RAIRO Operations Research
RAIRO Oper. Res. **37** (2003) 235-247
DOI: 10.1051/ro:2004003

APPROXIMATION ALGORITHMS FOR THE DESIGN OF SDH/SONET NETWORKS

Nadia Brauner[1], Yves Crama[2], Gerd Finke[1], Pierre Lemaire[1] and Christelle Wynants[3]

Abstract. In this paper, a graph partitioning problem that arises in the design of SONET/SDH networks is defined and formalized. Approximation algorithms with performance guarantees are presented. To solve this problem efficiently in practice, fast greedy algorithms and a tabu-search method are proposed and analyzed by means of an experimental study.

Keywords. Graph partitioning, approximations, heuristics, tabu, SONET/SDH networks.

1. Introduction

With the explosion of telecommunications, the optimal design of networks has become a major issue. One of the most important requirements is the reliability of the network, *i.e.* its aptitude for coping with signals and routing them (if possible automatically) even when a link breaks down. However, increased reliability translates into higher design costs, meaning that a compromise has usually to be found between these two conflicting criteria.

[1] Laboratoire Leibniz-IMAG, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France; e-mail: nadia.brauner@imag.fr, gerd.finke@imag.fr, pierre.lemaire@imag.fr

[2] École d'Administration des Affaires, Université de Liège, boulevard du Rectorat 7 (B31), 4000 Liège, Belgique; e-mail: y.crama@ulg.ac.be

[3] Electrabel Quantitive Analysis, 8 boulevard du Régent, 1000 Brussels, Belgique; e-mail: christelle.wynants@electrabel.com

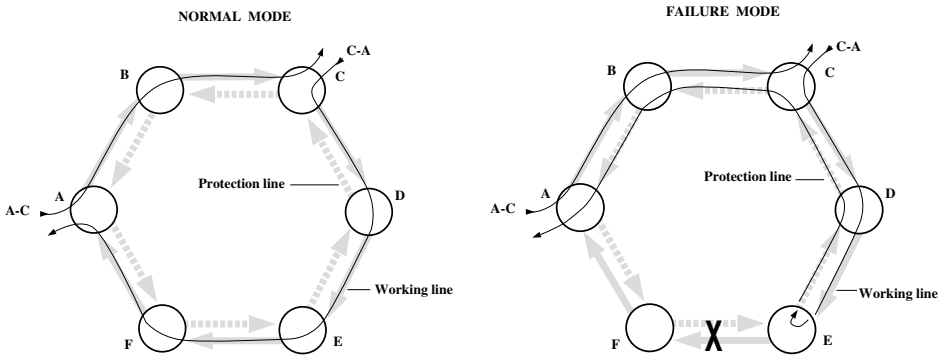

FIGURE 1. A unidirectional self-healing ring.

A *Synchronous Digital Hierarchy* (SDH), also known as a *Synchronous Optical NETwork* (SONET)[1], is a widely-used solution which meets both technical and economical expectations. Such a network is based on basic elements called *Self-Healing Rings* (SHR): nodes of the demand graph are gathered together and physically linked, so as to form cycles, by two separated fibers of opposite direction. At every node, an *Add/Drop Multiplexer* (ADM) is capable of detecting a failure and accordingly rerouting signals along one of the two fibers.

Several protocols exist for SDH/SONET networks. We focus here on the European standard, based on unidirectional rings.

In this case, as illustrated by Figure 1, one fiber is dedicated for normal work, while the other is saved for the cases of failure. The demand between two nodes of a ring is thus routed all the way around the ring (for example: a demand from A to C goes through AB, BC, and the opposite demand from C to A goes through all other links). As a consequence, an ADM must have a *capacity* large enough to be able to handle the sum of all demands in a ring.

Our purpose, in this paper, is to partition a demand graph into SONET/SDH rings at minimum cost.

