Exact solution of the SONET Ring Loading Problem

di

Mauro Dell’Amico
Martine Labbé
Francesco Maffioli

Maggio 1996
Exact Solution of the SONET Ring Loading Problem

Mauro Dell’Amico\textsuperscript{a1}, Martine Labbé\textsuperscript{b}, Francesco Maffioli\textsuperscript{c}

\textsuperscript{a} Dipartimento di Economia Politica, Università di Modena, via Berengario, 51, 41100 Modena, Italy.
\textsuperscript{b} Institut de Statistique et Recherche Opérationnelle and Service de Mathématiques de la Gestion, Universite Libre de Bruxelles, CP 210/01, Boulevard de Triomphe, B-1050 Bruxelles, Belgium
\textsuperscript{c} Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.zza L. da Vinci 32, 20133 Milano, Italy

Abstract

In this paper we address the problem of planning the capacity of the local rings in Synchronous Optical NETworks (SONET). We present efficient lower and upper bound procedures and a branch and bound algorithm which is able to find the exact solution of large instances within short computing times.

Keywords: Communications. Programming, linear: algorithms, Programming, integer: branch-and-bound.

1 Introduction

One of the most successful ways to exploit the great potential of fiber optics technology is offered by the recent introduction of the Synchronous Digital Hierarchy (SDH) in Europe and Synchronous Optical NETwork (SONET) in the US. This has suggested a number of new optimization problems, most of them in the field of Combinatorial Optimization. We address here the problem of planning the capacity of a single bi-directional ring of \( n \) fiber links connecting \( n \) Add-and-Drop Multiplexers (ADM) corresponding to \( n \) customer (or set of customers) locations.

This problem has been first studied in [1] where it is called the SONET Ring Loading Problem. To the author’s knowledge no exact approach has yet been proposed for this problem, which is known to be NP-hard [1], but not yet satisfactorily classified with respect to its approximability status: in particular it is still unknown if it can have a Polynomial Time Approximation Scheme. Only when the demands are all equal to one the problem is known to be polynomially solvable [2].

After having formulate the problem and presented a simple heuristic having a factor of 2 approximability guarantee in Section 2, we present in Section 3 a strongly polynomial algorithm for the continuous relaxation of the integer formulation. This is exploited in

\textsuperscript{1}Corresponding author: E-mail: dellamic@elet.polimi.it
Section 4 within an exact branch-and-bound approach, whose performances are extensively tested as shown by the computational results in Section 5.

2 Formulation and heuristic solution

We are given a ring graph $G = (V, E)$ that is a simple circuit with node set $V = \{1, \ldots , n\}$ and edge set $E = \{1, \ldots , n\}$. We also consider a set of $m$ demands $D = \{1, \ldots , m\}$. Each demand $i$ is defined by an origin node $o_i$, a destination node $d_i$, and a weight $w_i$ which represent the amount of traffic which must be routed from $o_i$ to $d_i$ along the ring. The entire traffic of a demand must be routed either clockwise or counter-clockwise and it cannot be splitted to be transmitted on both directions contemporarily. Each edge $\ell \in E$ may be initially loaded with a fixed traffic $t_\ell \geq 0$, which is independent of the demands in $D$ (this traffic is due to pre-routed demands). The problem we consider is to determine the routing of each demand in $D$, in order to minimize the total load of an edge.

Without loss of generality we can assume that all the input data are integers. In order to present a mathematical model we need to introduce some notation. For each demand $i$ let $\Pi_i^+$ denote the path from node $o_i$ to node $d_i$, clockwise, and $\Pi_i^-$ the path from $o_i$ to $d_i$, counter-clockwise. For each demand $i \in D$ let $x_i$ be a boolean variable which takes value one if the traffic $w_i$ is routed clockwise, value zero otherwise. We also use the continuous variable $z$ to identify the maximum load of an edge. The model is as follows.

\[
\begin{align*}
(P) \quad & \min z \\
\text{s.t.} \quad & z \geq \sum_{k: \ell \in \Pi_i^+} w_k x_k + \sum_{k: \ell \in \Pi_i^-} w_k (1 - x_k) + t_\ell, \quad \forall \ell \in E, \quad (1) \\
& x_k \in \{0, 1\}, \quad \forall k \in D. \quad (2)
\end{align*}
\]

The objective function (1) minimizes the maximum load of an edge. Constraints (2) ensure that the value of variable $z$ is not less than the load of each edge $\ell \in E$. In particular the first sum in (2) gives the total load of edge $\ell$ due to the demands routed clockwise; the second sum gives the load due to the demands routed counter-clockwise. Constraints (3) impose that the variables are binary.

The continuous relaxation $C(P)$ of problem $P$ is obtained by allowing the variables $x_i$ to assume any real value in $[0, 1]$, i.e. substituting (3) with

\[
0 \leq x_k \leq 1, \quad \forall k \in D. \quad (3')
\]

Then a demand can be splitted routing part of the traffic $w_i$ clockwise and the remaining traffic counter-clockwise.

A feasible solution for problem $P$ can be obtained from the solution of the continuous relaxation by rounding each variable with fractional value to the closest integer. This corresponds to route each fractional demand $k$ (i.e a demand associated with a value $x_k$ with $0 < x_k < 1$) either clockwise (counter-clockwise) if $x_k \geq 0.5$ ($x_k < 0.5$). We call HR
the heuristic algorithm which determine an approximate solution by applying this simple rounding technique.

**Theorem 2.1** The value of the solution obtained by algorithm HR is at most two times the value of the optimal solution.

**Proof.** Algorithm HR routes the entire traffic of a demand $k$ along the path $\Pi_k^+$ if $x_k \geq 0.5$ and along $\Pi_k^-$, otherwise. It follows that the traffic on an edge in the approximate solution is at most two times the traffic routed on that edge in the continuous relaxation. Thus the maximum load of an edge in the heuristic solution is at most $2L$, were $L$ is the value of the continuous relaxation. Since $L$ is a lower bound on the optimum solution value the result follows.

Algorithm HR can be improved by introducing a post-processing of the first $K$ demands with largest weight, having fractional values. In particular we start by rounding the values of the variables of all demands, except that of the $\lfloor \cdot \rfloor$ fractional selected. Then we enumerate the $2^K$ solutions obtained with all the possible routing combinations of the $K$ fractional demands and we choose the best integer solution. If $K$ is a constant independent of the input data the procedure can be implemented to run in polynomial time. We call this improved algorithm $KHR$.

In the next section we show how to solve efficiently problem $C(P)$. The effectiveness of procedure $KHR$ is evaluated in Section 5.

