Sonet Network Design Problems

Marie Pelleau[†,‡] marie.pelleau@univ-nantes.fr

Pascal Van Hentenryck[‡] pvh@cs.brown.edu

Charlotte Truchet[†] Charlotte.Truchet@univ-nantes.fr

† Université de Nantes
Département informatique
2 rue de la Houssinière
44322 Nantes cedex 3
France

‡ Brown University
Computer Science
115 Waterman Street
Providence, RI 02912
USA

This paper presents a new method and a constraint-based objective function to solve two problems related to the design of optical telecommunication networks, namely the Synchronous Optical Network Ring Assignment Problem (SRAP) and the Intra-ring Synchronous Optical Network Design Problem (IDP). These network topology problems can be represented as a graph partitioning with capacity constraints as shown in previous works. We present here a new objective function and a new local search algorithm to solve these problems. Experiments conducted in COMET allow us to compare our method to previous ones and show that we obtain better results.

1 Introduction

This paper presents a new algorithm and an objective function to solve two real-world combinatorial optimization problems from the field of network design. These two problems, the Synchronous Optical Network Ring Assignment Problem (SRAP) and the Intra-ring Synchronous Optical Network Design Problem (IDP), have been shown $\mathscr{NP}$-hard and have already been solved by combinatorial optimization techniques. This work extends the seminal ideas introduced by R. Aringhieri and M. Dell'Amico in 2005 in [2].

This paper 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. We will also introduce the models in a constrained optimization format for the two problems. We then present the previous works on SRAP and IDP in section 2. Section 3 describes the key ingredients necessary to implement the local search algorithms. Finally, the results are shown in Section 4.

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.

Submitted to:
LSCS 2009

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 paper, 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.

The model associated to these topologies 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 corresponds 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 $d_{uu} = 0$.

1.1.1 First topology (SRAP)

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.

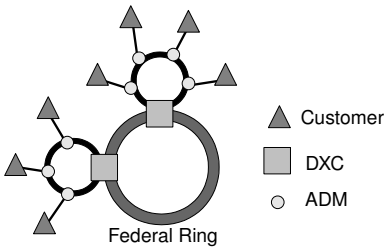

Figure 1: A SONET network with DXC.

Model This topology 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.

Formally, let $V_1, V_2, \ldots, V_k$ be a partitioning of V in k subsets. 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 2 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.

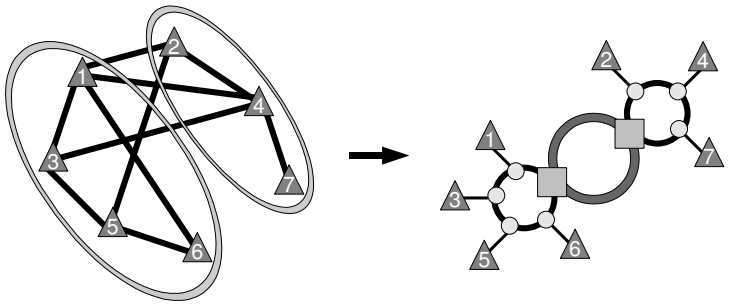

Figure 2: Relation between the node partitioning and the network topology.

For this problem we can easily compute a lower bound k_{lb} introduced in [6]. In fact, we want to know the minimum number of partitions needed to route all the traffic. Reasonning on the total traffic amount, if we sum all the traffic demands of the graph and divide it by the bandwidth B, we trivially obtain a minmum for the number of rings, that is, a lower bound of the number of partitions. Moreover, we cannot have fractional part of partition, that is why we take the upper round of this fraction.

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

1.1.2 Second topology (IDP)

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 3 illustrates this topology.

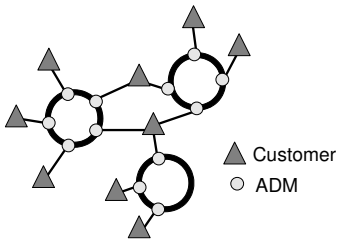

Figure 3: A SONET network without DXC.

