Problem Networks 41, 3–23 (2003)
8. Lee, Youngho, Sherali, Hanif D., Han, Junghee, Kim, Seong-in : A Branch-and-Cut Algorithm for Solving an Intraring Synchronous Optical Network Design Problem Networks 35, 223–232 (2000)

A COMET

In this appendix, some examples of COMET will be shown. All of them use the *local search* library `cotls`.

The Statement 1 shows the neighborhood used for the objectives functions z_1, z_2, z_3, z_5. The first line define and initialize the `NeighborSelector` used, as we want to minimize the objective, we use the `MinNeighborSelector` . As describe in the section 4.2 we first choose a partition p and a node x such as x is not in the partition p, the move is not taboo, and that the assignment minimize the most the objective (`z.getAssignDelta(x, p)`). From line 9 to 12, the `Closure` is described, the action that will be done if the assignment is choose. If this move generate a non-feasible solution, line 14, it choose another node y to swap with x such as it minimize the objective (`z.getSwapDelta(x, y)`). Like at the lines 9-12, the lines 18-22 define the `Closure`, the action that will be done if the swap is choose.

Statement 1 Neighborhood for objective functions z_1, z_2, z_3, z_5

```
1   MinNeighborSelector NS();
2   var{int} violations = S.violations();
3   range partitions = P.getPartitions();
4   range items = P.getItems();
5
6   selectMin(p in partitions, x in items :
7     tabu[x, p] < it && p != P.getPartition(x))(z.getAssignDelta(x, p)) {
8
9       neighbor(z.getAssignDelta(x, p), NS) {
10          tabu[x, P.getPartition(x)] = it + tbl;
11          P.setPartition(x, p);
12      }
13
14      if (violations + S.getAssignDelta(x, p) > 0) {
15          selectMin(y in P.getItemsPartition(p) :
16            tabu[y, P.getPartition(x)] < it) (z.getSwapDelta(x, y)) {
17
18              neighbor(z.getSwapDelta(x, y), NS) {
19                  tabu[x, P.getPartition(x)] = it + tbl;
20                  tabu[y, p] = it + tbl;
21                  P.swapPartition(x, y);
22              }
23          }
24      }
25  }
```

The Statement 2 illustrates the Tabu Search. The core of the search is at line 23, when it chooses the next move. The function `neighborhood(tabu, it,` `tbl)` has been introduce previously in Statement 1. This search tries to find a feasible solution, if it does find one, it reduces the number of partitions by one (lines 49-54) and continues the search until it reaches the lower bound or until the time limit is exceeded. The Tabu Search incorporates a intensification component. It returns to the best solution found so far if non improvement has been taken place for a number of iterations. This intensification is located in lines 36-37. It uses a variable `solution` to keep the best solution and a variable `stable` to count the number of consecutive non improving iterations. It also integrates a restarting component. If no feasible solution has been found after a number of iterations the search restarts. Restarting prevents the search to be trapped in a region of the search where there are no good solutions or where good solutions are hard to reach. The restarting component is isolated in lines 38-44. It uses the `delay` mode to propagate the random assignments globally.

The last statement, Statement 3, shows how to make differentiable objects in Comet . To illustrate our words, we use the basic objective function z_0. Firstly, the code in lines 10-20 is the constructor for this objective. It initializes the incremental variable `objective` to the number of non empty partitions. If the size of a partition changes, it will automatically update `objective`. The functions `getItems()` and `evaluation()`, described in line 22 and 24, are just two accessors.

The functions `getAssignDelta(int v, int i)` and `getSwapDelta(int v1, int v2)` return the value between the value of the objective if we apply the move considerated and the evaluation of the current solution. The fact of swapping to nodes, will never reduce the number of rings. That is why the function `getSwapDelta(int v1, int v2)` always return 0 (lines 39-41). On the other hand, an assignment can reduce the number of rings. The function `getAssignDelta(int v, int i)`, at lines 26 to 37 compute the difference between the current solution and the one generated by assigning the item `v` to the partition `i`.