### 3 A strongly polynomial algorithm for the continuous relaxation

Before giving the details of our efficient algorithm to solve problem $C(P)$, we need to introduce some further notation. For each pair of distinct edges $r \in E, s \in E$ let $C_{r,s}$ denote the set of *crossing demands*, i.e. the set of demands which must be routed exactly on one of the two edges $r, s$. Formally,

$$C_{r,s} = \{ k \in D : r \in \Pi_k^+ \text{ and } s \in \Pi_k^- \text{ or } r \in \Pi_k^- \text{ and } s \in \Pi_k^+ \}.$$  

Any pair of distinct edges $(r, s)$ partitions the node set into two subsets, and is thus called a *straight cut* of $G$.

In the following we will speak indifferently of the pair of distinct edges $(r, s)$ or of the cut $(r, s)$. Let $\tau$ denote the set of all possible straight cuts of $G$.

Given a demand index $i \in D$ and a partial solution $x_k, k = 1, \ldots, i$, let

$$F^i_\ell = \sum_{k \leq i : \ell \in \Pi_k^+} w_k x_k + \sum_{k \leq i : \ell \in \Pi_k^-} w_k (1 - x_k), \quad \ell \in E$$  

denote the total traffic on edge $\ell$, due to the first $i$ demands. In a solution to $C(P)$, the load of an edge $\ell$ is then $F^i_\ell + t_\ell$. 


We start the description of our algorithm by introducing a simple lower bound on the value of the optimal solution to C(P). Given a cut (r, s) one can see that the traffic of each crossing demand must be routed through edge r or edge s, so

\[ L_1 = \frac{1}{2} \max_{(r,s) \in \tau} \left( \sum_{k \in C_{r,s}} w_k + t_r + t_s \right) \]

is a valid lower bound.

Since the optimal load of an edge \( \ell \in E \) is at least \( t_\ell \), an improved bound is

\[ L = \max\{L_1, L_2\}, \]

with \( L_2 = \max_{\ell \in E} t_\ell \). The bound \( L \) is the optimal solution value for problem C(P). To prove this we give an \( O(mn^2) \) algorithm, called CSONET, which determines a feasible solution to C(P).

Our algorithm considers a demand at a time and routes it optimally by determining the maximum traffic which can be routed clockwise (or counter-clockwise) without exceeding the bound \( L \). More precisely we have \( m \) iterations, one for each demand. At iteration \( i \) the routing of the first \( i - 1 \) demands is already fixed so that the edges of the ring have preassigned loads and we consider demand \( i \) and the path \( \Pi_i^+ \). For all edges and straight cuts in \( \Pi_i^+ \) we compute two values \( C1 \) and \( C2 \), similar to \( L_1 \) and \( L_2 \), which give the total amount of traffic which have to be certainly routed on some edge of \( \Pi_i^+ \), disregarding demand \( i \). We do the same for path \( \Pi_i^- \) and we determine the gaps \( \Delta^+ \) and \( \Delta^- \) between the lower bound \( L \) and the above quantities. If \( \Delta^+ \geq \Delta^- \) then we route the traffic equal to the minimum of \( w_i \) and \( \Delta^+ \) along \( \Pi_i^+ \), and the possibly remaining traffic of demand \( i \) along \( \Pi_i^- \). A similar routing is made if \( \Delta^- \geq \Delta^+ \), but with paths \( \Pi_i^+ \) and \( \Pi_i^- \) exchanged.

The details of procedure CSONET are given in the pseudocode below.

**Algorithm CSONET**

**begin**

compute \( L_1 = \frac{1}{2} \max_{(r,s) \in \tau} \left( \sum_{k \in C_{r,s}} w_k + t_r + t_s \right) \);

compute \( L = \max(L_1, \max_{\ell \in E} t_\ell) \);

for each \( \ell \in E \) do \( F_{\ell}^0 := 0 \);

for \( i := 1 \) to \( n \) do

\( C1 := \frac{1}{2} \max_{(r,s) \in \Pi_i^+} \left( \sum_{k \geq i} w_k + F_{r}^{i-1} + F_{s}^{i-1} + t_r + t_s \right) \);

\( C2 := \max_{\ell \in \Pi_i^+} (t_\ell + F_{\ell}^{i-1}) \);

\( \Delta^+ := L - \max(C1, C2) \);

\( C1 := \frac{1}{2} \max_{(r,s) \in \Pi_i^-} \left( \sum_{k \geq i} w_k + F_{r}^{i-1} + F_{s}^{i-1} + t_r + t_s \right) \);

\( C2 := \max_{\ell \in \Pi_i^-} (t_\ell + F_{\ell}^{i-1}) \);

**end**
\[\Delta^- := L - \max(C1, C2)\;\]

if \(\Delta^+ \geq \Delta^-\) then
\[x_i := \min(1, \frac{\Delta^+}{w_i});\]
else
\[x_i := \max(0, 1 - \frac{\Delta^-}{w_i});\]
endif
endfor
end.

The following theorem proves the correctness of algorithm CSONET

**Theorem 3.1** Algorithm CSONET determines an optimal solution of problem C(P) and the optimal solution value is \(L\).

**Proof.** We show that at the end of each iteration \(i\) of the main loop for in CSONET, the following two conditions hold:

\[L \geq \frac{1}{2}(\sum_{k > i}^{\infty} w_k + F_r^i + F_s^i + t_r + t_s) \quad \forall (r,s) \in \tau,\quad (6)\]

\[L \geq t_{\ell} + F_{\ell}^i, \quad \forall \ell \in E.\quad (7)\]

If condition (7) is satisfied at the end of the \(n\)-th iteration, then no edge \(\ell\) has load greater than \(L\). Since value \(L\) is a lower bound on the optimal solution value to C(P), then the solution provided by algorithm CSONET is optimal and the result follows.

We now prove, by induction, that conditions (7)-(6) hold at each iteration. We will give the details of the proof for the case \(\Delta^+ \geq \Delta^-\), the prove for the other case \((\Delta^+ < \Delta^-)\) can be obtained with similar arguments.

The two conditions (7) and (6) are certainly satisfied at the beginning of the procedure (iteration 0), by definition of \(L\). Now suppose they hold at the end of iteration \(i - 1\) \((0 < i \leq n)\): we will show that they hold also at the end of iteration \(i\).

From the definition of \(C1\) and \(C2\), and the fact that conditions (7) and (6) are satisfied at the end of iteration \(i - 1\), we have that \(C1 \leq L, C2 \leq L\), so \(\Delta^+ \geq 0\) and the algorithm routes \(w_i x_i (=\min(w_i, \Delta^+))\) traffic units of demand \(i\), on each edge \(\ell \in \Pi^+_i\).