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

Formally, let $E_1, E_2, \ldots, E_k$ be a partitioning of E in k subsets and $Nodes(E_i)$ be the set of endpoint 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 connected to two rings.

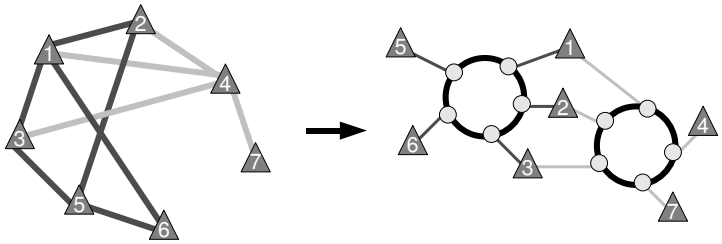

Figure 4: Relation between the edge partitioning and the network topology.

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].

Both of these problems are $\mathscr{NP}$-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). The principal constraint, the load constraint, is similar to a capacity constraint, yet different: a capacity constraint holds on the variables *in* the sum, while the load constraint holds on the variables *below* the sum. The question is how to choose the d_{uv} (which are data) that count for the load.

1.2 Brief introduction of Local Search

In order to efficiently and quickly solve these two combinatorial optimization problems, we decided to use Local Search instead of an exact algorithm. Indeed, it permits to search in a efficiently way among all the candidate solutions, by performing steps from a solution to another.

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 like the Maximum Clique Problem (L. Cavique, C. Rego and I. Themido in 2001 in [9]), or the Graph Coloring Problem (J.P. Hansen and J.K. Hao in 2002 in [10]). 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 perform 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 and 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. In our case, we can compute the lower bound to either prove that the obtained solution is optimum, or estimate its optimality, hence local search is well suited.

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 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. A pseudo-code is given on figure 1.

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 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.

```
Choose or construct an initial solution S_0 ;
S ← S_0 ;                                          /* S is the current solution */
S* ← S_0 ;                                         /* S* is the best solution so far */
bestValue ← objValue(S_0) ;                        /* bestValue is the evaluation of S* */
T ← ∅ ;                                            /* T is the Tabu list */
while Terminaison criterion not satisfied do
    N(S) ← all the neighboring solutions of S ;    /* Neighborhood exploration */
    S ← a solution in N(S) minimizing the objective ;
    if objValue(S) < bestValue then                /* The solution found is better than S* */
        S* ← S ;
        bestValue ← objValue(S) ;
    end
    Record tabu for the current move in T (delete oldest entry if necessary) ;
end
```

Algorithm 1: Tabu Search

1.3 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.

As we can see on the constraint (1), the sum are on datas (d_{uv}) and are determined by the variables ($u \in V_i, v \in V, v \neq u$). We will rely on COMET 's built-in invariants to define a constraint to represent the load.

2 Related work

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

2.1 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 shows the pseudo code for the edge-based heuristic.

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

```
F ← E ;          /* Initialize the set of edges that have not been used yet */
∀v ∈ V ring(v) ← v ;                  /* Assign each node to a different ring */
while F ≠ ∅ do          /* There is still some edges that have not been used */
    Choose a maximum capacity edge (u,v) ∈ F ;
    i ← ring(u), j ← ring(v) ;
    if V_i ∪ V_j is a feasible ring then   /* Merging the rings gives a feasible ring */
        ∀v ∈ V_j ring(v) ← i ;
        F ← F\{(x,y)|ring(x) = i, ring(y) = j} ;
    else
        F ← F\{(u,v)} ;
    end
end
```

Algorithm 2: Edge-Based Heuristic

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 3.
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.

```
U ← V ;          /* Initialize the set of nodes that have not been used yet */
for i = 1 to k do                  /* Assign k random nodes to the k partitions */
    Choose u ∈ U, V_i ← u, U ← U\{u}
end
while U ≠ ∅ do                                   /* There are some unused nodes */
    Choose a minimum capacity ring V_i
    Choose u ∈ U to maximize Σ_{v∈V_i} d_uv
    ring(u) ← V_i, U ← U\{u} ;                          /* Assign u to V_i */
end
```

Algorithm 3: Node-Based Heuristic

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 the smallest value obtained for k as input. 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.

