A Matheuristic for a Telecommunication Network Design Problem with Traffic Grooming

Xinyun Wu\textsuperscript{a}, Jean-François Cordeau\textsuperscript{b}, Zhipeng Lü\textsuperscript{a}

\textsuperscript{a}SMART, School of Computer Science \& Technology, Huazhong University of Science \& Technology, Wuhan, P.R.China, 430074

\textsuperscript{b}CIRRELT and HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal (Québec), Canada H3T 2A7

Abstract

This paper addresses a network design and traffic grooming problem arising in optical telecommunication networks based on wavelength division multiplexing. Given a set of nodes and a set of traffic demands between these nodes, the network design and traffic grooming problem (NDGP) consists of installing a minimum number of lightpaths between the nodes and of routing the demand on the lightpaths while respecting capacity constraints. We introduce a mathematical formulation of the NDGP as well as a hybrid algorithm capable of finding high quality solutions in short computing times. The proposed algorithm uses linear and mixed integer programming as slave methods and embeds them into a tabu search heuristic. Computational results and comparisons with an existing heuristic from the literature show the effectiveness of the proposed algorithm. Further analyses also show the efficiency of the neighborhood structure and of its evaluation technique.

Keywords: OR in telecommunications; network design with traffic grooming; matheuristic; mixed integer programming; tabu search.

Email addresses: jean-francois.cordeau@hec.ca (Jean-François Cordeau), zhipeng.lv@hust.edu.cn (Zhipeng Lü)

Preprint submitted to Elsevier

May 22, 2016
1. Introduction

The network design and traffic grooming problem (NDGP) arises in the design of optical telecommunication networks. One of the main costs in a wavelength division multiplexing (WDM) network is the transceivers installed, which provide optical connections (edges) between pairs of nodes in the network [18]. The optical connections in the network are called lightpaths and they are responsible for transferring traffic demands. In practice, the capacity of lightpaths is usually much larger than the bandwidth of one traffic demand. Assigning an exclusive lightpath to each demand is thus costly and unjustified. Since one lightpath can usually be shared by several demands, effective optimization algorithms can be developed to reduce the cost of the required transceivers.

The NDGP tackled in this paper was recently introduced by Wu et al. [28] and it can be stated as follows. Given a set of nodes and a set of traffic demands, the objective is to design an optical network with the smallest number of lightpaths to satisfy all the traffic demands. Wu et al. proposed a two-level iterated local search (TL-ILS) algorithm, which consists of two local search procedures in a nested structure. The main local search handles the transformation of the topology (to decrease the number of lightpaths) while the inner search validates the feasibility of the current solution (i.e. the grooming subproblem). The inner algorithm, which is implemented through a tree-search based neighborhood, is the most important part of the TL-ILS. This algorithm uses a breadth-first search strategy and a cutoff mechanism when there is no hope to find a better route. These components reduce redundant calculations when the candidate routes for a given traffic demand share common sub-paths.

In this paper, we propose a matheuristic, i.e., an algorithm that combines local search with mathematical programming techniques, for the NDGP. The proposed algorithm implements tabu search as the master algorithm and em-
beds two slave algorithms, which are linear programming and mixed integer
programming. This new algorithm has found improved solutions to each of
22 instances in a benchmark set.

During the last decades, optimization problems arising in WDM net-
works have received extensive attention due to their economic significance.
One of these problems is the traffic grooming problem which consists of
routing each traffic demand in the network while respecting the capacity of
the lightpaths. This problem was proven to be NP-complete by Wang and
Gu [25]. Chen and Rouskas proposed an effective and efficient hierarchical
traffic grooming framework for WDM networks [6], while Saleh and Kamal
addressed the problem of designing and provisioning WDM networks to sup-
port many-to-many traffic grooming in order to minimize the overall network
cost [22]. They also introduced two novel approximation algorithms for the
many-to-many traffic grooming problem [23]. A mathematical formulation
of the traffic grooming problem in WDM mesh networks was presented by
Zhu and Mukherjee [34]. Zhang et al. [32] proposed a multi-layer auxiliary
graph to jointly solve the electrical-layer routing and optical-layer routing
and spectrum assignment problems. This work was then improved by Zhang
et al. [31], who added a “sub-transponder layer” in the auxiliary graph and
proposed two spectrum reservation schemes for multi-flow transponders to
improve the transponder’s utilization.