We first prove that the thesis holds for the edges and the straight cuts in \(\Pi^+_i\). From the above considerations we have that \(w_i x_i \leq \Delta^+ \leq L - C2\) and \(C2 \geq t_{\ell} + F_{\ell-1}^i\) for each \(\ell \in \Pi^+_i\), so that \(L \geq t_{\ell} + F_{\ell-1}^i + w_i x_i\). Since \(F_{\ell}^i = F_{\ell-1}^i + w_i x_i\), condition (7) immediately holds. Similarly, for condition (6) we have

\[w_i x_i \leq \Delta^+ \leq L - C1 \leq L - \frac{1}{2}(\sum_{k > i}^{\infty} w_k + F_r^{i-1} + F_s^{i-1} + t_r + t_s)\]
for each \( r \in \Pi_i^+, s \in \Pi_i^+ \). Since demand \( i \) is not included in \( C_{r,s} \) for all \( r \in \Pi_i^+ \) and \( s \in \Pi_i^+ \) we have

\[
L \geq \frac{1}{2} \left( \sum_{k \in C_{r,s} \atop k > i} w_k + F_r^{i-1} + F_s^{i-1} + 2w_i x_i + t_r + t_s \right) = \frac{1}{2} \left( \sum_{k \in C_{r,s} \atop k > i} w_k + F_r^{i} + F_s^{i} + t_r + t_s \right)
\]

and condition (6) holds for any straight cut in \( \Pi_i^+ \).

We now prove that the two conditions (6), (7) hold for the edges and the straight cuts in \( \Pi_i^- \). If \( x_i = 1 \) then no traffic from demand \( i \) is routed on path \( \Pi_i^- \), so the only remaining case is \( 0 < \Delta^- < w_i \) (i.e. the algorithm routes \( \Delta^- \) traffic units along path \( \Pi_i^+ \) and \( w_i - \Delta^- \) along \( \Pi_i^- \)). By definition of \( \Delta^- \) we know that after iteration \( i \) at least one of the following two cases arises:

- case (i) an edge \( r \) in \( \Pi_i^+ \) is saturated i.e. \( t_r + F_r^{i-1} + w_i x_i = t_r + F_r^i = L \),
- case (ii) a straight cut \( (r, s) \) with \( r \in \Pi_i^+ \), \( s \in \Pi_i^+ \) is saturated, i.e.

\[
\frac{1}{2} \left( \sum_{k \in C_{r,s} \atop k > i} w_k + F_r^{i-1} + F_s^{i-1} + t_r + t_s + 2\Delta^- \right) = \frac{1}{2} \left( \sum_{k \in C_{r,s} \atop k > i} w_k + F_r^{i} + F_s^{i} + t_r + t_s \right) = L.
\]

In the remaining of the proof we refer to four particular edges \( r, s, u, v \) (see Figure 1), with \( r \in \Pi_i^+ \), \( s \in \Pi_i^+ \), \( u \in \Pi_i^- \), \( v \in \Pi_i^- \) and such that edge \( r \) precedes edge \( s \), if we move clockwise from \( o_i \) to \( d_i \), and \( u \) precedes \( v \), if we move counter-clockwise from \( o_i \) to \( d_i \).

![Figure 1](image-url)

**Figure 1:** The four particular edges \( r, s, u \) and \( v \).

*Case (i).* Let \( r \) denote the edge in \( \Pi_i^+ \) saturated by the routing of demand \( i \), i.e

\[
t_r + F_r^{i-1} + \Delta^- = L
\]

Since condition (6) holds, at the end of iteration \( i - 1 \), for any cut \( (r, j) \) with \( r \in \Pi_i^+ \) and \( j \in \Pi_i^- \), we have that

\[
\frac{1}{2} \left( \sum_{k \in C_{j,r} \atop k > i} w_k + F_j^{i-1} + t_j + t_r \right) \leq \frac{1}{2} \left( \sum_{k \in C_{j,r} \atop k > i} w_k + w_i + F_j^{i-1} + F_r^{i-1} + t_j + t_r \right) \leq L
\]

(9)
Multiplying inequality (9) by 2 and subtracting equation (8) we obtain:

$$\sum_{k > i} w_k + w_i + F_{ji}^{i-1} - \Delta^+ + t_j \leq L.$$ 

Observing that $F_{ji}^{i} = F_{ji}^{i-1} + w_i - \Delta^+$, for each edge $j \in \Pi_i^-$, we immediately conclude that $t_j + F_{ji}^{i} \leq L$, and condition (7) holds.

Condition (6) has to be considered for straight cuts in $\Pi_i^-$ only if $\Pi_i^-$ contains at least two distinct edges, say $u$ and $v$. Therefore we can refer to the two cuts $(u, r)$ and $(v, r)$. Adding two inequalities of the type (9) for $j = v$ and $j = u$, respectively, and subtracting (8) we obtain

$$\frac{1}{2}(\sum_{k > i} w_k + \sum_{k > i} w_k + 2w_i + F_{ju}^{i-1} + F_{ju}^{i-1} + t_u + t_v - 2\Delta^+) \leq L.$$ 

Reminding again that $F_{ju}^{i} = F_{ju}^{i-1} + w_i - \Delta^+$ and that $F_{vu}^{i} = F_{vu}^{i-1} + w_i - \Delta^+$, we obtain

$$\frac{1}{2}(\sum_{k > i} w_k + \sum_{k > i} w_k + F_{ju}^{i} + F_{ju}^{i} + t_u + t_v) \leq L.$$ 

It is now sufficient to observe that $C_{u,v} \subseteq C_{u,r} \cup C_{v,r}$ to derive

$$\frac{1}{2}(\sum_{k > i} w_k + F_{ju}^{i} + F_{ju}^{i} + t_u + t_v) \leq L,$$

which shows that condition (6) holds for any straight cut $(u, v)$ in $\Pi_i^-$. 

Case(ii). Let $(r, s)$ with $r \in \Pi_i^+$ and $s \in \Pi_i^+$ denote the saturated cut, i.e.

$$\frac{1}{2}(\sum_{k > i} w_k + F_{jr}^{i-1} + F_{js}^{i-1} + t_r + t_s) + \Delta^+ = L \tag{10}$$

and consider an edge $j \in \Pi_i^-$ and the two cuts $(j, r)$ and $(j, s)$. Note that inequality (9) can be rewritten with $s$ instead of $r$, so that adding the two inequalities of type (9) associated with cuts $(j, r)$ and $(j, s)$, and observing that: $C_{r,s} \subseteq C_{j,r} \cup C_{j,s}$ we obtain

$$t_j + w_i + F_{ji}^{i-1} + \frac{1}{2}(\sum_{k > i} w_k + F_{jr}^{i-1} + F_{js}^{i-1} + t_r + t_s) \leq 2L$$