2.2 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.

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

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.

2.3 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 further.

Previously, we saw that with local search it is necessary to define a neighborhood to choose the next solution. The authors of [2] use 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 try 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 solution should have very dense rings. They are both greedy and rely on a clique algorithm. In graph theory, a clique in an undirected graph $G = (V,E)$ is a subset of the vertex set $C \subseteq V$, such that for every two vertices in C, there exists an edge connecting the two. Finding a clique is not that easy, a way to do it is to use an "Union-Find" strategie, *Find* two clique A and B such that each node in A is adjacent to each node in B then merge the two cliques (*Union*). The associated heuristic starts by considering each node to be a clique of size one, and to merge two cliques into a larger clique until there are no more possible merges.

In the third method, *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 4 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.

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.

$U \leftarrow E$;
$r \leftarrow 0$;
while $U \neq \emptyset$ **do**
  Heuristicaly find a clique $C \subset U$ such that $weight(C) \leq B$;
  `/* Search a ring such that the weight of the ring plus the weight of the clique does not exceed B and is the biggest possible */`
  $j \leftarrow min\{B - weight(E_i) - weight(C) : i \in \{1,\dots,k\}, B - weight(E_i) - weight(C) \geq 0\}$;
  if $j = null$ **then**
    $r++$;
    $j \leftarrow r$;
  end
  $E_j \leftarrow E_j \cup C$;
  $U \leftarrow U \backslash C$;
end

Algorithm 4: Clique-BF

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

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, by taking into account only one ring, the one with the highest overload. 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.

3 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 objective functions. Finally we present the different local search algorithms.

3.1 Starting solution

Most of the times, local search starts from an random initial solution. However we have tested other possibilities and two other options proved to be more efficient.

The best initializing method assigned all the items, nodes for SRAP or edges for IDP, to the same partition. This solution is certainly unfeasible as all the traffic is on only one ring. This biases the search towards solutions with a minimum value for the cost and a very bad value for the capacity constraints' violations. Astonishingly this is the one that gave us the best results on large instances.

We had good confidence in another one which first computes the lower bound k_{lb} (described in section 2) and randomly assigns all the items to exactly k_{lb} partitions. The idea was to let the Local Search reduce the number of violations. This starting solution was good on small instances and not so good on large ones. It was the same with a random solution, which corresponds, for these problems, to a solution where all the items are randomly assigned to a partition.

3.2 Neighborhoods

In a generic partitioning problem there are usually two basic neighborhoods. From a given solution, we can move an object from a subset to another subset or swap two objects assigned to two different subsets. For SRAP a neighboring solution is produced by moving a node from a ring to another (including a new one) or by swapping two nodes assigned to two different rings. The same kind of neighborhood can be used for IDP: moving an edge from a ring to another or swapping two edges.

In some cases it is more efficient to restrain the neighborhood to the feasible space. We have tested different variants of the basic neighborhood applying this idea, by choosing the worst partition (wrt. the capacity constraint) and even by assigning it to the partition with the lowest load. Anyway it appears to be less efficient than the basic one. As will be seen later it seems that on these problems it is necessary to keep the search as broad as possible.

3.3 Objective function

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

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

where

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

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

This objective function minimizes the basic function z_0 and penalizes the unfeasible solutions, but contrarily to the previous objectives, this penalty is based on all the constraints. We consider that every constraint is violated by a certain amount (its current load minus B). By summing all the violations of the current solution, we obtain the total violation for all the constraints, and we can precisely say how far we are from a feasible one. If the current solution is feasible, $\sum_{p \in \mathit{partitions}} \mathit{violations}(p) = 0$.