Liu et al. [16] investigated the survivable traffic grooming problem for elas-
tic optical networks with flexible spectrum grid employing new transmission
technologies. Rubio et al. [21] proposed two evolutionary algorithms, which
are multiobjective variants of the standard differential evolution and variable
neighborhood search, for solving the traffic grooming problem. Wu et al. [30]
studied a very similar problem as this paper but with an additional simple
physical path constraint for each traffic demand. Dutta et al. [8] articulated
how the traditional optical networking research area of traffic grooming may
be combined with recent advances in Internet architecture, specifically the proposed ChoiceNet Future Internet architecture, to create an agile system capable of reflecting both provider and customer interests on an ongoing basis as network conditions change.

The traffic grooming problem is often combined with the routing and wavelength assignment problem, which is denoted as the traffic grooming routing and wavelength assignment (GRWA) problem [3, 5, 13, 15, 33]. Vignac et al. [24] proposed a mathematical formulation and an exact method for solving the GRWA problem. Their approach is highly effective for realistic size instances. The routing and wavelength assignment problem is itself an important problem aimed at reducing the usage of wavelengths (see, e.g., [4, 19, 20, 29]).

The NDGP is also related to the classical multi-commodity capacitated network design problem (MCND) [7, 9]. However, as it has been stated in [28], the NDGP is different from the MCND and its variants [1, 2, 12], not only because it requires the flow variables to be integer, but also because the network to be designed can be a multi-graph in the case of the NDGP. These features make the NDGP particularly challenging to solve. There are several similar problems studied in the literature but most consider a given set of edges that should either be open or closed. This contrasts with the NDGP in which only a set of nodes is given and the network, which may form a multi-graph, has to be designed from scratch.

The remainder of the paper is organized as follows. A new mathematical formulation of the problem is presented in Section 2. The framework of the proposed matheuristic is then presented in Section 3, while the formulation used in the slave algorithms is introduced in Section 3.1. The neighborhood structure of the master problem is described in Section 3.2. The proposed algorithm is detailed in Section 3.3. Experimental results and the analysis are described in Section 4, before concluding the paper in Section 5.
2. Problem definition and mathematical formulation

2.1. Problem description

The network design problem with traffic grooming is defined as follows. We are given a node set \( V = \{1, \ldots, n\} \) and a set of traffic demands \( \Gamma \). The elements of \( \Gamma \) are tuples \((s_t, d_t, c_t)\) representing the source and sink nodes as well as the bandwidth of each traffic demand \( t \). The objective is to design an optical network, denoted as a multi-graph \( G = (V, L) \), and groom all the demands in set \( \Gamma \) with the minimum number of lightpaths. Here, \( L \) represents the set of lightpaths used in the network. Note that set \( L \) is not defined a priori but is actually the output of the optimization model.

Note that the traffic demands and the lightpaths are undirected. The network to be designed is a multi-graph, i.e., there may be several lightpaths in the graph with the same source and sink nodes. What makes the problem difficult is that the traffic demands are unsplittable in the sense that every traffic demand has to be assigned to a unique sequence of lightpaths forming a path from the origin to the destination of the demand. Thus, we cannot consider the lightpath capacity to be additive on each edge.

The model introduced in [28] assumes that when several lightpaths are installed between a pair of nodes, their capacity can be shared, which may lead to infeasible solutions. Suppose there are three traffic demands with bandwidth of 2 and the capacity of a lightpath is 3. If these demands all travel through the same two nodes, there should be at least three lightpaths between the pair of nodes to carry them all. However, according to the model described in [28] two lightpaths are enough, which does not reflect reality. In the following section, we will introduce a new mathematical formulation that properly models the case where traffic demands are unsplittable.

2.2. Problem formulation

Let \( B \) denote the capacity of each lightpath. We define \( T^t_i \) as a binary parameter taking value 1 if traffic \( t \) originates from \( i \), -1 if traffic \( t \) is destined
to $i$, and 0 otherwise. Since in the worst case each traffic demand will be assigned to a different lightpath, the number of lightpaths will not exceed the number of traffic demands. Thus, an upper bound on the number of lightpaths is $m$. For each potential lightpath $l$ (with $1 \leq l \leq m$), we also define the following binary decision variables:

$x_l$ equals 1 if lightpath $l$ is in use, 0 otherwise;

$s^l_i$ equals 1 if the origin of lightpath $l$ is node $i$, 0 otherwise;

d^l_i$ equals 1 if the destination of lightpath $l$ is node $i$, 0 otherwise;

$v^t_l$ equals 1 if demand $t$ uses $l$ in the forward direction, 0 otherwise;

$w^t_l$ equals 1 if demand $t$ uses $l$ in the reverse direction, 0 otherwise.