Subtracting equation (10) and reminding that: (a) demand $i \notin C_{r,s}$; (b) $F_{ji}^{i} = F_{ji}^{i-1} + w_i - \Delta^+$ we obtain $t_j + F_{ji}^{i} \leq L$ and condition (7) holds.
We conclude the proof by showing that condition (6) holds also for any cut \((u,v)\) with \(u \in \Pi_i^-\) and \(v \in \Pi_i^-\). Consider the two cuts \((v,r)\) and \((u,s)\) and observe that \(C_{u,v} \cup C_{r,s} \subseteq C_{v,r} \cup C_{u,s}\). Adding the two inequalities of type (9) associated to the cuts \((u,s)\) and \((v,r)\), and subtracting equation (10) we obtain
\[
\frac{1}{2} \left( \sum_{k > i} w_k + F_{u}^{i-1} + F_{v}^{i-1} + t_u + t_v + 2(w_i - \Delta^+) \right) \leq L
\]
Since \(F_{\ell}^{i} = F_{\ell}^{i-1} + w_\ell - \Delta^+, \forall \ell \in \Pi_i^-\) condition (6) immediately follows.

3.1 Complexity and implementation

Algorithm CSONET performs \(m\) iterations, one for each demand. At the generic iteration \(i\) we have to compute two times the quantities \(C1\) and \(C2\). Each evaluation of \(C2\) requires \(O(n)\) time, whereas to determine the value of \(C1\) we have to consider all the possible straight cuts with both edges in \(\Pi_i^+\) (or in \(\Pi_i^-\)) and for each cut \((r, s)\) we have to compute (11)
\[
\sum_{k \geq i} w_k + F_{r}^{i-1} + F_{s}^{i-1} + t_r + t_s.
\]
A naive implementation requires \(O(m)\) time for each evaluation of (11), and \(O(mn^2)\) time to determine \(C1\). A more efficient implementation can be obtained using a matrix \(A = [a_{i,j}]\) to store the \(n \times (n - 1)/2\) sums \(\{\sum_{k \geq i} w_k : k \in C_{r,s}\}\), for all cuts \((r, s)\) of the ring. With this data structure we need \(O(1)\) time to evaluate (11) and \(O(n^2)\) time to determine \(C1\). Matrix \(A\) needs to be initialized at iteration 0 of the algorithm and updated when each iteration is concluded. The construction of the initial matrix can be done in \(O(mn)\) time, as we show here below. The updating, instead, requires \(O(n^2)\) time since it is enough to consider all the \((n^2 - n)/2\) cuts \((r, s)\) and subtract \(w_i\) from \(a_{r,s}\) if \(i \in C_{r,s}\). Therefore the overall time complexity of algorithm CSONET is \(O(mn^2)\).

Moreover we note that if matrix \(A\) is given, then the lower bound \(L1\) can be computed in \(O(n^2)\), and the lower bound \(L2\) can be obtained in \(O(n)\). Therefore if one is only interested into the value \(L\) of the solution to \(C(P)\) (and not to the corresponding values of \(x_i\)) the entire computation requires \(O(mn)\) time against the \(O(mn^2)\) time necessary to have the complete solution.

We now show how to compute the initial matrix \(A\) in \(O(mn)\) time. We select each edge \(r\) in turn and we compute the weight \(a_{r,s} = \sum_{k \in C_{r,s}} w_k + t_r + t_s\) for all \(s \in E, s \neq r\). The computation of the values \(a_{r,s}\) starts with \(s\) being the first edge following \(r\), clockwise, and continue choosing \(s\) as the next edge of the ring, clockwise, until we reach \(r\). At the first step, when \(r\) and \(s\) are adjacent, the value of \(a_{r,s}\) is given by the sum of all demands with origin or destination in the node common to the edges \(r\) and \(s\). When we consider a cut \((r,s')\) immediately after the computation of \(a_{r,s}\), with \(s\) and \(s'\) being adjacent edges
Figure 2: The case of adjacent edges while computing matrix $A$.

(see Figure 2), we do not need to compute the new value from scratch, but it is enough to set $a_{r,s'} = a_{r,s}$, and then to remove from $a_{r,s'}$ the weights of the demands which have their origin or destination in the node $j$ common to $s$ and $s'$ and destination or origin on the path from $r$ to $s$, clockwise, and to add to $a_{r,s'}$ the weights of the demands which have origin or destination on node $j$ and destination or origin on the path from $s'$ to $r$, clockwise. For a given edge $r$, during the computation of all the values $a_{r,s}$, we consider each demand at most twice, so giving a time complexity of $O(m)$. Repeating this procedure for the $n-1$ possible choices of $r$ we obtain the overall time complexity of $O(mn)$.

4 An exact algorithm

The results of the previous sections have been used to develop a depth-first branch-and-bound algorithm for the exact solution of problem $P$. Our basic algorithm is as follows.

We use a binary decision tree. At each node we solve problem $C(P)$ modified by fixing the routing of a set of demands. Among the demands routed with a fractional traffic, in the solution to $C(P)$, we determine the one having maximum traffic. Two child nodes are generated by assigning this demand either clockwise or counter-clockwise.

At the root node we set $t_{\ell} = 0$ for each $\ell \in E$, then we apply algorithm CSONET of the previous section to solve problem $C(P)$ and to obtain the lower bound $L$. The continuous solution is used by algorithm KHR of Section 2 to obtain an heuristic solution of value $UB$. If $L = UB$ we are done, otherwise we start the exploration of the branching decision tree. When a demand $i$ is routed along a path $\pi$ (i.e. a forward step is performed) we increase by $w_i$ the values $t_{\ell}$ for each $\ell \in \pi$. When a backtracking occurs the value $w_i$ is subtracted from $t_{\ell}$, for all $\ell \in \pi$. Observe that the loads $t_{\ell}$ completely describe the traffic due to a partial solution, therefore no modification in the algorithms of Sections 2 and 3 is required in order to compute lower and upper bounds for the intermediate nodes of the tree.

Preliminary computational experiments performed with instances randomly generated
as in [1] show that the time required by our algorithm to find an optimal solution is one
order of magnitude less than the time required by CPLEX to find an integer solution
which is within 0.5% of the optimum (for more details on the experiments with CPLEX
see [1]).

Stimulated by these results we tried to improve the performances of our basic algorithm
with several techniques.

Presorting of the demands. We observed experimentally that the value of the heuristic
solution obtained with KHR is very close to the optimum value. Therefore we decided
to change the branching rule. We do not select the fractional variable with maximum
traffic, but the first fractional variable in a presorted list. This list is defined in such a way
that that the first descent in the branch-decision-tree, from the root node to a leaf, builds
exactly the heuristic solution. Our guess is that a limited number of backtrackings (i.e.
routing changes) are required to obtain the optimal solution. We also tried a presorting
by nonincreasing weights.