This objective has also the nice property that it is merely local, depending only on the current solution and not on the other moves. Notice that a feasible solution with 4 rings will be preferred to an unfeasible solution with 3 rings, as z_0 is much smaller than the load of a ring.

3.4 Local Search

We have proposed a new algorithm called DMN2 which proved to be efficient on both problems. It is a variant of the Diversification by Multiple Neighborhood (DMN) proposed in [2]. DMN is based on Tabu Search, and adds a mechanism to perform diversification when the search is going round and round without improving the objective (eventhough it is not a local minimum). This replaces the classical random restart steps. We refine this particular mechanism by proposing several ways of escaping such areas.

More precisely, on our problems, after a series of consecutive non improving iterations, the DMN algorithm empties a partition by moving all its items to another partition, disregarding the capacity constraint and locally minimizing the objective function. There is a particular case for our function z_5, because it integrates the capacity constraints. In this case, the "z_5" version of DMN we have implemented moves the items to another partition minimizing z_5. The results in [2] show a general trend on SRAP and IDP: the more diversification is performed, the better are the results. Following this idea, we propose different ways of perfoming the DMN step, which gives our algorithm DMN2. In DMN2, 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, generates a random solution, in the same way as a classic random restart. Finally, d_3 randomly chooses a number m in the range [1, k], where k is the number of rings, and applies m random moves.

In the end, our general algorithm starts with a solution where all the items are in the same partition. Then it 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 assigning 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.

4 Results

The objective functions and the metaheuristics, respectively described in Section 3.3 and Section 3.4, 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 litterature.

4.1 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 number of T1 lines required for the anticipated volume of traffic between u and v. A T1 line has an approximate capacity of 1.5 Mbs. The number of T1 lines is randomly picked in the interval $[3,7]$, for low-demand cases, while it is selected from the range $[11,17]$, for the high-demand cases. The generated graphs 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 unfeasible by R. Aringhieri and M. Dell'Amico using CPLEX 8.0 (see [2]).

```
Construct the initial solution S_0 ;
S ← S_0 ;                                          /* S is the current solution */
S* ← S_0 ;                                         /* S* is the best solution so far */
bestValue ← objValue(S_0) ;                        /* bestValue is the evaluation of S* */
T ← ∅ ;                                            /* T is the Tabu list */
nonImprovingStep ← 0 ;
/* nonImprovingStep is the number of consecutive non improving iterations */
while bestValue > lowerbound and we still have time do
    N(S) ← all the neighboring solution of S ;
    S ← the solution in N(S) minimizing the objective ;
    if objValue(S) < bestValue then                /* The solution found is better than S* */
        S* ← S ;
        bestValue ← objValue(S) ;
        nonImprovingStep ← 0 ;
    else                                           /* The iteration did not improve S* */
        nonImprovingStep++ ;
    end
    Record tabu for the current move in T (delete oldest entry if necessary) ;
    if nonImprovingStep = maxNonImprovingStep then
        /* Stuck in a non improving area:  time for diversification        */
        nonImprovingStep ← 0 ;
        /* Uniformly chooses the diversification to apply                  */
        d ← UniformRandom(1,3) ;
        switch The value of d do
            case 1                                 /* Empties a partition */
                Randomly choose a partition p ;
                foreach item i on the partition p do
                    Choose the partition p' such that the assignment of i to p' minimizes the
                    objective function and p' ≠ p ;
                    Assign i to the partition p' ;
                end
            case 2                                 /* Random Restarts */
                foreach item i do
                    Randomly choose a partition p ;
                    Assign i to the partition p ;
                end
            case 3                                 /* Applies m random moves */
                m ← UniformRandom(1, numberOfRings) ;
                Apply m moves ;
            end
        end
    end
end
```

Algorithm 5: DMN2

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

4.2 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 3.4 for details). For each algorithm we consider the five objective functions of Section 3.3, but for the SS we use the three functions described in Section 3.4.

