Solution of the SONET Ring Assignment problem with Capacity Constraints

R. Aringhieri and M. Dell'Amico.

Solution of the SONET Ring Assignment Problem with capacity constraints.

In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, pages 93-116. Kluwer Academic Publisher, 2005.

DOI: 10.1007/0-387-23667-8_4

Chapter 1

SOLUTION OF THE SONET RING ASSIGNMENT PROBLEM WITH CAPACITY CONSTRAINTS

Roberto Aringhieri
DISMI - University of Modena and Reggio Emilia
Viale Allegri 13, 42100 Reggio Emilia - Italy
aringhieri.roberto@unimore.it

Mauro Dell'Amico
DISMI - University of Modena and Reggio Emilia
Viale Allegri 13, 42100 Reggio Emilia - Italy
dellamico@unimore.it

Abstract Synchronous Optical Network (SONET) in North America and Synchronous Digital Hierarchy (SDH) in Europe and Japan are the current transmission and multiplexing standards for high speed signals within the carrier infrastructure. The typical topology of a SONET network is a collection of rings connecting all the customer sites. We deal with a design problem in which each customer has to be assigned to exactly one ring and these rings have to be connected through a single federal ring. A capacity constraint on each ring is also imposed. The problem is to find a feasible assignment of the customers minimizing the total number of rings used. A Tabu Search method is proposed to solve the problem. The key elements are the use of a variable objective function and the strategic use of two neighborhoods. We have also implemented other techniques such as Path Relinking, eXploring Tabu Search and a Scatter Search. Extensive computational experiments have been done using two sets of benchmark instances. The performances of the proposed algorithms have also been compared with those of three multistart algorithms involving greedy methods previously proposed for the problem, and of the CPLEX solver. The computational experiments show the effectiveness of the proposed Tabu Search.

Keywords: Metaheuristics, SONET, Graph Partitioning

1. Introduction

Synchronous Optical Network (SONET) in North America and Synchronous Digital Hierarchy (SDH) in Europe and Japan are the current transmission and multiplexing standards for high speed signals within the carrier infrastructure.

A typical topology of a SONET network is a collection of *local rings* or simply *rings* directly connecting a subset of customer sites or nodes. Each node sends, receives and relays messages through a device called add-drop-multiplexer (ADM). In bidirectional rings the traffic between two nodes can be sent clockwise or counterclockwise, and the volume of traffic is limited by the ring capacity, which is equal in both directions. The capacity of the bidirectional ring has to accommodate the sum of bandwidth requests between all pairs of nodes connected by the ring. Finally, all rings are connected together by a special ring called *federal ring* through an expensive device, the digital cross connect (DXC), which joins each ring to the federal ring.

With this topology, the traffic between two customers may use up to three rings. A different design choice imposes that the traffic between two nodes is always transmitted using only one ring. In this case, a customer may be assigned to several rings. A design problem requiring this topology has been considered in Laguna, 1994, where a mathematical model and Tabu Search algorithm are presented.

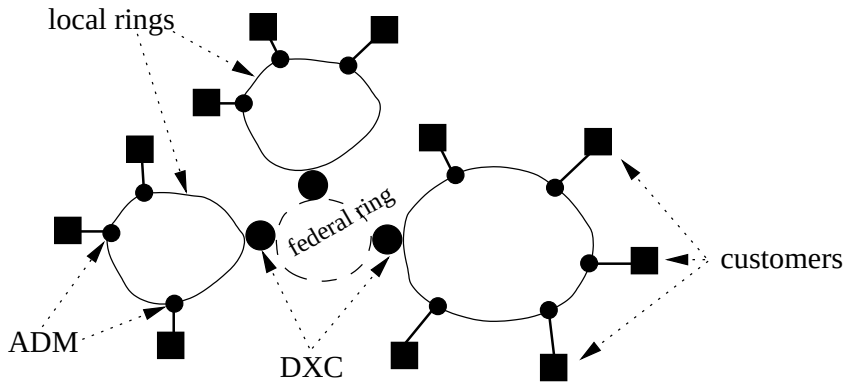

Figure 1.1. A SONET network

In this paper we deal with a basic design problem using the first topology and requiring that: i) each customer has to be assigned to exactly one local ring; and ii) the maximum capacity of each ring is bound by a common value. We want to minimize the number of local rings, that is the cost of DXCs installed. This problem is called SONET Ring Assign-

ment Problem (SRAP) with capacity constraints. An example of SONET network is given in Figure 1.1.

Fixing the number k of rings required, we derive a parametric version of the problem called k-SRAP. Goldschmidt, Laugier and Olinick (Goldschmidt et al., 2001) have shown that SRAP is $\mathcal{NP}$-complete by reducing the 3-Regular Graph Bisection Problem to a special case of SRAP and proposed three randomized greedy algorithms to solve it.

Mathematical formulations for SRAP are presented in Section 2. In Section 3 we briefly describe the greedy algorithms proposed in Goldschmidt et al., 2001. In Section 4 we introduce a basic Tabu Search (BTS) for solving the problem. BTS is then improved implementing several intensification and diversification strategies. The strategic use of two neighborhoods leads to a Tabu Search with Strategic Oscillation (TSSO) which is reported in Section 5. Path Relinking, eXploring Tabu Search and Scatter Search approaches are reported in Section 6. Extensive computational results are reported in Section 7, and some conclusions are presented in Section 8.

2. Mathematical Formulations

We introduce the following notation: let n be the number of nodes, B the common capacity bound, d_{uv} the symmetric traffic demand between the pair of nodes u and v ($u, v = 1, \ldots, n$, $u \neq v$) and $W_u = \sum_{v \neq u} d_{uv}$ the total traffic on node u ($u = 1, \ldots, n$). Let us define the binary variables: $p_{uvr} = 1$ ($u, v = 1, \ldots, n$, $u \neq v$, $r = 1, \ldots, n$) if nodes u and v belong both to ring r, $p_{uvr} = 0$ otherwise; $x_{ur} = 1$ ($u = 1, \ldots, n$, $r = 1, \ldots, n$) if node i is assigned to ring r, $x_{ur} = 0$ otherwise; $y_r = 1$ ($r = 1, \ldots, n$) if ring r is used, $y_r = 0$ otherwise. A model for SRAP derived from that

presented in Goldschmidt et al., 2001, is the following:

$$\text{SRAP} : \min \quad \sum_{r=1}^{n} y_r$$

$$\text{s.t.} \quad \sum_{u=1}^{n} x_{ur} W_u - \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} p_{uvr} d_{uv} \leq B,\ r = 1, .., n \tag{1.1}$$

$$\frac{1}{2} \sum_{u=1}^{n} W_u - \sum_{r=1}^{n} \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} p_{uvr} d_{uv} \leq B \tag{1.2}$$

$$\sum_{r=1}^{n} x_{ur} = 1, \quad u = 1, .., n \tag{1.3}$$

$$p_{uvr} \leq x_{ur}, \quad r = 1, .., n,\ \{u, v = 1, .., n | u < v\} \tag{1.4}$$

$$p_{uvr} \leq x_{vr}, \quad r = 1, .., n,\ \{u, v = 1, .., n | u < v\} \tag{1.5}$$

$$x_{ur} \leq y_r, \quad u, r = 1, .., n, \tag{1.6}$$

$$x_{ur}, p_{uvr} \in \{0, 1\}, \quad r = 1, .., n, \quad u, v = 1, .., n$$

Constraints (1.1) and (1.2) assure, respectively, that the bounds on the total traffic through each ring and through the federal ring are satisfied. Constraints (1.3) assure that each node is assigned to exactly one ring. Constraints (1.4)-(1.6) impose the congruence of the sets of variables. The objective function is to minimize the number of rings.

It is worth noting that this formulation is not useful for solving the problem with a commercial and general purpose code (e.g. CPLEX), due to the enormous computing times. A model with only constraints (1.1)-(1.5) and no objective function can be used for checking the feasibility of an instance. We adopted this method in Section 7.

A graph theory model for SRAP can be defined as follows. Let $G = (V, E)$ be the graph representing the SONET network (V is the set of customer's nodes) and d_{ij} the cost of edge $(i, j) \in E$. The problem of finding a solution for SRAP corresponds to finding a partition $V_1, V_2, \ldots V_k$ of V into disjoint sets such that k is minimized and

$$\sum_{u \in V_i} \sum_{v \in V, v \neq u} d_{uv} \leq B, \quad i = 1, \ldots, k \tag{1.7}$$

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{u \in V_i} \sum_{v \in V_j} d_{uv} \leq B \tag{1.8}$$

where (1.7) and (1.8) assure, respectively, that the bounds on the total traffic through each ring and through the federal ring are satisfied.

3. Previous work

Goldschmidt, Laugier and Olinick (Goldschmidt et al., 2001) have proposed three different greedy heuristics for solving SRAP: the *edge-based*, the *cut-based* and the *node-based* heuristics. Both the edge-based and the cut-based heuristics start their computation assigning each node to a different ring, and, at each iteration two different rings are merged if the resulting ring is feasible. In the edge-based heuristic, the edges are ordered by non decreasing weight and the pair of rings connected by the maximum weight edge is chosen. In the cut-based heuristic the pair of rings sharing the maximum traffic is chosen. Both the edge-based and the cut-based heuristics start their computation assigning each node to a different ring.

Finally, the node-based heuristic receives a tentative number of rings k as input, then assigns to each of the k rings a node randomly chosen. At each iteration the heuristic selects the ring with the the largest unused capacity, say r, then, the unassigned node with largest traffic toward ring r is chosen and added to the ring. The node-based heuristic is repeated ten times and every time a feasible solution is computed the value of k is decreased by one.

These heuristics are applied by the authors to a set of benchmark instances. A feasible $\bar{k}$ is computed by running the edge-based and cut-based heuristics and given as input to the node-based heuristic. Note that both the edge-based and cut-based heuristics may have different behavior if different tie break rules are used. Therefore the authors propose to repeat these procedures ten times and to choose randomly how to break ties. When the heuristic solution is feasible with a value greater than the simple lower bound

$$k_{lb} = \left\lceil \frac{\sum_u \sum_{v \neq u} d_{uv}}{2B} \right\rceil, \tag{1.9}$$

a tentative to prove the optimality of the heuristic solution is performed by means of CPLEX.

4. Basic Tabu Search

We propose a solution algorithm based on a Tabu Search (TS) scheme which explores the space of the possible partitions of graph G looking for a partition associated with a feasible solution of SRAP with a minimum number of rings.

In order to simplify the presentation we introduce the following notation. Let r be a ring index. The *traffic values* with respect to ring r

are:

$$I_r = \sum_{u,v \in r, u \neq v} d_{uv} \qquad \text{(total internal traffic)}$$

$$E_r = \sum_{u \in r, v \notin r} d_{uv} \qquad \text{(total external traffic)}$$

$$T_r = I_r + E_r \qquad \text{(total traffic on the ring } r\text{)}$$

Note that a ring r is feasible if $T_r \leq B$. Given a node u define:

$$W_{u,r} = \sum_{v \in r, v \neq u} d_{uv} \quad \text{(total traffic toward node } u \text{ from the ring } r\text{)}$$

$$W_u = \sum_{v \neq u} d_{uv} \qquad \text{(total traffic toward node } u\text{)}$$

Observe that

$$W_u = \sum_r W_{u,r}.$$

Finally define:

$$F = \frac{\sum_r E_r}{2} \qquad \text{(total traffic through the federal ring)}$$

$$BN = \max(\, F,\, \max_r T_r) \quad \text{(network bottleneck)} \tag{1.10}$$

4.1. Neighborhood N_1 and traffic value update

The basic Tabu Search (BTS) uses a *node improvement* neighborhood, say N_1, which, starting from a given solution (feasible or unfeasible), moves in turn each node from its ring to a different one or to a new ring, such that the resulting ring (existing or new) is feasible.

In Figure 1.2 an example of move generated by N_1 is depicted: the node u is moved from the ring s to the ring t; the dotted region represents the federal ring and r is a generic ring such that $r \neq s, t$. We observe that moving u from s to t modifies the following traffic values: T_s, T_t, F and possibly BN.

In Figure 1.3 we depict the traffic values before and after moving node u from ring s to ring t.