For further details on SDH/SONET networks, the reader is referred to *The SONET Home Page* [25] and to Demetris Mili's webpage [21] for a more technical description. For precisions on the european standards, the European Telecommunications Standards Institute (ETSI) [5] publishes documentations, for instance [6, 7]. For a more complete view on SONET architectures, technologies, and design methods, see the book by Wu [24].

The reader is also referred to the following related references. Goldschmidt, Laugier and Olinick [13] deal with the problem of partitioning a demand-graph into SONET rings linked together by a "federal link"; Fortz, Sorianio and Wynants [8] propose a tabu-search heuristic for partitioning a demand-graph into SONET rings

[1]SDH is the name of the European standard; SONET is the american standard.

of different capacities. In both cases, only the cost of creating a ring is considered. Other tabu-based solution techniques are proposed by Laguna [17] and by Aringhieri and Dell Amico [1]. Lee *et al.* [18] and Sutter *et al.* [23] propose alternative approaches to this problem, based on integer programming.

In the case of bidirectional SONET rings, once the rings are built, one must cope with the Ring Loading Problem of balancing demands one way or the other around a ring. This problem is well-solved in [16, 22].

In Section 2 we formally define the problem and give complexity results. Section 3 presents several approximation algorithms with performance guarantees. Sections 4 and 5 are dedicated to practical solution methods, namely fast greedy heuristics and a tabu search approach. The experimental behaviour and performances of these methods are analysed in Section 6.

2. The ADR problem

2.1. Formalization

We focus on the following problem: given a simple demand graph, we want to partition it into sub-graphs (rings) at minimum cost. Every demand must be satisfied (*i.e.*: the edge representing this demand must belong to a sub-graph representing a ring); a demand cannot be split but a node can be shared by several rings. A fixed cost r is imposed for creating a ring and a cost l is incurred for every node linked to a ring. We call a *r-link* (for "link to a ring") such a connection of a node to a ring. Moreover, each ring capacity is a constant C.

Formally, this leads to the following decision problem:

Assignment of Demands to Rings (ADR):
Instance: a simple graph $G = (V, E)$, a demand per edge: $d_e \in \mathbb{N}, \forall e \in E$, a capacity $C \in \mathbb{N}$, costs $r \in \mathbb{N}$ and $l \in \mathbb{N}$, a bound $B \in \mathbb{N}$;
Question: is there a partition of the edges of G into R sub-graphs $G_i = (V_i, E_i)$ such that: $\forall i \ : \sum_{e \in E_i} d_e \leq C$ and $l \sum_{i=1}^{R} |V_i| + rR \leq B$?

We shall also deal with a special case that we call ADRL (L means that only links have a cost) where demands are unitary and only the cost of linking a node to a ring is considered, *i.e.*: $r = 0, l = 1$. In the literature, this particular case is also called C-Edge-Partitioning Problem [12].

Let us point out that we only deal with the partition of the demand graph. The best way of connecting nodes within a ring is not considered here.

2.2. Complexity

Table 1 sums up complexity results for the ADR problem and some subcases. Most of these results are straightforward.

The polynomiality of the case $C = 2$ is derived from an algorithm by Masuyama and Ibaraki [20] that finds an optimal decomposition of a graph into k-chains, for any positive k.

TABLE 1. Complexity results for ADR (m is the number of demands).

Restrictions					Complexity	
C	r	l	demands	graph		
$C = 2$	-	-	-	-	polynomial - $\mathcal{O}(m)$	[2, 12]
$C = 3$	-	-	unitary	grid	polynomial - $\mathcal{O}(m)$	[19]
$C = 3$	-	-	unitary	$K_{n,p}$	polynomial - $\mathcal{O}(m)$	[19]
$C = 3$	-	-	unitary	tree	polynomial - $\mathcal{O}(m)$	[19]
C even	-	-	unitary	$K_{2,p}$	polynomial - $\mathcal{O}(m)$	[19]
$C \geq 3$	-	$l = 0$	-	-	$\mathcal{NP}$-complete (bin-packing)	[9]
$C \geq 3$	$r = 0$	-	unitary	-	strongly $\mathcal{NP}$-complete	[2, 12]

The $\mathcal{NP}$-completeness of the ADRL case when $C = 3$ is given by a reduction from EPT (Edge Partition into Triangles [14]).