We gave a time limit of 5 minutes to each run of an algorithm. However we observed that 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}. In case the lower bound is not reached, we define as a *high-quality solution* a solution for which the evaluation of the objective is equal to $k_{lb}+1$. Remind that objective functions z_2 and z_3 cannot be applied with the Scatter Search.

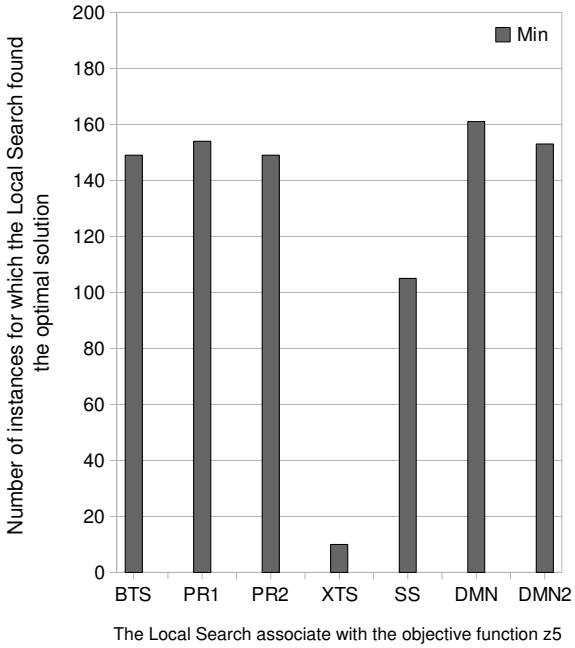

Figure 5: Results for IDP.

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. However the other objectives found good solutions. Our conclusion is that maybe the other functions do not enough discriminate the different solutions. 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". 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 more iterations 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 partitions 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.