Improvement of the lower bound. Given the solution to \( C(P) \), we consider the \( K \) largest
demands with fractional value (where \( K \) is the same parameter introduced in Section 2
to obtain the improved heuristic algorithm). Similarly to procedure KHR, we enumerate
the \( 2^K \) partial solutions obtained with all the possible routing combinations of these \( K \)
demands and we apply the lower bound procedure to the remaining \( m - K \) demands of
each partial solution. The minimum of these bounds is a valid lower bound which improves
bound \( L \). We apply this technique to all the nodes of the tree.

Second lower bound. Given a straight cut \((r, s)\) and the crossing demands \( C_{r,s} \), we know
that each of these demands must be routed through edge \( r \) or \( s \). Thus minimizing the
maximum load on \( r \) and \( s \), with an integer assignment of the demands in \( C_{r,s} \), gives a
valid lower bound on the optimum solution value. This problem can be transformed into
a subset-sum problem in which a set of \( q \) items is given, each of which has an associated
weight, and a subset \( S \) of the \( q \) items which maximizes the sum of the weights of the item
in \( S \) without exceeding a given capacity \( c \) is sought. Given a straight cut \((r, s)\), we define
an instance of subset-sum with capacity \( c = \frac{1}{2} \sum_{i \in C_{r,s}} w_i \) and with an item \( i \), having weight
\( w_i \), associated to each demand \( i \in C_{r,s} \). The optimal solution to the subset-sum problem
minimizes the difference between \( c \) and the sum of the weights of the items selected, so it
also minimizes the difference between the items selected and discarded. In terms of the
SONET problem this solution minimizes the difference of the traffic loads on \( r \) and \( s \) and
thus the maximum of the two. For each of the \( n(n - 1)/2 \) straight cuts we solve the cor­
responding subset-sum problem with a C translation of the FORTRAN routine contained
in the book by Martello and Toth [3]. We apply this bound to all the nodes of the tree.

All the above improvements of the basic branch-and-bound have been tested with
several computational experiments. The impact of each technique on the computational
times is the following.

- Each kind of presorting leads to smaller computational times for some instances and largest for other instances, so, on average, no improvement can be obtained with this technique.

- The improvement of the lower bound halves the computational times.

- The second lower bound has value larger than that of the continuous relaxation only for a few instances. In these cases, it allows to reduce drastically the size of the branch-decision-tree, but on average the computational time required to compute the bound is not compensated by the reduction of the tree.

From the above preliminary computational experiments we have seen that significative improvements are obtained only with a strenghtening of the lower bound coming from continuous relaxation. In order to obtain the maximum information from this bound we have developed the following fixing procedure which tries to fix the values of the fractional variables either to zero or one. The procedure performs a series of iterations. At each iteration we consider, in turn, each demand, say \(j\), with fractional value in the optimal solution to \(C(P)\). We route the entire traffic of demand \(j\) clockwise updating the values \(t_e\) accordingly and we compute again the lower bound \(L\). If the lower bound is larger than or equal to the current best solution value then the demand must be routed counter-clockwise to have a chance to obtain a better solution value. If the value of the fractional variable has not been fixed by this tentative, we route the demand \(j\) counter-clockwise and we repeat the same steps to try to fix it. When all the fractional demands have been considered we compute the new solution of the continuous relaxation of the problem obtained from \(P\) by fixing the routing of some demands, and a new iteration is started. The algorithm stops when, at one iteration, no demand have been fixed. The pseudocode follows.

**Procedure** FIXING

**input:** upper bound value \(U\);

**repeat**
- compute lower bound \(L\); find the optimal solution \(x^*\) of \(C(P)\);
- \(NOFIX := TRUE\);
  **for** each demand \(j\) with \(x^*_j \neq \lfloor x^*_j \rfloor\) **do**
  - route demand \(j\) clockwise; compute lower bound \(L\);
  - if \(L \geq U\) **then**
    - fix \(x^*_j := 0\); let \(NOFIX := FALSE\);
  - else
    - route demand \(j\) counter-clockwise; compute lower bound \(L\);
    - if \(L \geq U\) **then** fix \(x^*_j := 1\); let \(NOFIX := FALSE\);
  **endif**
  **endfor**
**until** \((NOFIX = TRUE)\)
The time complexity of procedure FIXING is $O(I(mn^2 + Fmn))$ where $I$ is the number of iteration performed and $F$ the number of fractional variables. In the worst case it is $O(m^3 n^2)$, but in practice, since the number of fractional demands is small, it is quite fast.

The use of this procedure at each node of the branch-decision-tree reduced the computational times of one order of magnitude. Moreover we noted that with this procedure the improved lower bound has no more effect. So our final code is the basic branch-and-bound with procedure FIXING applied at each node.

5 Computational results

We have coded in C language the heuristic procedure KHR of Section 2, the lower bound procedure CSONET of Section 3 and the branch-and-bound of the previous Section. Moreover we have implemented two greedy heuristics as those described in [1]. The first heuristic, that we call 1PG, is a One Phase Greedy which consider a demand at a time and tentatively routes it clockwise and counter-clockwise. The demand is routed on the path which determines the minimum increasing in the current solution value. The second heuristic, say 2PG, is a Two Phase Greedy which tentatively routes the current demand clockwise and counter-clockwise, then complete the partial solution with 1PG and routes the demand along the path associated with the minimum solution value. This algorithm is slightly different from that in [1], since we use 1PG to determine the routing of each demand, instead Cosares and Saniee apply 1PG only if there is a tie in the partial load determined by the routing of the current demand clockwise or counter-clockwise.

We generated random instances as in [1]: the weights of the demands are randomly selected according to the uniform distribution in $[5,100]$ and origins and destinations are different integers also uniformly randomly selected. The results for this class of problems, with values of $n = (20,40,60,80,100)$ and values of $m$ ranging from 50 to 1000 are reported in Table I.

Let $z^H$ be the solution value of the generic heuristic $H$ for a specific instance, and let $z^*$ be its optimum solution value (computed using our branch-and-bound algorithm). For each instance we compute the percentage error respect to the optimal solution, i.e. $err = 100(z^H - z^*)/z^*$. The columns labelled $\Delta_{\text{avg}}$ and $\Delta_{\text{max}}$, in the table, report, respectively, the average and the maximum percentage error. The columns labelled $\text{time}$ give the average computing time in seconds on a PC Pentium with a clock at 100 Mhertz. For each entry in the tables 20 instances have been generated and solved. The labels $F_{\text{avg}}$ and $F_{\text{max}}$ indicate, respectively, the average and maximum number of fractional variables obtained in the solution of the continuous relaxation.