Note that traffic T_s decreases by

$$W_{u,t} + \sum_{r \neq s,t} W_{u,r} = W_u - W_{u,s}.$$

Analogously traffic T_t increases by

$$W_{u,s} + \sum_{r \neq s,t} W_{u,r} = W_u - W_{u,t}.$$

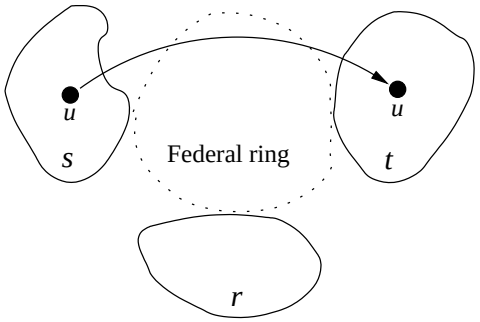

Figure 1.2. Neighborhood N_1: u is moved from s to t

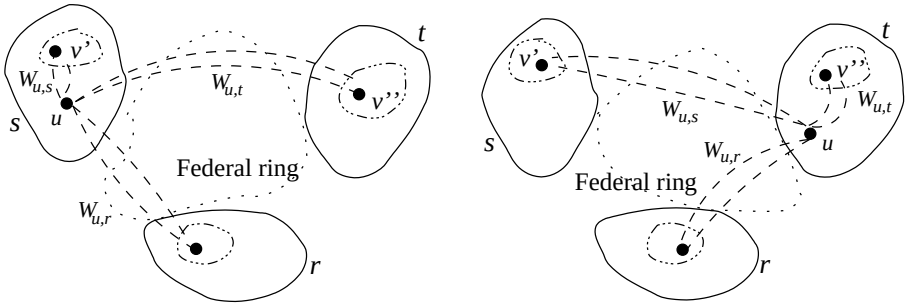

Figure 1.3. Neighborhood N_1: traffic values

Further note that the traffic F on the federal ring F varies by $W_{u,s} - W_{u,t}$.

Therefore the modified traffic values after moving u from s to t are:

$$\widehat{T}_s = T_s - W_u + W_{u,s} \tag{1.11}$$

$$\widehat{T}_t = T_t + W_u - W_{u,t} \tag{1.12}$$

$$\widehat{F} = F - W_{u,t} + W_{u,s} \tag{1.13}$$

where the hat symbol denotes the modified values.

Using the updated values (1.11)-(1.13) we can compute the new bottleneck and evaluate the new solution.

After selecting the best move in N_1, it is necessary to update the data structures. Considering again Figure 1.3, we observe that $W_{v',s}$ is decreased by $d_{uv'}$ while $W_{v'',t}$ is increased by $d_{uv''}$. Instead traffic $W_{v,r}$

with $v \neq u$ and $r \neq s, t$ does not change, hence

$$\widehat{W}_{v,r} = \begin{cases} W_{v,r} - d_{uv} & r = s, \quad v \in s \\ W_{v,r} + d_{uv} & r = t, \quad v \in t \\ W_{v,r} & \text{otherwise} \end{cases} . \tag{1.14}$$

4.2. Tabu lists

We have used two tabu lists to avoid visiting, for a certain period, already visited solutions.

The first list called *node-list* stores the nodes visited in the last iterations. A node contained in node-list is not taken into account when exploring the neighborhood.

The second list called *node-from-list* stores the moved nodes within the identifier of the original ring from which the node has been removed. During the exploration of the neighborhood we avoid inserting a node stored in this list into its original ring.

We maintain the length l_1 of node-list smaller than the length l_2 of node-from-list in order to fix a node for l_1 iterations (node-list) and then to forbid its return to the original ring for l_2 other iterations (node-from-list).

The list length varies during the search (see e.g. Dell'Amico and Trubian, 1998) between $\frac{1}{2}$ and $\frac{3}{2}$ of a given *tabu-tenure* value. We decrease l_1 and l_2 to intensify the search within promising regions, whilst we increase them to speed up the leaving from not promising regions. We define an *improving phase* a set of Δip consecutive iterations which lower the objective function. On the contrary, a *worsening phase* is a set of Δwp consecutive iterations such that the objective function value does not improve. The initial values of l_1 and l_2 are respectively equal to *tabu-tenure*$_1$ and *tabu-tenure*$_2$ and vary as follows:

$$l_i = \max(l_i - 1, \frac{1}{2} \textit{tabu-tenure}_i), \quad i = 1, 2 \tag{1.15}$$

after an improving phase and

$$l_i = \min(l_i + 1, \frac{3}{2} \textit{tabu-tenure}_i), \quad i = 1, 2 \tag{1.16}$$

after a worsening phase. On the basis of a set of preliminary experiments, we set the parameter values as reported in Table 1.1.

4.3. Objective functions

In order to drive the search toward feasible and good solutions we have to take into account both the number of rings k and the value of

Table 1.1. Tabu List: values of parameters controlling the length of tabu lists

values	l_1	l_2
tabu-tenure	5	10
Δip	5	5
Δwp	3	3

the bottleneck BN. We have defined the following tentative objective functions:

$$z_1 = k + \max\{0, BN - B\},$$
$$z_2 = z_1 + \begin{cases} k\,T_c & \text{if a new ring } c \text{ has been created} \\ 0 & \text{otherwise} \end{cases},$$
$$z_3 = kB + BN.$$

Function z_1 leads the search toward solutions with a small number of rings while penalizing unfeasible solutions. Function z_2 modifies z_1 by strongly penalizing moves creating a new ring. Finally, function z_3 leads the search toward solutions with a minimum value of k, and among these toward solutions with minimum bottleneck value..

We observed that the expected solution improvement depends on the feasibility status of the current solution. For example, if the current solution is feasible and a move creates a new feasible solution, then we are interested in minimizing k. On the contrary, if the current solution is unfeasible and a move creates a new unfeasible solution, then we would like to minimize BN. Therefore we have studied a further multiobjective function which drives the search taking into account these different cases. Denoting with $\widetilde{BN}$ the bottleneck values associated with unfeasible solutions, we define:

$$z_4 = \begin{cases} k\,B + BN & \text{if the move starts from a feasible solution} \\ & \text{and generates a feasible one} \\ (k+1)\widetilde{BN} & \text{if the move starts from a feasible solution} \\ & \text{and generates an unfeasible one} \\ k\,B & \text{if the move starts from an unfeasible solution} \\ & \text{and generates a feasible one} \\ n\,\widetilde{BN} & \text{if the move starts from an unfeasible solution} \\ & \text{and generates an unfeasible one} \end{cases}$$

Since for a feasible solution $k\,B + BN \leq (k+1)B$, $(k+1)B \leq (k+1)\widetilde{BN}$ and $k\,B \leq n\,\widetilde{BN}$, then z_4 drives the search from unfeasible solutions to feasible ones. Note that z_4 is a sort of extension of z_3.