Given this notation, we can formulate the NDGP as follows.
Minimize

$$\sum_{l=1}^{m} x_l$$

subject to:

$$\sum_{i=1}^{n} s^l_i = 1 \quad l = 1, 2, ..., m$$

(2)

$$\sum_{i=1}^{n} d^l_i = 1 \quad l = 1, 2, ..., m$$

(3)

$$x_l - v^t_l - w^t_l \geq 0 \quad t = 1, 2, ..., m \quad l = 1, 2, ..., m$$

(4)

$$\sum_{l=1}^{m} v^t_l (s^l_i - d^l_i) + \sum_{l=1}^{m} w^t_l (d^l_i - s^l_i) = T^t_i \quad t = 1, 2, ..., m \quad i = 1, 2, ..., n$$

(5)

$$\sum_{t=1}^{m} c_t (v^t_l + w^t_l) \leq B x_l \quad l = 1, 2, ..., m$$

(6)

$$x_l, s^l_i, d^l_i, v^t_l, w^t_l \in \{0, 1\} \quad t, l = 1, 2, ..., m \quad i = 1, 2, ..., n.$$  

(7)
Constraints (2) and (3) ensure that each lightpath has one source and one sink. Constraints (4) ensure that every traffic demand only uses feasible lightpaths, and one lightpath cannot be used by the same traffic demand both forward and backward. Constraints (5) ensure that the path for each traffic is feasible. Constraints (6) ensure that capacity constraints are satisfied.

3. Solution method

The solution method that we propose to solve the NDGP is a matheuristic that relies on a local search to design the network and on linear programing to solve the grooming problem. For an overview of matheuristics, we refer the interested reader to Maniezzo et al. [17]. The pseudocode of the algorithm is given in Algorithm 1, where \( X_{\text{best}} \) records the best solution, i.e., the network with the fewest lightpaths found so far. At the beginning, \( X_{\text{best}} \) is initialized by a greedy algorithm. The initialization method is the same as in [28] and it can be described as follows: from an empty solution, assign each traffic demand sequentially in the current network, introducing a new lightpath whenever the assignment of a demand is infeasible. In each local search iteration, a randomly selected lightpath is removed from \( X_{\text{best}} \). The resulting network is then passed to the TS\(_{k\text{NDG}}\) procedure, whose role is to try to find a feasible network with a number of lightpaths \( k = |X_{\text{best}}| - 1 \). The solution with the minimum number of lightpaths found by the algorithm is returned at the end of the \( \text{nbIter} \) local search iterations, where \( \text{nbIter} \) is a parameter of the algorithm, and \( \text{iterTS} \) represents the number of iterations used by procedure TS\(_{k\text{NDG}}\).

In the above description, the stopping criterion is the total number of iterations (\( \text{nbIter} \)), but it is easy to adapt the algorithm so as to use an alternative stopping criterion, such as the total computing time.

It can be seen that our approach is based on a reduction to the decision version of the problem, the \( k\text{-NDGP} \). In order to solve a NDGP instance, i.e.,
Algorithm 1 Algorithm for the NDGP

1: $X_{\text{best}} \leftarrow \text{Initialization}()$
2: $\text{iter} \leftarrow 0$
3: while $\text{iter} < \text{nbIter}$ do
4: $X \leftarrow \text{removeLightpath}(X_{\text{best}})$
5: $(\text{succ}, X, \text{iterTS}) \leftarrow \text{TS}_{\text{kNDG}}(X, \text{nbIter} - \text{iter})$
6: $\text{iter} \leftarrow \text{iter} + \text{iterTS}$
7: if $\text{succ} = \text{true}$ then
8: $X_{\text{best}} \leftarrow X$
9: else
10: return $X_{\text{best}}$
11: end if
12: end while
13: return $X_{\text{best}}$

to design a network $G = (V, L)$ with the fewest lightpaths, our algorithm tries repeatedly to find a network $G$ with $k$ lightpaths, i.e., it tackles a series of $k$-NDGP instances with decreasing values of $k$. This is one of the main differences between the proposed algorithm and that of [28]. Therefore, in the following sections we mainly discuss the $k$-NDGP.