3. Approximations

In this section, we focus on the ADRL subcase, *i.e.* all demands are equal to 1, and the cost r is zero. With these assumptions we have the following lower bound on the value of a solution:

Proposition 3.1 (see also Goldschmidt *et al.* [12])**.** *Let $G = (V, E)$ be a simple graph and S be a solution of ADRL for G. If L is the total number of r-links in S, then:*

$$L \geq \left\lceil \frac{1 + \sqrt{1 + 8C}}{2} \right\rceil \frac{|E|}{C} \geq |E| \sqrt{\frac{2}{C}}.$$

Proof. Let $\mathcal{L}_d$ be the minimum number of r-links (nodes) of a ring satisfying d demands. That is, $\mathcal{L}_d$ is the number of nodes of the smallest clique with at least d edges, so that

$$\mathcal{L}_d \geq \min \left\{ v \,:\, v \geq 0, \frac{v(v-1)}{2} \geq d \right\} \geq \left\lceil \frac{1 + \sqrt{1 + 8d}}{2} \right\rceil.$$

A solution of ADRL for a capacity C is a partition of the demand-graph into subgraphs of at most C edges and the cost per edge of a ring of x demands is at least $\frac{\mathcal{L}_x}{x}$. Moreover the function $x \mapsto \frac{\mathcal{L}_x}{x}$ is a decreasing function. Therefore, the cost per edge is at least $\frac{\mathcal{L}_C}{C}$ and a solution costs at least $\frac{\mathcal{L}_C}{C}|E|$.

The second lower bound is a trivial approximation of the first one and is asymptotically equivalent to it. □

The above bound is very pessimistic for most demand-graphs. However, it is tight if the graph can be partitioned into C-cliques.

This bound is the basis to establish the performance guarantees of some algorithms. The first one consists of using the Masuyama and Ibaraki decomposition

of a graph into 2-chains [20]. It is an exact algorithm for $C = 2$; for other cases, the following straightforward proposition holds:

Proposition 3.2. *For every connected demand graph and for every capacity $C \geq 3$, Masuyama and Ibaraki's algorithm for partitioning the edges of a graph into 2-chains yields a linear-time α-approximation for the ADRL problem, where:*

$$\alpha = \frac{3C}{2\left\lfloor \frac{1+\sqrt{1+8C}}{2} \right\rfloor} \leq \frac{3}{2}\sqrt{\frac{C}{2}}.$$

In order to improve the performance, one can think of covering the graph with trees larger than 2-chains. There exists an exact algorithm for solving ADRL on a tree with capacity $C = 3$ [19]. The solutions returned by such an algorithm have at most $\frac{3m+1}{2}$ r-links and they can be used as approximations for any connected demand graph and any capacity $C \geq 3$. The result is a linear-time $\left(\frac{3}{2} + \frac{1}{2m}\right)\sqrt{\frac{C}{2}}$-approximation. Even if the guarantee is similar, in practice this leads to better results than considering only 2-chains.

This approach of covering the demand-graph by trees has been extended by Goldschmidt *et al.* [12] to trees with $\frac{C}{2}$ to C edges. They proved the following result:

Proposition 3.3 (Goldschmidt *et al.* [12])**.** *There exists an algorithm that partitions a graph into trees of $\frac{C}{2}$ to C edges, and this gives a linear-time α-approximation for the ADRL problem, with:*

$$\alpha = \sqrt{\frac{C}{2}} + \sqrt{\frac{2}{C}}.$$

Here is another approximation algorithm, based on a completely different approach even though it also produces trees of at most C edges. It is easy to see that splitting an Eulerian path – assuming there is one – into subpaths of length at most C provides a feasible solution for ADRL with cost at most $|E| + \left\lceil \frac{|E|}{C} \right\rceil$. If the graph is not Eulerian, the same method holds, but we have to consider some additional splits: when an odd-degree node is reached, there may be no more unsatisfied demands to leave it. Every time this happens, a new ring must be started, thus implying an additional cost of 1; however, if we take care to start with an odd-degree node at the beginning of a ring or after a split, the number of additional splits cannot exceed $\frac{\delta}{2}$, where δ is the number of odd-degree nodes in the graph. Altogether, the maximal cost is $|E| + \left\lceil \frac{|E|}{C} \right\rceil + \frac{\delta}{2}$ and we get the following proposition:

Proposition 3.4. *The heuristic based on the search for an Eulerian path is a linear-time α-approximation for the ADRL problem, with:*

$$\alpha = \left(1 + \frac{1}{C} + \frac{1}{|E|C} + \frac{\delta}{2|E|} - \frac{1}{|E|}\right)\sqrt{\frac{C}{2}}$$

where δ is the number of odd-degree nodes.

This last algorithm has a better guarantee on Eulerian graphs, or when $C \leq \frac{|E|-1}{\sqrt{2|E|}-1}$, than the algorithm by Goldschmidt *et al.* [12].

All those approximations build trees and their performance ratio is indeed the ratio between the number of nodes per edge for a tree and for a C-clique, which is very pessimistic.

Another completely different approach is to consider the ADRL problem as a Set-Covering problem. Then the well-known Greedy Algorithm for set-covering can be applied. This algorithm is a $\ln(C)$-approximation in the general case [4] and a $\ln\left(\sqrt{\frac{C}{2}}\right)$-approximation for ADRL [3]. This algorithm, however, is based on the search for a minimum-cost ring at each iteration, which is a $\mathcal{NP}$-hard problem if the capacity C is not fixed.

4. Greedy algorithms

We now turn to a practical efficient solution of the ADR problem with no restriction. Two main approaches are considered: first we focus on the number of rings, second on the number of r-links. An experimental comparison of these heuristics can be found in Section 6.

4.1. Adaptation of the FFD algorithm

One can remark that, when the cost for a r-link is null ($l = 0$, and approximately when $l \ll r$), the structure of the demand-graph does not matter anymore and ADR is a Bin-Packing problem [9]. It is therefore natural to look for an adaptation of the well-known and efficient First-Fit Decreasing (FFD) algorithm[2] [15], which provides a guarantee of creating no more than $\frac{11}{9}R^* + 4$ rings, where R^* is the minimum number of rings in a feasible solution. FFD is thus a $\frac{11}{9}$-approximation algorithm for the case: $r = 1, l = 0$.

In order to integrate the cost of a r-link, several alternatives have been tried. First, we refine the sorting of the demands: among the set of equal-value demands, we sort them according to their endpoints so that adjacent demands are more likely to be considered successively and put into the same ring. The second alternative consists in taking some freedom with the FFD rules, and not putting a demand in the first ring that can contain it, but in the one which maximizes the number of common endpoints among such rings.

[2]The FFD algorithm is a mere application of the rules: "(1) sort the demands in non-increasing order and (2) assign each demand to the first ring that can contain it".

4.2. Looking for a small number of r-links

The second approach favors a small number of r-links.

The first algorithm consists in greedily adding the "best node" to the current ring, that is the node that allows to satisfy the greatest sum of demands, without exceeding the capacity constraint.

Another greedy algorithm consists in adding the "best demand" to the current ring, that is the greatest demand among those which have the largest number of nodes in the current ring, and which can be added without violating the capacity constraint. To speed up the algorithm, one may consider only the demands which are adjacent to the latest one. This simplification turns out, not only to be faster, but also to deliver slightly better solutions than the original method.