4.4. Procedure BTS

After reading the input instance and initializing the main data structures, the Basic Tabu Search BTS starts a main cycle which is repeated until *MaxTime* seconds have elapsed.

At each iteration, the best move $Move$ is selected by the function **BestMove**. If $Move \neq \emptyset$, the current solution Sol is modified (**Update**($Move$, Sol)) and the best solution $Best$ is updated if $z(Sol) < z(Best)$.

The update phase concerns both the parameters controlling the length of tabu lists and the traffic values: the parameters are updated with respect to the objective function improvement ($z(Sol) < \bar{z}$) while the traffic values are updated according to the equations (1.11)-(1.14) by procedure **UpdateTrafficValues**.

The tabu lists are updated by the **UpdateTabuList** procedure: if $Move \neq \emptyset$, $Move$ is inserted in the node-list and node-from-list, otherwise a dummy move is inserted. Then, if $ip = \Delta ip$ or $wp = \Delta wp$, the length of the tabu lists are updated as in (1.15) and (1.16).

The pseudo-code of the BTS algorithm is as follows.

```
BTS( Instance, MaxTime ) {
  GetData( Instance );
  Init( Move, Sol, Best );
  Compute( W_{u,r}, W_u, T_r, F, BN );
  Elapsed := 0; Starting := Now( );
  while ( Elapsed ≤ MaxTime ) {
    Move := BestMove( Sol, N_1 )
    if ( Move ≠ ∅ ) {
      z̄ := z(Sol);
      Sol := Update( Move, Sol );
      if ( z(Sol) < z(Best) ) Best:=Sol;
      if ( z(Sol) < z̄ ) { ip := ip + 1; wp := 0; }
      else { wp := wp + 1; ip := 0; }
      UpdateTrafficValues( Move, W_{u,r}, T_r, F, BN );
    };
    UpdateTabuList( Move, ip, wp );
    Elapsed := Now( ) - Starting;
  };
  return( Best );
}.
```

At the end of the computation the best solution computed is returned as output of the procedure.

4.5. Complexity of neighborhood evaluation

The neighborhood evaluation is the most time consuming component of the whole algorithm. We observe that in the worst case each of the n nodes can be moved into $n-1$ rings. Since the computation of the best move requires the assessment of all the possible moves, the neighborhood evaluation of N_1 needs to generate $O(n^2)$ moves. We also observe that all the objective functions proposed in §4.3 require the computation of BN. As described in §4.1, moving u from s to t can increase or decrease the BN value.

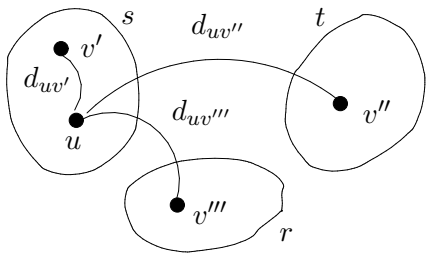

Figure 1.4. Neighborhood N_1: variation of BN value

We illustrate this case in Figure 4.5 where the node u, which is connected only with $v' \in s$, $v'' \in t$ and $v''' \in r$, is moved from s to t. Suppose that: $F = 80$, $T_t = 85$, $T_r = 100$, $T_s = BN = 110$, $d_{uv'} = 15$, $d_{uv''} = 5$ and $d_{uv'''} = 7$. Applying the equations (1.11)-(1.13), we obtain

$$\widehat{T}_s = T_s - (d_{uv''} + d_{uv'''}) = 98$$
$$\widehat{T}_t = T_t + (d_{uv''} + d_{uv'''}) = 97$$
$$\widehat{F} = F + (d_{uv'} - d_{uv''}) = 90$$

and T_r is not modified. The new value of BN is now equal to $T_r = 100$, hence it is decreased. On the other side, if $T_t = 100$ the new value of BN increases to 112.

As shown in the above example, the computation of BN involves not only $\widehat{T}_t$ and F, but also all the T_r values with $r \neq s, t$, hence an $O(n)$ time is required to compute BN after each node movement and the neighborhood evaluation is done in $O(n^3)$ in the worst case.

This complexity can be reduced to $O(n^2 \log n)$ maintaining a heap data structure to order the T_r values. Some special cases (e.g. when

$BN = T_t$) can be also considered to reduce the average computation time. However, in the practical application (see Section 7), the number of rings is small and no special implementation is required.

5. Tabu Search with Strategic Oscillation

The computational experiments performed (see Section 7, Table 1.4) show that procedure BTS is not very effective to solve SRAP hence we have tried to improve it by means of a simple variable neighborhood strategy consisting in a further neighborhood N_2 and a switching rule to substitute N_1 to N_2 and vice versa.

Employing only the neighborhood N_1, BTS often terminates its computation returning a solution containing k large rings and a single ring, say r, containing few nodes. In these cases, we expect that a local optimal solution with k rings can be obtained spreading the nodes belonging to r over the remaining k rings. However BTS is not able to find this solution since moving all but one nodes belonging to r increases the bottleneck while it mantains the same number of rings.

The introduction of the *empty minimum cardinality ring* neighborhood, say N_2, is aimed at dealing with this kind of solution: a node belonging to r is compulsively moved to one of the remaining rings without taking into account the feasibility of the new solution.

The switching of N_1 to N_2 is controlled by two conditions:

condition C_1: a number i_{ni} of consecutive not improving iterations has been performed;

condition C_2: the current solution is feasible.

At each iteration we check condition C_1, C_2 or both (see Section 7 for implementation details) and, if necessary, we transform the current solution into a new one through neighborhood N_2 (**NewSol**()).

We insist with neighborhood N_2 until the minimum cardinality ring has a single node. At this point, N_2 is applied once again and we return to N_1 for the next iterations.

This is a particular implementation of the strategic oscillation technique in which we have only two neighborhoods and the second one is applied for a fixed number of iterations, namely the cardinality of the minimum cardinality ring.