3.1. A MIP approach to the grooming subproblem

In this section we introduce an exact method to solve the grooming subproblem of the NDGP. We define the grooming subproblem as follows. Given a fixed optical network and a set of traffic demands, the goal is to determine the routes for all the traffic demands ensuring that there is no overloaded lightpath in the network. From the previous definition of NDGP, it is clear that the grooming subproblem is exactly the NDGP with fixed lightpath decision variables ($x^l$, $s^l_i$ and $d^l_i$). For the grooming subproblem, if the integrality constraints on the flow variables ($v^l_i$ and $w^l_i$) are relaxed, the problem becomes a multi-commodity flow problem with constraints but without an objective function. This multi-commodity flow problem with fractional flow variables can easily be solved by linear programming. However, an easy constraints satisfaction formulation does not help very much. In order to embed the exact method into the local search heuristic, it is desirable to understand the causes of infeasibility when the instance is infeasible. Therefore, we change
this decision problem into another optimization problem as follows. For each lightpath \( l \), we define a new constant \( \Lambda^l_i \) equal to 1 if the lightpath originates from \( i \), -1 if it terminates at \( i \), and 0 otherwise. We also define constants \( \delta_l \) representing the overload for each lightpath \( l \). The model for the grooming subproblem can be given as follows.

Minimize
\[
F = \sum_{l=1}^{\mid L \mid} \delta_l \tag{8}
\]

subject to
\[
\sum_{l=1}^{\mid L \mid} (\Lambda^l_i v^t_l - \Lambda^l_i w^t_l) = T^t_i \quad t = 1, 2, ..., m \quad i = 1, 2, ..., n \tag{9}
\]
\[
\sum_{t=1}^{m} c_l (v^l_t + w^l_t) - \delta_l \leq B \quad l = 1, 2, ..., \mid L \mid \tag{10}
\]
\[
\delta_l \geq 0 \quad l = 1, 2, ..., \mid L \mid \tag{11}
\]
\[
v^l_t, w^l_t \in \{0, 1\} \quad t, l = 1, 2, ..., m. \tag{12}
\]

Constraints (9) ensure the flows for each traffic demand. They are almost identical to constraints (5) in the original model. Constraints (10) guarantee that the flow on lightpath \( l \) minus \( \delta_l \) should be less than or equal to the capacity of one lightpath. The objective function (8) is the total overload of the network. It is clear that the solution represents a feasible grooming configuration if and only if \( F = 0 \). Moreover, the solution indicates the overloaded lightpaths when the grooming configuration is infeasible.

This mixed integer programming problem can be solved by a MIP solver such as CPLEX in reasonable time, because the number of variables is not excessive. Another good feature of this formulation is that it has a small integrality gap. This characteristic helps a lot when the model is used to evaluate the neighborhood structure.
3.2. A matheuristic for the $k$-NDGP

In this section, we explain how the exact algorithm for the grooming subproblem is embedded into a local search algorithm. The local search algorithm is considered as the master algorithm while the MIP and LP procedures are used as slave procedures integrated into the local search.

In the following, we first present the local search approach by defining the search space, the cost function, and the neighborhood. The procedure to tackle the $k$-NDGP will be described in Section 3.3.

3.2.1. Search space and cost function

The search space (set of configurations) explored by our local search procedure is denoted by $S$. In the proposed local search approach, a configuration $X$ is any network of $k$ edges (lightpaths) with the assignment for each traffic demand.

The cost $f(X)$ is defined as the overload of the network:

$$f(X) = \sum_{t=1}^{m} \left[ \max \left( \sum_{t=1}^{m} c_t(v_t^t + w_t^t) - Bx_t, 0 \right) \right].$$

We can notice that a configuration $X$ represents a feasible solution of the $k$-NDG if and only if $f(X) = 0$. In our algorithm, $f(X)$ is evaluated by solving the grooming subproblem which is described in Section 3.1.

3.2.2. Neighborhood definition

Given a configuration $X \in S$, we denote by $\langle x, y \rangle$ the operation, named a twist operator, by which a lightpath $x$ is removed from $G$, while a new lightpath $y$ is added to $G$ with the restriction that $x$ and $y$ must share a common vertex, and the traffic is re-groomed on $G$. In addition, $X \oplus \langle x, y \rangle$ denotes the configuration obtained by applying the twist operator $\langle x, y \rangle$ to $X$.