For these experiments we fixed the parameter $K$ of heuristic KHR to five.
Table I. Uniformly random instances with $w_i \in [5,100]$: heuristic solutions

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$m$</th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{max}$</th>
<th>$F_{avg}$</th>
<th>$F_{max}$</th>
<th>time</th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{max}$</th>
<th>$F_{avg}$</th>
<th>$F_{max}$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>KHR</td>
<td>20</td>
<td>50</td>
<td>2.3</td>
<td>6.3</td>
<td>1.55</td>
<td>3</td>
<td>0.01</td>
<td>16.8</td>
<td>40.0</td>
<td>0.01</td>
<td>9.7</td>
<td>20.1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50</td>
<td>1.2</td>
<td>2.7</td>
<td>1.60</td>
<td>3</td>
<td>0.04</td>
<td>20.2</td>
<td>50.1</td>
<td>0.01</td>
<td>12.0</td>
<td>21.6</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>50</td>
<td>0.5</td>
<td>1.0</td>
<td>1.20</td>
<td>3</td>
<td>0.04</td>
<td>21.2</td>
<td>32.2</td>
<td>0.01</td>
<td>12.0</td>
<td>18.0</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>100</td>
<td>1.1</td>
<td>2.9</td>
<td>1.75</td>
<td>3</td>
<td>0.08</td>
<td>19.2</td>
<td>29.3</td>
<td>0.01</td>
<td>12.3</td>
<td>22.7</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>50</td>
<td>0.6</td>
<td>1.7</td>
<td>1.40</td>
<td>2</td>
<td>0.20</td>
<td>24.0</td>
<td>31.6</td>
<td>0.01</td>
<td>14.3</td>
<td>19.6</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>50</td>
<td>0.3</td>
<td>0.6</td>
<td>1.30</td>
<td>3</td>
<td>0.46</td>
<td>25.8</td>
<td>31.2</td>
<td>0.04</td>
<td>17.9</td>
<td>24.1</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>150</td>
<td>0.7</td>
<td>1.8</td>
<td>1.50</td>
<td>5</td>
<td>0.36</td>
<td>18.5</td>
<td>31.6</td>
<td>0.01</td>
<td>9.6</td>
<td>17.5</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td>0.5</td>
<td>1.0</td>
<td>1.45</td>
<td>3</td>
<td>0.78</td>
<td>23.6</td>
<td>32.6</td>
<td>0.04</td>
<td>14.6</td>
<td>19.6</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td>0.2</td>
<td>0.5</td>
<td>1.00</td>
<td>1</td>
<td>1.60</td>
<td>25.0</td>
<td>32.6</td>
<td>0.12</td>
<td>18.0</td>
<td>25.0</td>
</tr>
<tr>
<td>1PG</td>
<td>80</td>
<td>200</td>
<td>0.5</td>
<td>1.4</td>
<td>1.65</td>
<td>3</td>
<td>0.90</td>
<td>21.1</td>
<td>33.9</td>
<td>0.01</td>
<td>13.4</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>50</td>
<td>0.3</td>
<td>0.7</td>
<td>0.95</td>
<td>2</td>
<td>1.81</td>
<td>24.1</td>
<td>37.6</td>
<td>0.08</td>
<td>16.0</td>
<td>21.1</td>
</tr>
<tr>
<td></td>
<td>800</td>
<td>50</td>
<td>0.1</td>
<td>0.3</td>
<td>1.15</td>
<td>3</td>
<td>3.58</td>
<td>26.7</td>
<td>31.0</td>
<td>0.28</td>
<td>20.4</td>
<td>23.7</td>
</tr>
<tr>
<td>2PG</td>
<td>100</td>
<td>250</td>
<td>0.6</td>
<td>1.2</td>
<td>1.65</td>
<td>3</td>
<td>1.72</td>
<td>23.2</td>
<td>28.8</td>
<td>0.04</td>
<td>15.6</td>
<td>21.1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>50</td>
<td>0.2</td>
<td>0.6</td>
<td>1.11</td>
<td>2</td>
<td>2.29</td>
<td>24.0</td>
<td>27.2</td>
<td>0.09</td>
<td>15.9</td>
<td>20.2</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50</td>
<td>0.1</td>
<td>0.2</td>
<td>1.12</td>
<td>2</td>
<td>4.81</td>
<td>27.3</td>
<td>35.9</td>
<td>0.22</td>
<td>20.8</td>
<td>23.0</td>
</tr>
</tbody>
</table>

Pentium/100 seconds; averages over 20 instances.

Procedure KHR dominates the other heuristics and it is fast. Its average errors are very small: they exceed one percent only for small instances ($n = 20$, $m \leq 100$, and $n = 40$, $m = 100$). The maximum error never exceeds a few percent. Heuristics 1PG and 2PG have errors one order of magnitude larger than that of KHR. 1PG has negligible computing time, but produces the worst solutions. The average error of 2PG is one third to one half less than that of 1PG, but its computing time is two order of magnitude larger. KHR has computing times larger than that of 1PG, but very small: less than five seconds for the largest instances.

It is worth noting that currently the real instances have $n \leq 20$ and $m \leq 200$ and the largest weight of a demand is bounded by 100, so the first three rows of our table give a sample of real-life problems.

The same instances of Table I have been solved with the branch-and-bound algorithm of the previous Section. The results are given in Table II. For each pair $n, m$ we give the average and maximum computing time (over 20 instances), the average and maximum number of explored nodes, the average and maximum depth of the branch-decision-tree and the ratio between the value of the lower bound at the root node ($L1$) and the optimum solution value ($z^*$).
Table II. Uniformly random instances with $w_j \in [5, 100]$; exact solution