A possible pseudo-code of our TS with Strategic Oscillation (TSSO) algorithm is as follows.

```
TSSO( Instance, MaxTime ) {
  GetData( Instance );
  Init( Move, Sol, Best );
  Compute( W_{u,r}, W_u, T_r, F, BN );
  Elapsed := 0; Starting := Now( );
  switch := FALSE;
  while ( Elapsed ≤ MaxTime ) {
    Move := BestMove( Sol, N_1 )
    if ( Move ≠ ∅ ) {
      z̄ := z(Sol);
      Sol := Move( Move, Sol );
      if ( z(Sol) < z(Best) ) Best:=Sol;
      if ( z(Sol) < z̄ ) {
        ip := ip + 1; wp := 0; i_ni := 0; }
      else {
        wp := wp + 1; ip := 0; i_ni := i_ni + 1; }
      UpdateTrafficValues( Move, T_{u,r}, T_r, F, BN );
    };
    UpdateTabuList( Move, ip, wp, i_ni );
    if ( Check( C_1, C_2 ) )
      Sol := NewSol( Sol, N_2 );
    Elapsed := Now( ) - Starting;
  };
  return( Best );
}.
```

Note that procedure **NewSol**() consists of a number of moves with neighborhood N_2 which spread the nodes belonging to the minimum cardinality ring to the remaining rings.

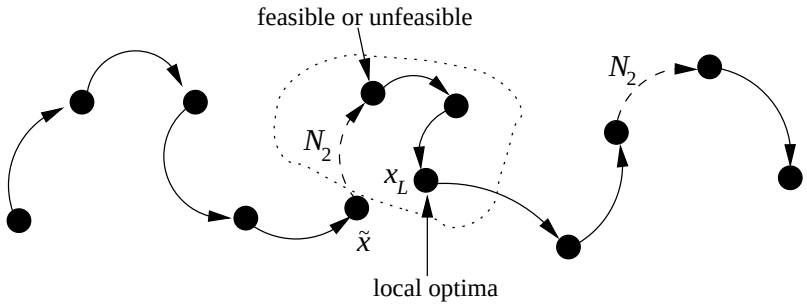

Figure 1.5. Switching N_1 and N_2: exploration of the solution space

A typical exploration of the solution space obtained by switching N_1 and N_2 is depicted in Figure 5 where a black circle is a solution, a solid arc is a set of moves generated by N_1 and a dashed arc is a set of moves generated by N_2. After some moves, the search reach $\tilde{x}$ and the neighborhoods are switched; a new solution is obtained by spreading the nodes belonging to r over the remaining k rings.

The technique of switching N_1 and N_2 is similar to the *strategic oscillation* described in Glover and Laguna, 1997, which operates by orienting moves in relation to a *critical level*. A critical level usually identifies a solution such that the normal search would not be able to improve. For example, a desired or undesidered form of a graph structure. When the critical level is reached, the strategic oscillation changes the rules of selecting moves in such a way that the critical level is crossed, that is the solution should be improved. In Figure 5, the critical level can be represented by the dotted region around the local optima x_L.

Finally, we note that all the possible moves generated by N_2 are $O(n^2)$, hence also the evaluation of N_2 has computational complexity $O(n^3)$.

6. Other intensification and diversification strategies

Path Relinking (PR) (Glover, 1999), eXploring Tabu Search (XTS) (Dell'Amico et al., 1999) and Scatter Search (SS) (Glover, 1997; Glover, 1999; Glover et al., 2000) are strategies aimed to intensify the search into promising regions or to diversify it generating new restarting solutions. They are successfully applied to several optimization problems. We have implemented all these techniques to compare their results to those obtained by TSSO.

We use the main ingredients of PR and XTS to differentiate the search within our tabu search when necessary. More precisely, we use the same framework of TSSO by replacing procedure **NewSol** with a method for generating a diverse solution based on PR or XTS.

We implemented SS along the guidelines of Laguna, 2001. SS can be intended as a general methodology in which the intensification and diversification phases are driven by a set of solutions. These solutions are then combined to generate further solutions. Our main concern is to exploit the techniques embedded in SS to improve the quality of the solution computed by BTS.

This Section is organized in three subsections describing the proposed algorithms: PR (§6.1), XTS (§6.2) and SS (§6.3).

6.1. Path Relinking

A *relinking path* is a set of moves connecting two different solutions. It can be defined as the *more direct route* or the *fewest number of moves* to connect two different solutions. Clearly, a relinking path is also a sequence of solutions.

An example is reported in Figure 1.6: the original path is shown by the solid line while the relinking one is shown by the dashed line.

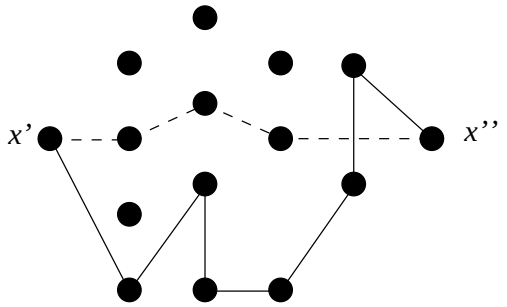

Figure 1.6. PR: example from Glover et al., 2000

The basic idea of PR is to generate a new solution by choosing one of those along the relinking path.

We consider two different assignments x' and x'' of nodes to rings. The hamming distance

$$H(x', x'') = \sum_u \sum_r \frac{|x'_{ur} - x''_{ur}|}{2}. \tag{1.17}$$

returns the minimum number of moves required to get x'' from x'. Let M_{PR} be this set of moves. Looking at Figure 1.6, M_{PR} contains all the four moves denoted by the dashed line.

We propose two PR implementations. The first one, say PR1, generates the set M_{PR1} relinking the current solution to the best one. Then, $\lfloor M_{PR1}/2 \rfloor$ moves have been selected from M_{PR1} in such a way that the objective function is minimized.

The second PR implementation proposed, say PR2, try to augment the diversification by considering more paths at the same time. Instead of relinking the current solution to the best one, we maintain an *elite solution set*, say Δ, containing a fixed number of best solutions computed during the search and we generate all the paths from the current solution to the solutions in Δ. This implementation has been proposed in Laguna et al., 1999.

An example of PR2 is depicted in Figure 1.7 where two different relinking paths are generated by two solutions belonging to Δ.