Statement 2 Tabu Search

```
range items = S.getItems();
range partitions = P.getPartitions();

int tbl = tblStart; int it = 1;
int stable = 0; int stableLimit = 250000;
int restartFreq = 10000;

int tabu[items, partitions] = -1;

var{int} violations = S.violations();
var{int} evaluation = Z.evaluation();
int best = evaluation;

Solution solution(ls);
Solution bestSol(ls, new MinimizeIntValue(Z.evaluation()));

MinNeighborSelector NS();
int t0 = System.getWCTime();

while (evaluation > goal && System.getWCTime()-t0 < time) {
    while (violations > 0 && System.getWCTime()-t0 < time) {

        NS = neighborhood(tabu, it, tbl);

        if(NS.hasMove())
            call(NS.getMove());

        if (evaluation < bestSol.getObjectiveValue().getInt())
            bestSol = new Solution(ls, new MinimizeIntValue(evaluation));

        if (evaluation < best) {
            best = evaluation;
            solution = new Solution(ls);
            stable = 0;
        } else {
            if (stable == stableLimit) {
                solution.restore(); stable = 0;
            } else {
                stable++;
                if (it % restartFreq == 0) {
                    with delay(ls)
                       P.randomize();
                }
            }
            it++;
        }
    }

    if (evaluation > goal) {
        tbl = tblStart; stable = 0;
        reducePartitionsBy1(type);
        if (evaluation < bestSol.getObjectiveValue().getInt())
           bestSol = new Solution(ls, new MinimizeIntValue(evaluation));
    }
}
bestSol.restore();
```

Statement 3 Objective function z_0

```
class PartitionObjectiveZ0 implements PartitionObjective {
    Solver<LS> ls;
    PartitionLoadAndCutLoad p;

    var{int} objective;

    range nodes;
    range partitions;

    PartitionObjectiveZ0(Solver<LS> ls, PartitionLoadAndCutLoad p){
        ls = ls;
        p = p;

        nodes = p.getNodes();
        partitions = p.getPartitions();

        objective = new var{int}(ls, 0..nodes.getSize());
        var{int} partitionSize[i in partitions] = p.getPartitionSize(i);
        objective <- card(setof(x in partitions)(partitionSize[x] > 0));
    }

    range getItems() {return p.getItems();}

    var{int} evaluation() {return objective;}

    int getAssignDelta(int v, int i){
        int fromPartition = p.getPartition(v);
        int toPartition = i;

        if(fromPartition == toPartition)
            return 0;

        int fromPartitionSize = p.getPartitionSize(fromPartition);
        int toPartitionSize = p.getPartitionSize(toPartition);

        return min(0, fromPartitionSize-2) - min(0, toPartitionSize-1);
    }

    int getSwapDelta(int v1, int v2){
        return 0;
    }
}
```

B Additional Results

In this section, we report the results obtained with the algorithms. Each table represents a set of 40 instances. Tables 1 and 2 show the results for, respectively, the high-demand and the low-demand geometric graphs. Similarly tables 3 and 4 summarize the results for, high and low demand random graphs. The results for the instances generated by Y. Lee, H. Sherali, J. Han and S. Kim in 2000 ([8]), are illustrated in Table 5. The following notation is used in all the tables :

n	is the number of nodes in the graph
m	is the number of edges in the graph
T	is the sum of all the demands in the graph
k_{lb}	is the lower bound
SRAP	is the number of rings obtained
	if a feasible solution is not found, the cell will contained a dash "-"
IDP	is the the smallest number of ADM found.

Graph	n	m	T	k_{lb}	SRAP	IDP
GH.15.1	15	43	921	2	3	19
GH.15.2	15	50	1032	2	3	18
GH.15.3	15	42	888	2	2	17
GH.15.4	15	75	1590	3	-	26
GH.15.5	15	76	1603.5	3	-	27
GH.15.6	15	41	871.5	2	2	16
GH.15.7	15	36	739.5	2	2	17
GH.15.8	15	52	1110	2	3	20
GH.15.9	15	48	1023	2	3	20
GH.15.10	15	39	790.5	2	2	16
GH.25.1	25	57	1191	2	3	28
GH.25.2	25	52	1069.5	2	2	27
GH.25.3	25	50	1032	2	2	27
GH.25.4	25	64	1323	3	3	28
GH.25.5	25	86	1830	3	-	35
GH.25.6	25	91	1878	4	-	35
GH.25.7	25	71	1515	3	4	32
GH.25.8	25	63	1326	3	3	28
GH.25.9	25	63	1309.5	3	3	29
GH.25.10	25	88	1824	3	-	35
GH.30.1	30	72	1504.5	3	3	33
GH.30.2	30	75	1636.5	3	4	35
GH.30.3	30	78	1665	3	4	36
GH.30.4	30	72	1504.5	3	3	33
GH.30.5	30	73	1546.5	3	4	34
GH.30.6	30	87	1824	3	-	37
GH.30.7	30	94	1995	4	-	41
GH.30.8	30	92	1959	4	-	41
GH.30.9	30	75	1585.5	3	4	34
GH.30.10	30	57	1207.5	2	3	33
GH.50.1	50	110	2310	4	5	57
GH.50.2	50	130	2716.5	5	-	59
GH.50.3	50	143	3007.5	5	-	64
GH.50.4	50	102	2140.5	4	4	54
GH.50.5	50	107	2226	4	5	55
GH.50.6	50	116	2427	4	5	57
GH.50.7	50	108	2275.5	4	5	56
GH.50.8	50	126	2697	5	-	59
GH.50.9	50	142	3046.5	5	-	63
GH.50.10	50	112	2428.5	4	-	59