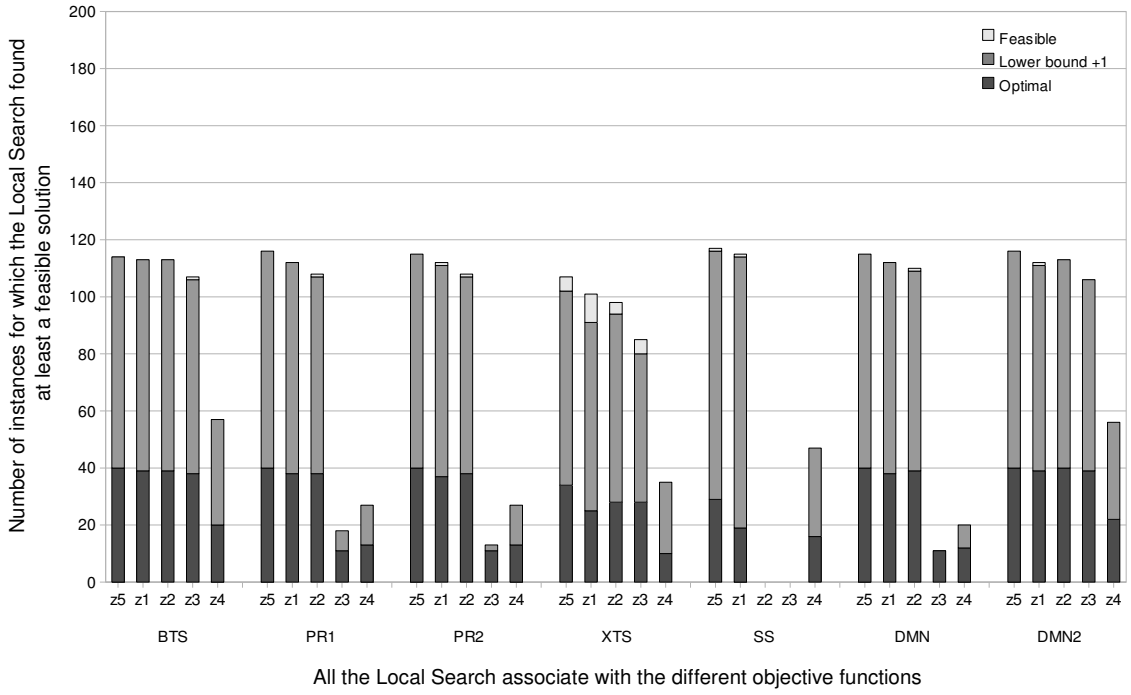

Figure 6: Results for SRAP.

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, supposed to be the most improving one, is not that good in the COMET implementation. However the one we add, z_5, is always 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 better than the first one with the objectives z_3 and z_4.

For the details of our results see the report [11].

5 Conclusion

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

We have implemented in COMET the algorithms and the objective functions described in this paper. We found relevant to add a variant of one of their local search algorithm and a new objective function. Unfortunately, we cannot 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. Unfortunately we did not found their results for the SRAP problem. Still for the problem SRAP, compare to the results obtained by O. Goldschmidt, A. Laugier and E. Olinick in 2003, [6] we obtained better results, we have more instances for wich the algorithm reach the lower bound and less unfeasible instances. It would be interesting to have all the instances and the results to fully compare our results.

In the end we can exhibit two main observations. Firstly, for these two problems, the more an algorithm uses diversification the better it is. Actually, we have tried different intensification methods for the local search algorithms but none of them improved the results, worst, they gave us pretty bad results.

Secondly, based on our results, we can say that our objective function implemented in COMET finds more good solutions than the other ones. It is a constraint-based objective function taking into account the violation of every constraint. Hence it has the asset of being both more generic and precise than the dedicated functions, with better results.

References

[1] Van Hentenryck, Pascal and Michel, Laurent (2005): *Constraint-Based Local Search*. The MIT Press

[2] Aringhieri, Roberto, Dell'Amico, Mauro (2005): *Comparing Metaheuristic Algorithms for Sonet Network Design Problems*. Journal of Heuristics 11, pp. 35–57

[3] Dell'Amico, Mauro, Trubian, Marco (1998): *Solution of Large Weighted Equicut Problems*. European J. Oper. Res 106(2/3), pp. 500–521

[4] Glover, Fred, Laguna, Manuel (1997): *Tabu Search*. Boston. Kluwer Academic Publishers

[5] Glover, Fred (1997): *A Template for Scatter Search and Path Relinking*. Lecture Notes in Computer Science 1363, pp. 13–54

[6] Goldschmidt, Olivier, Laugier, Alexandre, Olinick, Eli V. (2003): *SONET/SDH Ring Assignment with Capacity Constraints*. Discrete Appl. Math. 129, pp. 99–128

[7] Goldschmidt, Olivier, Hochbaum, Dorit S., Levin, Asaf, Olinick, Eli V. (2003): *The Sonet Edge-Partition Problem*. Networks 41, pp. 3–23

[8] Lee, Youngho, Sherali, Hanif D., Han, Junghee, Kim, Seong-in (2000): *A Branch-and-Cut Algorithm for Solving an Intraring Synchronous Optical Network Design Problem*. Networks 35, pp. 223–232

[9] Cavique, Lus, Rego, Csar, Themido, Isabel (2001): *A Scatter Search Algorithm for the Maximum Clique Problem*. Essays and Surveys in Metaheuristics Kluwer pp. 227-244

[10] Hamiez, Jean-Philippe, Hao, Jin-Kao (2002): *Scatter Search for Graph Coloring*. Lecture Notes In Computer Science 2310, pp. 195–213

[11] Pelleau, Marie (2009): *Sonet Network Design Problem*.