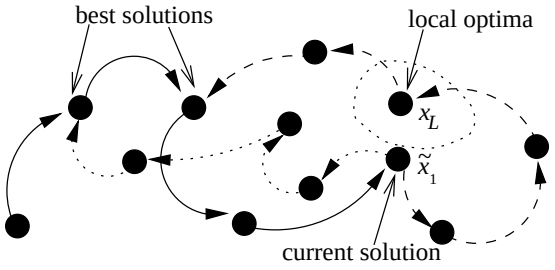

Figure 1.7. PR2: relinking paths

6.2. eXploring Tabu Search

The N_1 evaluation can generate many solutions which have quite similar objective function values. The basic idea of XTS is to adopt a *long term memory structure* to record them. Then, any of these solutions can be used to restart the search.

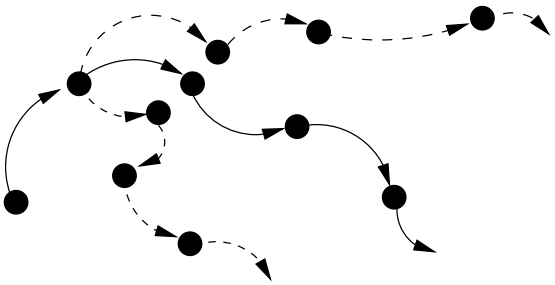

Figure 1.8. XTS: restarting and exploration with two stored solutions

An example of XTS is given in Figure 1.8: the dashed lines represent the solution space explored restarting the search from two different recorded solutions.

As already done in Dell'Amico and Trubian, 1998, our XTS implementation stores the second best not-tabu solution computed during each N_1 evaluation. These solutions are collected in a fixed length list called *second-list* (SL) with a copy of the tabu lists and the search parameters in order to restart the search with the same conditions encountered when the solution was inserted in the list SL.

Figure 1.8 reports an example of restarting phase employing XTS. Exploring N_1 from x we reach x_{first} and x_{second}: solution x_{first} is selected

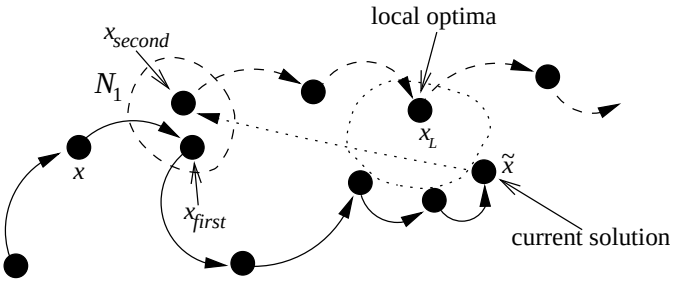

Figure 1.9. XTS: expected behavior of the search

whereas solution x_{second} is stored in SL. When we reach $\tilde{x}$, condition C_1 and C_2 tell us to diversify the search so we restart the search from x_{second}.

6.3. Scatter Search

SS is a general methodology in which the intensification and diversification phases are driven by a set of solutions called *Reference Set* which is composed of a subset of *high quality* solutions (HQ) and by a subset of *diverse* solutions (DV).

Let:

$$D_u(x, x^*) = \begin{cases} 0 & \text{if } u \text{ belongs in the same ring both in } x \text{ and } x^* \\ 1 & \text{otherwise} \end{cases}.$$

The function measuring the *diversity* of x with respect to the best solution x^* is

$$D(x, x^*) = \sum_{u=1}^{n} D_u(x, x^*). \tag{1.18}$$

Our implementation traces the general scheme proposed in Laguna, 2001. We report only the methods which are specific for SRAP:

- *Diversification Generation Method.* We iteratively generate pairs of random solutions with $h = 2, \ldots, k_{lb} + 10$ rings. In the first solution, we randomly assign each node to one of the h rings. The second solution is built from the first one by moving each node to a randomly chosen different ring (when $h = 2$ a node is moved to the other ring with 0.5 probability).

- *Solution Combination Method.* We combine the solutions belonging to the Reference Set to obtain a new, possibly better, solution

by means of the following score function:

$$s_{u,r} = \frac{\sum_{j \in S} z(x^j) x^j_{ur}}{\sum_{j \in S} z(x^j)} \tag{1.19}$$

where $z(x^j)$ denotes the objective value of the j-th solution x^j belonging to a subset S of the Reference Set. Then, the new solution $\bar{x}$ is equal to $\bar{x}_{u,r} = 1$ if $s_{u,r} = \max_r s_{u,r}$, $\bar{x}_{u,r} = 0$ otherwise.

- *Improvement Method.* We tested two different techniques to enhance the quality of the solutions generated: the first is a pure Local Search (LS) based on neighborhood N_1; the second one is our procedure BTS.

The objective functions used with SS are z_1 and a modified version of z_4:

$$\hat{z}_4 = \begin{cases} k\,B + BN & \text{if the solution is feasible} \\ n\,BN & \text{otherwise.} \end{cases}$$

Note that the original z_4 cannot be used with SS since it is based on the concept of move, but it is used within the improvement method.

Each new solution $\tilde{x}$ is inserted in HQ if its objective function value is better than that of the worst solution in HQ. On the other side, a new solution is inserted in DV if

$$\min_{x \in HQ} D(\tilde{x}, x) > \min_{y \in D} \left\{ \min_{x \in HQ} D(y, x) \right\}.$$

At the beginning, procedure **Init** initializes the reference set with the feasible or unfeasible solutions computed by the diversification generation method. During each iteration, the solution combination method (**CombineSol**) generates some new solutions which could be inserted in the reference set (**InsertRefSet**) after have being improved by **ImproveSol** procedure. The iteration terminates when none of the new combined solutions is inserted. Before starting a new iteration, SS tries to insert in the reference set some new solutions generated by the diversification generation method (**UpdateRefSet**). A possible pseudo-code of our SS is as follows.

```
SS( Instance, MaxIters ) {
  GetData( Instance );
  Init( RefSet, HQ, DV );
  Elapsed := 0; Starting := Now( ); Iters := 0;
  for ( Iters 1 to MaxIters ) do {
    Inserted := TRUE;
    while ( Inserted ) {
      Inserted := FALSE;
      for ( h = 2 to 5 ) do
        for each ( h-upla x_1, ..., x_h ∈ HQ ∪ DV ) do {
          NewSol := CombineSol( x_1, ..., x_h );
          ImproveSol( NewSol );
          Inserted := InsertRefSet( NewSol );
        };
    };
    if ( Iters ≤ MaxIters )
      UpdateRefSet( RefSet, HQ, DV );
  };
  return( HQ );
}.
```

7. Computational results

The algorithms described in the previous Sections were implemented in ANSI C language and tested on a Linux PC Pentium III/600 with 524 Megabytes of main memory. Computational experiments have been made using two sets of benchmark instances, referred to as $B1$ and $B2$.