In addition to this algorithms, we also adapted the eulerian-path based approach (described in the previous section) to design another algorithm, which works for any demands and costs.

5. A tabu method for the ADR problem

In the previous section, we presented several greedy algorithms to build feasible solutions for the ADR problem. We now try to improve those initial solutions by means of a meta-heuristic. We propose here a tabu-search method (for description and details about tabu-search methods, see Glover and Laguna [10, 11]). The experimental study of the behaviour and the performances of this tabu-method can be found in Section 6.

Neighborhood: the neighborhood of a solution is merely defined as the set of all feasible solutions that can be reached by exchanging one demand of a ring with one (or zero) demand of another ring.

The first quality of this neighborhood is its size which permits quick iterations. There is however a major drawback since the number of rings cannot increase. We therefore miss some possibly optimal solutions, since only the most promising solutions are kept. Therefore, a diversification step is added to allow a modification of the number of rings.

Tabu list: let us suppose that we have just exchanged demand d_1 from ring R_1 and demand d_2 from ring R_2 (d_2 may be null, and thus only d_1 is added to R_2). We implemented three different tabu-lists.

The first rule forbids moving d_1 or d_2. The second one forbids exchanging d_1 for d_2 (no matter which rings they are in). The third one forbids putting d_1 back into R_1 or d_2 into R_2.

In all three cases, the movements are tabu for a fixed number n_t of iterations. Moreover, as those tabu-list management rules disallow more than the explored solutions, the aspiration mechanism [11] may be turned on and thus some unexplored but tabu solutions may be considered if they are better than the best one so far.

Diversification: we perform a diversification step, when no better solution has been found for too long, as follows. First, every ring is split into p parts, and each demand is assigned to a part randomly with equal probability. Then all those parts are greedily merged into new rings according to their common vertices. This procedure may increase the number of rings, but not much, and thus it balances the drawback of the neighborhood.

In our implementation of the tabu-search, every procedure is parameterized (including the greedy algorithm used for the initial solution) and every strategy described above can be used.

6. Empirical results

In this section, we present an experimental study of the solution methods described above. All procedures have been coded in C++ and run under Linux on a Pentium III with 256Mo RAM. The programs, as well as the tests, are available from the web (`http://www-leibniz.imag.fr/~lemaire/`).

6.1. The data

The algorithms have been tested on several kinds of generated instances.

The "Gnp" and "geometric" instances are taken from Goldschmidt *et al.* [13] and tend to represent real instances. These graphs are parameterized by their number n of nodes; every node is given a weight (1 with probability 0.8, and 2 with probability 0.2). An edge that connects two nodes of weights w_i, w_j carries a demand of $w_i + w_j - 1$. In a "Gnp" graph, two nodes are connected with a given probability p, independently of the other pairs of nodes. In a "geometric" graph, two nodes, uniformly distributed on a square, are connected if their Euclidian distance is less than a given bound L. Parameters p and L have been chosen so that graphs have a total demand of about 100. Ring capacity C for those graphs was chosen to be equal to 40 and 60.

In addition to those realistic graphs, we generated "purely" random graphs. Those "rand" graphs are parameterized by n, p, and d: they have n nodes, an edge has probability p to exist and, if it exists, it carries a demand uniformly distributed between 1 and d. Ring capacity C for those graphs was chosen to be equal to 25, 40 and 60.

We also used simple graphs, such as grids, bipartite complete graphs, and trees, with unitary demands and a ring capacity C of 3.

For every instance, we considered the different combinations of the costs r and l, taken in 0, 1 and 10.

A large number of other tests has been performed to find good tunnings for the algorithms. They do not appear here.

6.2. The results

Many tests have been carried out in order to study the behavior of our tabu search method. What follows is a representative synthesis of all the gathered results.

General behavior: for almost every instance, the behavior is the same: most of the gain is obtained during the first iterations (but with some plateaux a pure-descent heuristic would not go through). After this relatively short phase, there are few new best solutions found. However, this classical tabu-search behavior is dependent on the settings.