Table 1: Results for High-Demand Geometric Graphs (B = 622 Mbs)

Graph	n	m	T	k_{lb}	SRAP	IDP
GL.15.1	15	41	313.5	3	3	18
GL.15.2	15	52	378	3	-	23
GL.15.3	15	39	330	3	-	21
GL.15.4	15	41	295.5	2	3	16
GL.15.5	15	60	454.5	3	-	26
GL.15.6	15	46	343.5	3	-	22
GL.15.7	15	44	313.5	3	3	20
GL.15.8	15	55	403.5	3	-	24
GL.15.9	15	344	247.5	2	3	18
GL.15.10	15	62	474	4	-	27
GL.25.1	25	54	403.5	3	4	29
GL.25.2	25	61	480	4	-	31
GL.25.3	25	49	354	3	3	30
GL.25.4	25	59	429	3	4	27
GL.25.5	25	61	484.5	4	-	32
GL.25.6	25	74	561	4	-	37
GL.25.7	25	49	325.5	3	3	28
GL.25.8	25	57	445.5	3	4	31
GL.25.9	25	63	471	4	-	33
GL.25.10	25	60	489	4	-	31
GL.30.1	30	57	424.5	3	4	34
GL.30.2	30	57	436.5	3	4	34
GL.30.3	30	57	408	3	4	33
GL.30.4	30	54	397.5	3	4	34
GL.30.5	30	58	391.5	3	3	33
GL.30.6	30	54	405	3	4	32
GL.30.7	30	57	412.5	3	4	34
GL.30.8	30	60	430.5	3	4	34
GL.30.9	30	57	418.5	3	4	32
GL.30.10	30	76	567	4	-	38
GL.50.1	50	85	616.5	4	5	54
GL.50.2	50	92	729	5	-	57
GL.50.3	50	98	757.5	5	-	58
GL.50.4	50	84	649.5	5	6	55
GL.50.5	50	90	708	5	-	54
GL.50.6	50	92	694.5	5	-	58
GL.50.7	50	85	601.5	4	5	52
GL.50.8	50	86	646.5	5	5	51
GL.50.9	50	63	499.5	4	4	50
GL.50.10	50	87	652.5	5	-	57

Table 2: Results for Low-Demand Geometric Graphs (B = 622 Mbs)

Graph	n	m	T	k_{lb}	SRAP	IDP
RH.15.1	15	48	990	2	3	21
RH.15.2	15	58	1203	2	-	23
RH.15.3	15	53	1113	2	3	22
RH.15.4	15	38	778.5	2	2	18
RH.15.5	15	47	955.5	2	3	21
RH.15.6	15	45	930	2	3	21
RH.15.7	15	39	804	2	2	19
RH.15.8	15	41	891	2	2	20
RH.15.9	15	43	909	2	3	21
RH.15.10	15	58	1197	2	-	22
RH.25.1	25	62	1297.5	3	3	34
RH.25.2	25	53	1069.5	2	3	32
RH.25.3	25	52	1084.5	2	3	31
RH.25.4	25	61	1314	3	3	35
RH.25.5	25	67	1402.5	3	4	36
RH.25.6	25	62	1296	3	3	35
RH.25.7	25	51	1072.5	2	3	32
RH.25.8	25	66	1353	3	4	36
RH.25.9	25	41	831	2	2	28
RH.25.10	25	49	997.5	2	3	30
RH.30.1	30	50	1056	2	3	35
RH.30.2	30	67	1402.5	3	4	41
RH.30.3	30	52	1096.5	2	3	34
RH.30.4	30	56	1185	2	3	34
RH.30.5	30	82	1656	3	-	44
RH.30.6	30	70	1482	3	4	41
RH.30.7	30	67	1404	3	4	41
RH.30.8	30	62	1332	3	3	38
RH.30.9	30	60	1288.5	3	3	39
RH.30.10	30	64	368	3	3	39
RH.50.1	50	81	1738.5	3	4	59
RH.50.2	50	68	1413	3	3	56
RH.50.3	50	89	1872	4	-	64
RH.50.4	50	89	1848	3	4	64
RH.50.5	50	66	1380	3	3	54
RH.50.6	50	67	1384.5	3	3	56
RH.50.7	50	90	1876.5	4	-	65
RH.50.8	50	80	1689	3	4	60
RH.50.9	50	73	1573.5	3	4	57
RH.50.10	50	86	1870.5	4	4	61

Table 3: Results for High-Demand Random Graphs (B = 155 Mbs)