7.1. Benchmark instances $B1$

The set $B1$ has been generated by Goldschmidt et al., 2001, and used to test their greedy algorithms. It contains 160 instances separated in two sets of instances:

- 80 *geometric* instances representing *natural cluster*, that is the fact that customers try to communicate more with their close neighbors than with their distant ones;
- 80 *random* instances.

The traffic demand between two nodes is determined by a discrete, uniform random variable indicating the equivalent number of T1 lines (a T1 line has a capacity of 1.544 Mbs) required for the estimated traffic. Each set contains both *high* and *low* demand graphs. The 40 high demand

graphs have the ring capacity B set to 622 Mbs and demand generated in $[11, 17]$ while the 40 low demand instances have $B = 155$ Mbs and demand generated in $[3, 7]$. Finally, the dimension of the instances is 15, 25, 30 and 50 nodes (10 instances each).

The reported results concern only the 118 instances proved to be feasible by optimally solving SRAP using CPLEX (the remaining 42 instances are unfeasible). The feasible instances have been solved by CPLEX within an average computing time of 20 minutes (with a maximum of 23 hours) whereas the unfeasibility of an instance has been proven within 57 hours, on average.

We have also implemented the greedy randomized procedures of Goldschmidt et al., 2001, and run each procedure with 10^5 restarts. With these algorithms 110 out of 118 instances are solved. Note that the three greedy algorithms with 10^5 restarts run for about 1 minute to solve an instance with 15 nodes and more than 1 hour for an instance with 50 nodes.

The maximum running time has been set to 3 seconds for all algorithms, except SS which terminates after 5 iterations for the version with LS and 1 iteration for the version with BTS.

In Table 1.2 we report the average computation time in milliseconds (avg. ms) and the number of instances optimally solved by TSSO (#). Procedure TSSO was tested using both conditions C_1 and C_2, one of them or none. Parameter i_{ni} in C_1 was set to 10. Using the multi-

Table 1.2. TSSO: avg. computation time in milliseconds and # of instances optimally solved

	z_1		z_2		z_3		z_4	
C_1, C_2	avg. ms	#	avg. ms	#	avg. ms	#	avg. ms	#
Y, Y	58.0	116	67.6	112	433.8	8	96.4	118
Y, N	33.4	99	37.4	96	–	6	41.7	100
N, Y	54.1	116	53.3	113	–	1	60.4	118
N, N	98.6	98	110.1	98	393.3	88	286.1	90

objective function z_4, all the feasible instances are solved setting both conditions C_1 and C_2 or only C_2. Using only C_2 TSSO has the best average computation time.

Table 1.3 highlights the results concerning the instances not solved with the greedy algorithms. We report the simple lower bound value k_{lb} (defined by (1.9)) and the optimal solution value k^* (obtained with CPLEX). The results show the effectiveness of z_1 and z_4 to drive the

Table 1.3. TSSO: instances not solved by the greedy algorithms

instance	n	m	k_{lb}	k^*	z_1		z_2		z_4	
					k	ms	k	ms	k	ms
RH.15.3	15	53	2	3	3	0	-	-	3	0
RH.30.10	30	64	3	3	3	280	4	10	3	60
RH.50.3	50	89	4	4	4	1240	4	1950	4	280
RH.50.4	50	89	3	4	4	240	4	330	4	250
RH.50.7	50	90	4	5	5	270	5	320	5	260
RL.25.1	25	51	3	4	4	30	-	-	4	30
RL.30.3	30	58	3	4	4	20	4	10	4	10
RL.30.6	30	56	3	4	4	20	4	20	4	20

search while z_2 fails to find a feasible solution for two instances and one is not optimal; z_3 fails to solve all the instances proposed.

The results reported in Table 1.4 highlights the effectiveness of the diversification method based on neighborhood N_2 to improve the quality of the solutions computed by BTS: BTS solves the 84,5% and the 76,3%

Table 1.4. Comparison of the performances of BTS and TSSO

	z_1		z_2		z_3		z_4	
	avg. ms	#	avg. ms	#	avg. ms	#	avg. ms	#
BTS	98.6	98	110.1	98	393.3	88	286.1	90
TSSO	54.1	116	53.3	113	–	1	60.4	118

of instances solved by TSSO with only condition C_2 when the objective function used is z_1 and z_4, respectively.

Table 1.5 reports the results obtained running PR1, PR2 and XTS. The parameters of these procedures have been defined through preliminary experiments as follows. Only condition C_1 is tested, $i_{ni} = 50$, 75 and 150 for PR1, PR2 and XTS, respectively. The cardinality of the elite set of PR2 was fixed to 15 and the length of the second-list was set to n. Algorithm PR2 dominates the other methods, but the performances of the three algorithms are similar. Also note that, in contrast with the behaviour observed for TSSO and BTS, method PR and XTS has significantly better performances with z_1 rather than z_4 and without checking condition C_2. We observe also that PR1, PR2 and XTS solves less instances than the greedy algorithms.

Table 1.6 reports the gaps, in terms of rings, for the instances not optimally solved by PR1, PR2 and XTS employing objective function

Table 1.5. Performances of PR1, PR2 and XTS.

	PR1		PR2		XTS	
z	avg. ms	#	avg. ms	#	avg. ms	#
z_1	94.2	102	114.4	106	152.9	103
z_2	84.8	99	127.7	104	191.5	98
z_3	412.5	97	537.9	102	368.2	88
z_4	388.0	97	245.5	97	273.8	87

Table 1.6. Gaps for instances not optimally solved by PR1, PR2 and XTS using z_1

Gap (# of rings)	PR1	PR2	XTS
$k^* + 1$	11	12	9
$k^* + 2$	2	0	3
$k^* + 2$	3	0	3
Total	16	12	15

z_1. It is interesting to observe that the maximum gap for PR2 is one, whereas PR1 and XTS have gaps up to 5. As expected, PR2 exhibits superior performance.