Some good settings: for nearly all instances, the third management rule for the tabu-list leads to the best results. As to the tabu list length, a value of $n_t = 3\sqrt{|E|}$, where $|E|$ is the number of demands, yields good results, which may be slightly improved by increasing n_t slowly during the iterations.

The diversification of the solutions does deserve further work. Our method performs poorly if rings are split into only two parts. However, gains are obtained with 3 or 4 parts. Moreover, at least with unitary demands, to force a diversification at the beginning of the search is beneficial most of the time. For variable demands, it seems to be worthwhile to develop further refinements.

The initial solution does not seem to have a big influence on the final solution, as long as it is not too bad. That is, we can use any of our greedy algorithms.

Once we knew some good settings for the tabu-search, we have tested and compared the different heuristics: the FFD-based algorithm (FFD), the best-node algorithm (BN), the Eulerian-path based algorithm (EUL), the best-demand algorithm (BD), and the tabu-search (TS) on the instances described above.

The settings for TS were as follows. We used FFD as initial algorithm; we used the third management rule for the tabu-list and this list was set to a constant length of 15; we forced diversification after 500 unsuccessful iterations and then the rings were split in 3 parts. These settings were chosen since they appear to be good instance-independent settings for TS (see above).

From our tests, it appears that the cost r, for creating a ring, does not have a big influence on the behavior and on the relative performance of the algorithms (both the greedy algorithms and the tabu search). Indeed, the number of rings hardly changes from one solution to the other; hence the gain is made on the number of r-links. Apart from this cost, the other parameters of an instance are greatly influent, in particular the type of the graph. The main results are gathered in Tables 2–4, but the comments include additional results.

On simple instances, the algorithms perform well (*cf.* Tab. 2). On grids and bipartite complete graphs of unitary demands, EUL is always very close to the optimal value, with an average distance of 0.26%. BN is by far the worst on this kind of instances. On trees, BD takes the advantage over the other algorithms whereas FFD performs poorly.

TABLE 2. Performances of greedy and TS algorithms on simple instances with $r = l$. Gap is the gap to optimum, if known (grids and $K_{n,p}$), or to a good lower bound (trees).

graph	$\|E\|$	**algorithm**	**gap** (%)
grids $n \times p$ $C = 3$	$n = 3, 5, 10, 15, 20, 25, 50$ $p = 3, 5, 10, 15$	FFD	9.24
		BN	11.49
		EUL	0.52
		BD	3.51
		FFD+TS	0.00
$K_{n,p}$ $C = 3$	$n = 3, 5, 10, 15, 20, 25, 50$ $p = 3, 5, 10, 15$	FFD	2.87
		BN	14.07
		EUL	0.00
		BD	1.03
		FFD+TS	0.00
trees of n nodes $C = 3$	$n = 3, 5, 10, 15, 20, 25, 50$	FFD	11.24
		BN	8.75
		EUL	7.10
		BD	4.84
		FFD+TS	1.69

Even without a good initialization, TS finds optimal solutions for all cases of grids and $K_{n,p}$ in less than 500 iterations (in most cases in less than 100 iterations). On trees, it permits to reach quickly near-optimal or even optimal solutions.

On most realistic graphs, FFD is the best greedy algorithm and EUL the worst. However, the solutions given by the different greedy heuristics do not differ much, usually by less than 10%. On random graphs, FFD and EUL exchange their role: FFD is the worst, EUL the best, and this tendency increases as the graph becomes bigger and/or more dense (that is: with more edges for the same number of nodes). Indeed, for very big and dense graphs, EUL is by far the best greedy algorithm and FFD the worst (*cf.* Tab. 3).

TS permits a gain of at least about 10%, and generally more, over the best greedy solutions, with only few iterations (less than 500). Going on with TS for a longer time does not provide much improvement on realistic instances, but allows few-percent additional gains on random graphs (*cf.* Tab. 4).