Graph	n	m	T	k_{lb}	SRAP	IDP
RL.15.1	15	32	250.5	2	3	20
RL.15.2	15	43	340.5	3	-	23
RL.15.3	15	28	208.5	2	2	19
RL.15.4	15	38	288	2	3	21
RL.15.5	15	40	310.5	3	-	22
RL.15.6	15	34	249	2	3	20
RL.15.7	15	51	405	3	-	26
RL.15.8	15	36	265.5	2	3	20
RL.15.9	15	32	252	2	3	19
RL.15.10	15	40	298.5	2	3	22
RL.25.1	25	51	364.5	3	4	33
RL.25.2	25	42	325.5	3	3	30
RL.25.3	25	54	438	3	-	36
RL.25.4	25	50	358.5	3	4	35
RL.25.5	25	44	345	3	4	32
RL.25.6	25	51	364.5	3	4	34
RL.25.7	25	49	373.5	3	4	33
RL.25.8	25	40	303	2	3	28
RL.25.9	25	47	372	3	4	33
RL.25.10	25	48	361.5	3	4	32
RL.30.1	30	44	336	3	3	34
RL.30.2	30	66	487.5	4	-	45
RL.30.3	30	58	415.5	3	4	39
RL.30.4	30	55	432	3	4	39
RL.30.5	30	47	361.5	3	4	36
RL.30.6	30	56	429	3	-	40
RL.30.7	30	47	354	3	3	36
RL.30.8	30	54	373.5	3	4	40
RL.30.9	30	63	487.5	4	-	43
RL.30.10	30	53	405	3	4	38
RL.50.1	50	70	513	4	5	58
RL.50.2	50	85	643.5	5	-	67
RL.50.3	50	65	489	4	4	58
RL.50.4	50	66	484.5	4	4	58
RL.50.5	50	61	471	4	4	53
RL.50.6	50	62	462	3	4	57
RL.50.7	50	71	529.5	4	5	60
RL.50.8	50	81	609	4	-	65
RL.50.9	50	68	540	4	-	57
RL.50.10	50	62	459	3	4	52

Table 4: Results for Low-Demand Random Graphs (B = 155 Mbs)

Graph	n	m	T	k_{lb}	SRAP	IDP
LSHK.15.30.1	15	30	355.5	5	-	28
LSHK.15.30.2	15	30	370.5	6	-	28
LSHK.15.30.3	15	30	372	6	-	28
LSHK.15.30.4	15	30	337.5	5	-	27
LSHK.15.30.5	15	30	354	5	-	27
LSHK.15.30.6	15	30	375	6	-	28
LSHK.15.30.7	15	30	316.5	5	-	27
LSHK.15.30.8	15	30	363	6	-	28
LSHK.15.30.9	15	30	279	4	-	25
LSHK.15.30.10	15	30	285	4	-	26
LSHK.20.30.1	20	30	360	5	-	32
LSHK.20.30.2	20	30	304.5	5	-	29
LSHK.20.30.3	20	30	340.5	5	-	29
LSHK.20.30.4	20	30	385.5	6	-	33
LSHK.20.30.5	20	30	364.5	6	-	30
LSHK.20.30.6	20	30	324	5	-	31
LSHK.20.30.7	20	30	357	5	-	32
LSHK.20.30.8	20	30	304.5	5	-	28
LSHK.20.30.9	20	30	297	5	-	30
LSHK.20.30.10	20	30	279	4	-	29
LSHK.20.35.1	20	35	348	5	-	30
LSHK.20.35.2	20	35	463.5	7	-	35
LSHK.20.35.3	20	35	340.5	5	-	32
LSHK.20.35.4	20	35	323	6	-	34
LSHK.20.35.5	20	35	315	5	-	31
LSHK.20.35.6	20	35	322.5	5	-	30
LSHK.20.35.7	20	35	361.5	6	-	33
LSHK.20.35.8	20	35	3297	5	-	30
LSHK.20.35.9	20	35	351	5	-	33
LSHK.20.35.10	20	35	331.5	5	-	30
LSHK.30.35.1	30	35	378	6	-	36
LSHK.30.35.2	30	35	328.5	5	6	36
LSHK.30.35.3	30	35	318	5	-	35
LSHK.30.35.4	30	35	315	5	-	35
LSHK.30.35.5	30	35	402	6	-	37
LSHK.30.35.6	30	35	367.5	6	-	37
LSHK.30.35.7	30	35	406.5	6	-	36
LSHK.30.35.8	30	35	307.5	5	6	34
LSHK.30.35.9	30	35	411	6	-	37
LSHK.30.35.10	30	35	357	5	-	36

Table 5: Results for the second set of instances (B = 72 Mbs)