Table 1.7 reports the results obtained by SS using LS (SS-LS) or BTS (SS-BTS) as improvement method.

Table 1.7. SS: results of SS using as improvement methods LS or BTS.

	SS-LS		SS-BTS	
z	avg. ms	#	avg. ms	#
z_1	566.1	87	1468.2	116
z_4	1100.4	84	8593.1	113

For procedure SS-LS we set the number of high quality solutions and the number of diverse solutions to 15 and we gave a limit of 5 iterations. For SS-BTS, the cardinality of the two sets was fixed to 20, one iteration was performed and a limit of 20 milliseconds was given to each run of BTS.

Using SS-LS we obtain very poor performances with a significant computing effort. Procedure SS-BTS has better performances than the pure BTS, but with very high computing times. This confirms the effectiveness of the intensification and diversification strategies embedded in SS.

7.2. Benchmark instances $B2$

We have generated further benchmark instances in order to provide more computational results for our algorithms. The main idea is to generate *harder* instances with respect to those belonging to $B1$. We define as hard a feasible instance such that the greedy algorithms are not able to find the optimal solution, within a reasonable computing time.

We have generated 230 instances by taking the 42 unfeasible instances of $B1$ and randomly eliminating traffic demands until we reduce the total traffic of an amount greater than or equal to the difference between the value of the bottleneck ring in the (unfeasible) solution computed by TSSO and the bound B. We insert an instance in $B2$ if 10^3 restarts of the greedy algorithm determine a solution worse than the best one computed by CPLEX.

The distribution of the new instances among geometric and random graphs, high or low demand and the various values of n, are reported in tables 1.8a and 1.8b. Note that all the 230 instances are feasible. We tried to solve these instances giving the greedy algorithm a limit of 10^5 restarts, but we obtained only 46 feasible solutions.

Table 1.8a. Set $B2$: Distribution of geometric and random instances in $B2$

	High	Low	Total
Geometric	70	70	140
Random	20	70	90
Total	90	140	230

Table 1.8b. Set $B2$: Distribution of the instances by cardinality

	15	25	30	50	Total
n	50	40	50	90	230

Table 1.9 reports the results obtained by TSSO algorithm with objective functions z_1 and z_4 and the same parameter values used for the instances in $B1$.

We report the average computation time in milliseconds, the number of feasible solutions obtained and the number of solutions with fewer rings than those computed by BTS.

Although TSSO with z_4 solves all the instances in $B1$, it does not find a feasible solution for 5 instances in $B2$. However, TSSO with z_1 always finds a good feasible solution for all the instances in $B2$. We note that the average computation time is increased with respect to that of Table 1.4 but this is mainly due to the larger number of instances with 50 nodes.

Table 1.10 reports the results obtained by PR1, PR2 and XTS on set $B2$ using the same parameter adopted for solving the instances in $B1$

Table 1.9. TSSO: results for instances in $B2$

z	TSSO					
	C_1 and C_2			only C_2		
	avg. ms	#	BTS	avg. ms	#	BTS
z_1	137.5	230	12	117.3	230	15
z_2	121.8	190	13	112.5	190	15
z_3	409.4	16	-	-	0	-
z_4	135.8	225	14	110.7	225	17

again. The results confirm the effectiveness of PR2 with respect to PR1

Table 1.10. Performances of PR1, PR2 and XTS.

	PR1			PR2			XTS		
z	avg. ms	#	BTS	avg. ms	#	BTS	avg. ms	#	BTS
z_1	129.4	228	14	198.7	230	17	224.3	227	2
z_2	127.1	195	20	174.9	193	23	220.9	198	13
z_3	180.3	222	10	225.7	223	27	241.4	220	8
z_4	138.8	229	22	201.9	230	26	200.7	226	13

and XTS.

Table 1.11 reports the performances of SS obtained adopting the same parameter values used to solve the instances in $B1$. The results confirm

Table 1.11. SS: performances of SS on set $B2$

	SS-LS			SS-BTS		
z	avg. ms	#	BTS	avg. ms	#	BTS
z_1	1022.9	118	0	2638.7	229	22
z_4	2432.6	165	2	3602.1	229	32

the effectiveness of SS with BTS as improvement method.

8. Conclusions

We have considered a basic problem arising in the designing of SONET networks, which can be formulated as a graph partitioning problem with capacity constraints. We first introduced a Basic Tabu Search, then we improved it by including diversification strategies based on Strategic

Oscillation, Path Relinking and eXploring Tabu Search. A pure Scatter Search algorithm has been also implemented. The six resulting procedures have been tested through extensive computational experiments using benchmark instances both from the literature and new ones. As competitors we considered the only methods available in the literature, namely three greedy algorithms, which have been executed with 10^5 restarts. The computational experiments show the superiority of the Tabu Search method enhanced with a strategic oscillation. Moreover comparisons of the different diversification methods are outlined.

Acknowledgements

This research was supported by Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR), Italy and by Consiglio Nazionale delle Ricerche (CNR), Italy.

References

Dell'Amico, M., Lodi, A., and Maffioli, F. (1999). Solution of the cumulative assignment problem with a well-structured tabu search method. *Journal of Heuristics*, 5(2):123–143.

Dell'Amico, M. and Trubian, M. (1998). Solution of large weighted equicut problems. *European J. Oper. Res.*, 106(2-3):500–521.

Glover, F. (1997). A template for scatter search and path relinking. In Hao, J., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D., editors, *Lecture Notes in Computer Science*, volume 1363, pages 13–54.

Glover, F. (1999). Scatter search and path relinking. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 297–316. McGraw Hill.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684.

Goldschmidt, O., Laugier, A., and Olinick, E. V. (2001). SONET/SDH ring assignment with capacity contraints. Technical Report 01-EMIS-05. To appear on Discrete Applied Mathematics.

Laguna, M. (1994). Clustering for the design of sonet rings in interoffice telecommunications. *Management Science*, 40(11):1533–1541.

Laguna, M. (2001). Scatter search. In Pardalos, P. M. and Resende, M. G. C., editors, *Handbook of Applied Optimization*. Oxford Academic Press.

Laguna, M., Martí, R., and Campos, V. (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers Oper. Res.*, 26:1217–1230.