The fact that TS mainly fails to find improved solutions after the first 500 iterations or so reveals the lack of efficiency of the diversification phase.

All the greedy heuristics are very quick and they all have good performances. Indeed they all outperform the best guarantees provided by the algorithms of Section 3 by about 60% on realistic graphs, and by more than 100% on big random graphs (*cf.* Tab. 3).

The two best greedy algorithms appear to be FFD and EUL, depending on the type of graph. An advantage of using these two heuristics together is that they are really complementary: when one performs poorly, the other one does very good.

TABLE 3. Performances of the greedy algorithms on realistic and random instances ($r = 0, l = 1$). RP is the ratio with the best greedy solution; BG is the ratio with the best guarantee from Section 3 [3].

instance	**algorithm**	**CPU-time** (s)	**RP**	**BG**
6 "gnp" graphs $n = 15, p = 95$ $C = 40$	FFD	0.000	1.00	0.40
	BN	0.003	1.06	0.36
	EUL	0.000	1.06	0.37
	BD	0.002	1.03	0.34
6 "geometric" graphs $n = 15, d = 8$ $C = 40$	FFD	0.000	1.01	0.38
	BN	0.000	1.20	0.37
	EUL	0.002	1.09	0.38
	BD	0.002	1.14	0.35
6 "rand" graphs $n = 15, p = 85, d = 5$ $C = 40$	FFD	0.002	1.00	0.37
	BN	0.000	1.06	0.39
	EUL	0.002	1.04	0.38
	BD	0.002	1.04	0.38
6 "rand" graphs $n = 75, p = 50, d = 5$ $C = 25$	FF D	0.010	1.35	1.18
	BN	0.020	1.19	1.04
	EUL	0.000	1.00	0.87
	BD	0.050	1.22	1.07

TABLE 4. Performances of TS on instances of Table 3, for 1500 and 5000 iterations. The gain is (1) over the initial solution and (2) over the best greedy solution; BI is the iteration when the best solution was found.

instance	**♯iter**	**CPU-time** (s)	**gain (%)** (1)	(2)	**BI**
6 "gnp" graphs $n = 15, p = 95$ $C = 40$	1500	1.6	19.67	19.34	377
	5000	5.3	19.67	19.34	454
6 "geometric" graphs $n = 15, d = 8$ $C = 40$	1500	2.0	8.91	8.20	266
	5000	6.4	8.91	8.20	301
6 "rand" graphs $n = 15, p = 85, d = 5$ $C = 40$	1500	1.0	42.12	20.96	846
	5000	3.2	43.57	22.95	2673
6 "rand" graphs $n = 75, p = 50, d = 5$ $C = 25$	1500	188	36.67	1.15	1491
	5000	629	39.92	3.52	4428

[3] Choosing $r = 0$ unables this comparison; however this is not a limitation, since r do not seem to have a big impact on the solutions. Furthermore, the guarantees are computed using a lower bound instead of the optimal solution, and are thus optimistic.

TS, as it is implemented and with standard instance-independent settings, permits to achieve improvements of more than 10% on realistic instances in little time.

7. Conclusions and perspectives

In this paper, we have presented and formalized the ADR problem, which turned out to be $\mathcal{NP}$-hard. Several fast approximation algorithms for the particular case of ADRL have been recalled or introduced. An interesting point would be to find better bounds than the pessimistic ones proposed.

For practical solutions, fast and efficient greedy algorithms have been proposed, together with a tabu-search approach to improve the solutions they deliver. The results are good and the approach can be considered to be successful. However, the method leaves room for improvements. In particular, the diversification step deserves more attention; a dynamic management of the tabu list is also very promising according to some preliminary tests.

References

[1] R. Aringhieri and M. Dell'Amico, *A Variable-Neighborhood Variable-Objective Tabu Search Algorithm for the SONET Ring Assignment with Capacity Constraints*. DISMI, Università di Modena e Reggio Emilia, 10 (2001).