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>$m$</td>
<td>$avg$</td>
<td>$max$</td>
<td>$avg$</td>
<td>$max$</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>0.26</td>
<td>0.82</td>
<td>35.8</td>
<td>215</td>
</tr>
<tr>
<td>100</td>
<td>102</td>
<td>2.55</td>
<td>44.6</td>
<td>127</td>
<td>10.2</td>
</tr>
<tr>
<td>200</td>
<td>2.68</td>
<td>7.86</td>
<td>23.0</td>
<td>77</td>
<td>9.3</td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>2.49</td>
<td>7.70</td>
<td>46.0</td>
<td>209</td>
</tr>
<tr>
<td>200</td>
<td>7.71</td>
<td>17.71</td>
<td>38.0</td>
<td>133</td>
<td>12.2</td>
</tr>
<tr>
<td>400</td>
<td>36.40</td>
<td>81.89</td>
<td>87.0</td>
<td>323</td>
<td>21.4</td>
</tr>
<tr>
<td>60</td>
<td>9.83</td>
<td>22.16</td>
<td>42.3</td>
<td>103</td>
<td>14.1</td>
</tr>
<tr>
<td>300</td>
<td>24.17</td>
<td>71.92</td>
<td>43.1</td>
<td>263</td>
<td>13.5</td>
</tr>
<tr>
<td>600</td>
<td>233.94</td>
<td>2541.64</td>
<td>674.4</td>
<td>12761</td>
<td>19.1</td>
</tr>
<tr>
<td>80</td>
<td>200</td>
<td>25.30</td>
<td>109.95</td>
<td>56.8</td>
<td>507</td>
</tr>
<tr>
<td>400</td>
<td>135.31</td>
<td>682.79</td>
<td>185.3</td>
<td>1637</td>
<td>19.2</td>
</tr>
<tr>
<td>800</td>
<td>251.03</td>
<td>949.85</td>
<td>44.8</td>
<td>219</td>
<td>18.5</td>
</tr>
<tr>
<td>100</td>
<td>250</td>
<td>41.97</td>
<td>129.26</td>
<td>52.0</td>
<td>208</td>
</tr>
<tr>
<td>500</td>
<td>162.16</td>
<td>752.41</td>
<td>253.8</td>
<td>2191</td>
<td>22.7</td>
</tr>
<tr>
<td>1000</td>
<td>304.84</td>
<td>1117.81</td>
<td>72.0</td>
<td>338</td>
<td>24.0</td>
</tr>
</tbody>
</table>

Pentium/100 seconds; averages over 20 instances.

All the instances with size as those of the largest real life problems ($n \leq 20$ and $m \leq 200$) are solved by the exact approach with small computing times. Larger instances are also solved in reasonable computing times, but the maximum times grow up to one hour. The average number of explored nodes is, in general, small. This is mainly due to our fixing procedure (see Section 4). For large instances however the lower bound at the root node, in many cases, is equal to the optimum solution value, but the algorithm determines a feasible solution with this value only after visiting a large number of nodes.

To better understand the behaviour of the heuristic KHR and of the branch-and-bound procedure we generated instances from three further classes: (i) $w_j \in [5, 1000]$; (ii) $w_j \in [50, 100]$ and (iii) $w_j \in [500, 1000]$.

The performances of the three heuristics are very similar to those obtained with weights $w_j \in [5, 100]$, but the new instances are more difficult for the branch-and-bound. The results are reported in Tables III-V: the numbers in brackets give the number of instances which have not been solved within one hour of CPU time (in these cases the averages are computed on the number of solved instances).
Table III. Uniformly random instances with $w_j \in [5, 1000]$: exact solution

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>avg</th>
<th>max</th>
<th>avg</th>
<th>max</th>
<th>avg</th>
<th>max</th>
<th>$L1/z^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>0.16</td>
<td>1.57</td>
<td>399.9</td>
<td>1569</td>
<td>199.7</td>
<td>784</td>
<td>0.998</td>
</tr>
<tr>
<td>100</td>
<td>0.47</td>
<td>1.57</td>
<td>376.3</td>
<td>1195</td>
<td>187.9</td>
<td>597</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>3.93</td>
<td>15.71</td>
<td>636.7</td>
<td>1632</td>
<td>318.1</td>
<td>816</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>2.99</td>
<td>6.29</td>
<td>516.3</td>
<td>1123</td>
<td>257.9</td>
<td>561</td>
<td>0.997</td>
</tr>
<tr>
<td>200</td>
<td>20.27</td>
<td>103.71</td>
<td>1396.8</td>
<td>9094</td>
<td>698.0</td>
<td>4547</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>39.60</td>
<td>80.14</td>
<td>557.8</td>
<td>1989</td>
<td>278.6</td>
<td>994</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>150</td>
<td>15.09</td>
<td>33.00</td>
<td>903.6</td>
<td>2490</td>
<td>451.7</td>
<td>1245</td>
<td>1.000</td>
</tr>
<tr>
<td>300</td>
<td>24.04</td>
<td>73.86</td>
<td>310.6</td>
<td>1414</td>
<td>155.0</td>
<td>707</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>55.00</td>
<td>128.86</td>
<td>160.3</td>
<td>666</td>
<td>79.8</td>
<td>333</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>200</td>
<td>20.43</td>
<td>48.71</td>
<td>333.6</td>
<td>894</td>
<td>166.5</td>
<td>447</td>
<td>1.000</td>
</tr>
<tr>
<td>400</td>
<td>73.86</td>
<td>198.00</td>
<td>308.1</td>
<td>943</td>
<td>153.8</td>
<td>471</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>260.32</td>
<td>1831.29</td>
<td>4813.8</td>
<td>33127</td>
<td>2406.6</td>
<td>16563</td>
<td>0.998</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>250</td>
<td>50.29</td>
<td>136.71</td>
<td>541.6</td>
<td>1608</td>
<td>270.6</td>
<td>804</td>
<td>1.000</td>
</tr>
<tr>
<td>500</td>
<td>161.07</td>
<td>781.00</td>
<td>782.0</td>
<td>6512</td>
<td>390.9</td>
<td>3256</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>106.86</td>
<td>660.00</td>
<td>195.7</td>
<td>1516</td>
<td>97.8</td>
<td>758</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>200</td>
<td>180.71</td>
<td>3952</td>
<td>590.7</td>
<td>1976</td>
<td>0.999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>73.86</td>
<td>198.00</td>
<td>308.1</td>
<td>943</td>
<td>153.8</td>
<td>471</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>143.47</td>
<td>424.29</td>
<td>145.9</td>
<td>609</td>
<td>72.8</td>
<td>304</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>348.07</td>
<td>2609.86</td>
<td>42582.5</td>
<td>394236</td>
<td>21291.1</td>
<td>197118</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>79.99</td>
<td>220.00</td>
<td>85.2</td>
<td>229</td>
<td>42.3</td>
<td>114</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>229.74</td>
<td>534.29</td>
<td>135.7</td>
<td>717</td>
<td>67.8</td>
<td>358</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Pentium/100 seconds; averages over 20 instances.