[2] N. Brauner, Y. Crama, P. Lemaire and C. Wynants, *Complexité et approximation pour la conception de réseaux SONET/SDH*. Technical Report 61, Les Cahiers du Laboratoire Leibniz-IMAG, (October 2002). `http://www-leibniz.imag.fr/LesCahiers/`

[3] N. Brauner and P. Lemaire, *A Set-Covering Approach for SONET Network Design*. Technical Report 62, Les Cahiers du Laboratoire Leibniz-IMAG (October 2002). `http://www-leibniz.imag.fr/LesCahiers/`

[4] V. Chvátal, A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.* **4** (1979) 233-235.

[5] ETSI. ETSI – Telecom Standards. `http://www.etsi.org`

[6] ETSI. Transmission and Multiplexing (TM); Synchronous Digital Hierarchy (SDH); Network protection schemes; Rings and other schemes. Technical specification (November 1997). ref: TS 101 010 v1.1.1.

[7] ETSI. Transmission and Multiplexing (TM); Synchronous Digital Hierarchy (SDH); Network protection schemes; Types and characteristics. Technical specification (November 1997). ref: TS 101 009 v1.1.1.

[8] B. Fortz, P. Soriano and C. Wynants, A Tabu Search Algorithm for Self-Healing Ring Network Design. *Eur. J. Oper. Res.* **151** (2003).

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability (A Guide to the Theory of NP-Completeness)*. W.H. Freeman And Company (1979).

[10] F. Glover and M. Laguna, *Modern Heuristic Techniques for Combinatorial Problems*, *Chapter 3: Tabu Search*. C.R. Reeves, Blackwell Scientific Publications edition (1993).

[11] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, London (1997).

[12] O. Goldschmidt, D.S. Hochbaum, A. Levin and E.V. Olinick, The SONET Edge-Partition Problem. *Networks* **41** (2003) 13-23.

[13] O. Goldschmidt, A. Laugier and E.V. Olinick, SONET/SDH Ring Assignment with Capacity Constraints. *Discrete Appl. Math.* **129** (2003) 99-128.

[14] I. Holyer, The NP-completeness of some edge-partition problems. *SIAM J. Comput.* **4** (1981) 713-717.
[15] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey and R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.* **3** (1974) 299-325.
[16] S. Khanna, A Polynomial Time Approximation Scheme for the SONET Ring Loading Problem. *Bell Labs Technical Journal* (1997) 36-41.
[17] M. Laguna, Clustering for the Design of SONET Rings in Interoffice Telecommunications. *Manage. Sci.* **40** (1994) 1533-1541.
[18] Y. Lee, H.D. Sherali, J. Han and S. Kim, A branch-and-cut algorithm for solving an intraring synchronous optical network design problem. *Networks* **35** (2000) 223-232.
[19] P. Lemaire, Optimisation de réseaux SONET/SDH : éléments théoriques et résolution pratique (juin 2001). Mémoire de DEA.
[20] S. Masuyama and T. Ibaraki, Chain Packing in Graphs. *Algorithmica* **6** (1991) 826-839.
[21] D. Mili, Self-Healing Ring Architectures for SONET Network Applications. `http://www.doc.ic.ac.uk/∼nd/surprise_96/journal/vol2/dm9/article2.html`
[22] A. Schrijver, P. Seymour and P. Winkler, The Ring Loading Problem. *SIAM J. Discrete Math.* **11** (1998) 1–14.
[23] A. Sutter, F. Vanderbeck and L.A. Wolsey, Optimal Placement of Add/Drop Multiplexers: Heuristic and Exact Algorithms. *Oper. Res.* **46** (1998) 719-728.
[24] T.-H. Wu, *Fiber Network Service Survivability.* Artech House, Inc. (1992).
[25] The SONET Home Page. `http://www.sonet.com`