Table IV. Uniformly random instances with $w_j \in [50, 100]$: exact solution

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>avg</th>
<th>max</th>
<th>avg</th>
<th>max</th>
<th>avg</th>
<th>max</th>
<th>$L1/z^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>0.31</td>
<td>3.14</td>
<td>330.3</td>
<td>2382</td>
<td>164.9</td>
<td>1191</td>
<td>0.994</td>
</tr>
<tr>
<td>100</td>
<td>2.20</td>
<td>7.86</td>
<td>916.2</td>
<td>3244</td>
<td>458.0</td>
<td>1622</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1.26</td>
<td>4.71</td>
<td>91.8</td>
<td>441</td>
<td>45.7</td>
<td>220</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>4.09</td>
<td>17.29</td>
<td>498.3</td>
<td>1735</td>
<td>249.0</td>
<td>867</td>
<td>1.000</td>
</tr>
<tr>
<td>200</td>
<td>16.81</td>
<td>106.86</td>
<td>1068.5</td>
<td>7902</td>
<td>534.1</td>
<td>3951</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>33.52</td>
<td>138.29</td>
<td>746.5</td>
<td>4394</td>
<td>373.3</td>
<td>2197</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>150</td>
<td>8.96</td>
<td>26.71</td>
<td>479.6</td>
<td>2020</td>
<td>239.6</td>
<td>1010</td>
<td>0.998</td>
</tr>
<tr>
<td>300</td>
<td>42.74</td>
<td>262.43</td>
<td>735.7</td>
<td>6272</td>
<td>367.6</td>
<td>3136</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>106.86</td>
<td>660.00</td>
<td>195.7</td>
<td>1516</td>
<td>97.8</td>
<td>758</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>200</td>
<td>180.71</td>
<td>1181.6</td>
<td>3952</td>
<td>590.7</td>
<td>1976</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>60.34</td>
<td>180.71</td>
<td>1181.6</td>
<td>3952</td>
<td>590.7</td>
<td>1976</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>143.47</td>
<td>424.29</td>
<td>145.9</td>
<td>609</td>
<td>72.8</td>
<td>304</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>348.07</td>
<td>2609.86</td>
<td>42582.5</td>
<td>394236</td>
<td>21291.1</td>
<td>197118</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>79.99</td>
<td>220.00</td>
<td>85.2</td>
<td>229</td>
<td>42.3</td>
<td>114</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>229.74</td>
<td>534.29</td>
<td>135.7</td>
<td>717</td>
<td>67.8</td>
<td>358</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Pentium/100 seconds; averages over 20 instances.
Table V. Uniformly random instances with \( w_j \in [500, 1000] \): exact solution

<table>
<thead>
<tr>
<th>( n )</th>
<th>( m )</th>
<th>( \text{avg} )</th>
<th>( \text{max} )</th>
<th>( \text{avg} )</th>
<th>( \text{max} )</th>
<th>( \text{avg} )</th>
<th>( \text{max} )</th>
<th>( L_1/z^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>50</td>
<td>4.40</td>
<td>15.71</td>
<td>3442.7</td>
<td>11066</td>
<td>1721.1</td>
<td>5533</td>
<td>0.995</td>
</tr>
<tr>
<td>100</td>
<td>29.86</td>
<td>14.14</td>
<td>1838.3</td>
<td>6947</td>
<td>919.0</td>
<td>3474</td>
<td>0.997</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>42.14</td>
<td>130.14</td>
<td>6109.6</td>
<td>20117</td>
<td>3054.7</td>
<td>10058</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>15.09</td>
<td>48.71</td>
<td>2843.7</td>
<td>10667</td>
<td>1421.6</td>
<td>5333</td>
<td>0.998</td>
</tr>
<tr>
<td>200</td>
<td>1796.14</td>
<td>11493.3</td>
<td>57066</td>
<td>193262</td>
<td>12293.0</td>
<td>96631</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>454.61</td>
<td>234.77</td>
<td>24827.5</td>
<td>77485</td>
<td>12413.5</td>
<td>38742</td>
<td>0.998</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>107.49</td>
<td>271.86</td>
<td>7317.2</td>
<td>22843</td>
<td>4533.1</td>
<td>16264</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1293.29</td>
<td>9066.6</td>
<td>32528</td>
<td>61453</td>
<td>4533.1</td>
<td>16264</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>200</td>
<td>234.77</td>
<td>895.71</td>
<td>6001.7</td>
<td>12460</td>
<td>3000.7</td>
<td>11422</td>
<td>0.998</td>
</tr>
<tr>
<td>400</td>
<td>271.86</td>
<td>1535.28</td>
<td>7704.1</td>
<td>36110</td>
<td>3851.8</td>
<td>18055</td>
<td>0.998</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>256.14</td>
<td>1353.28</td>
<td>7704.1</td>
<td>36110</td>
<td>3851.8</td>
<td>18055</td>
<td>0.998</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1000.67</td>
<td>1293.29</td>
<td>9066.6</td>
<td>32528</td>
<td>4533.1</td>
<td>16264</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>500</td>
<td>234.77</td>
<td>895.71</td>
<td>6001.7</td>
<td>12460</td>
<td>3000.7</td>
<td>11422</td>
<td>1.000</td>
</tr>
<tr>
<td>1000</td>
<td>1002.67</td>
<td>1293.29</td>
<td>9066.6</td>
<td>32528</td>
<td>4533.1</td>
<td>16264</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Pentium/100 seconds; averages over 20 instances.

6 Some final remarks

During the final writing of this work it came to our attention that part of it, namely the arguments upon which the algorithm for the continuous relaxation of Section 3 is based has been independently developed in an undated report by Schrijver, Seymour and Winkler [4]. In this report the authors develop a very efficient heuristic for SONET Ring Loading Problem, and also prove that it is 3/2-Approximable by a simple Greedy rounding algorithm. Moreover their algorithm finds the optimum solution whenever all demands are equal to one.

Whenever the computing time available is sufficient it is obviously much more satisfactory to use the exact method presented here. As computational results of Section 5 tend to show, this exact approach is quite efficient and probably suitable in most practical situations.

Acknowledgements

This work has been partially sponsored by Human Capital and Mobility Contract n. ERBCHRCXT930087. The third author thanks the NSFR of Belgium for financial support.
References


17
24. Fernando Vianello [1987] “Effective Demand and the Rate of Profit. Some Thoughts on Marx, Kalecki and Stagflation”, pp. 41
27. Giovanni Proacci [1988] “The State and Social Control in Italy During the First World War”, pp. 18
40. Leonardo Paggi [1988] “‘Americanismo e riformismo. La socialdemocrazia europee nell’economia mondiale aperta”, pp. 120
43. Giovanna Proacci [1989] “State coercion and social solidarity in Italy (1915-1918): the moral and political content of social ances”, pp. 41
44. Carlo Alberto Magni [1989] “Reputazione e credibilità di una minaccia in un giro bancario”, pp. 56
55. Paolo Silvestri [1990] “Sull’autonomia finanziaria dell’università”, pp. 31


118. Mario Forni e Marco Lippi [1995] “Permanent income, heterogeneity and the error correction mechanism.” pp. 21


128. Carlo Alberto Magni [1996] “Reportable and una tantum real options a dynamic programming approach” pp. 23


137. David Avra Lane, Irene Poli, Michele Lalla, Alberto Roverato [1996] “Lezioni di probabilità e inferenza statistica - Esercizi svolti -” pp. 302


139. Luisa Malaguti e Costanza Torricelli [1996] “Monetary policy and the term structure of interest rates”, pp. 30