The twist operation is illustrated in Figure 1. The bold gray lines represent the lightpaths and the thin black lines represent the traffic demands,
where \( \Gamma = \{ t_{AB}, t_{CB}, t_{BD} \} \) and \( L = \{ l_{AC}, l_{AB}, l_{AD} \} \). Let us denote by \( X' = X \oplus < l_{AD}, l_{BD} > \) the configuration obtained by applying move \( < l_{AD}, l_{BD} > \) to configuration \( X \). We have \( X' \) with \( L = \{ l_{AC}, l_{AB}, l_{BD} \} \) and the assignment for \( t_{BD} \) changes from \( \{ l_{AB}, l_{AD} \} \) to \( \{ l_{BD} \} \).

![Figure 1: Illustration of the twist operator \(< l_{AD}, l_{BD} >\) ](image)

The time complexity will be \( O(2|L|(|V|−1)) \) if all the configurations in the neighborhood are checked. However, the calculation will be excessive if doing so. In most cases, there is no positive effect to applying a move to a lightpath which is in a part of the graph with no capacity problems. Instead, we only apply moves to lightpaths which are adjacent to an overloaded lightpath. More specifically, assuming there is an overloaded lightpath \( l_{ij} \) with endpoints \( i \) and \( j \), the lightpaths \( l_{iv} \) (\( l_{jv} \)) with one endpoint \( i \) (\( j \)) will be evaluated to see if it would improve the cost function to twist it to \( l_{jv} \) (\( l_{iv} \)), which is the move \( < l_{iv}, l_{jv} > \) (\( < l_{jv}, l_{iv} > \)), in the hope that some traffic demands on \( l_{ij} \) will change their route to void using \( l_{ij} \) so that the overload can be decreased.

For example, in Figure 1, if lightpath \( l_{AB} \) is overloaded, the two moves that should be evaluated are \( < l_{AD}, l_{BD} > \) and \( < l_{AC}, l_{CB} > \). In this way, the calculation can be reduced to \( O(2\mathcal{O}D) \), where \( \mathcal{O} \) represents the number of overloaded lightpaths and \( D \) represents the average degree of the vertices in the network.

At each iteration, the best move, which leads to the minimum \( f(X) \), is chosen and applied to the current configuration. In other words, the best improvement strategy is used.
3.2.3. The essence of the twist neighborhood

In this section we explain the essence of the proposed twist neighborhood. For most of the configurations which are infeasible solutions of the problem, there are usually only a few overloaded lightpaths. Thus, it makes sense that we can just do operations related to the overloaded lightpaths to save time.

The basic idea is to keep the overloaded lightpaths unchanged, but to make changes to those that are neighbors of the overloaded ones, in the hope that some traffic demands on the overloaded lightpath will change their routes and reduce the overload. For example in Figure 2, there is an illustration of a part of one configuration. To make things clearer, we only present three lightpaths and three traffic demands here to represent three different kinds of lightpaths and traffic demands involved during a twist operation. Suppose lightpath $AB$ is overloaded and lightpath $AC$ is the one to be moved, i.e., the move operator is $\langle AC, BC \rangle$. There are three kinds of traffic demands which are as follows:

- $t_1$ The traffic demand only passes through the lightpath to be moved ($AC$) but not the overloaded one ($AB$);
- $t_2$ The traffic demand passes through both the overloaded lightpath ($AB$) and the one to be moved ($AC$);
- $t_3$ The traffic demand only passes through the overloaded lightpath ($AB$) but not the one to be moved ($AC$).

After the move operation, $t_2$ is much likely to change its route to avoid traversing $AB$. If that is the case the overload will decrease. But one can also observe that $t_1$ is likely to change its route to traverse $AB$. If so, the overload may be worse. Usually $t_3$ is not affected by the move. Therefore, if there are more traffic demands of the type $t_1$ than $t_2$ affected by the move, this move may reduce the overload of the corresponding lightpath.
3.2.4. Completeness of the twist move

Proposition 1 states that the twist operator is sufficient to explore the whole solution space of the k-NDGP.

**Proposition 1.** If $X$ is a feasible solution for k-NDGP, then any configuration $X'$ can be altered to $X$ by performing a finite number of twist moves.

**Proof.** Let us first define a basic move operator $(p,q)$ which consists of deleting lightpath $p$ from $G$ and adding a new lightpath $q$ to $G$ without other restrictions. Assuming that $X$ is a feasible solution, it is obvious that any configuration $X'$ can be altered to $X$ by performing a finite number of basic moves $(p,q)$, only if we restrict that $p \in X'$ and $q \in X$. For each $(p,q)$, if $p$ and $q$ share a common vertex then it is equivalent to the twist operator $< p, q >$; otherwise, it can be equal to a sequence of two twist operators $[< p, r >, < r, q >]$ only if $r$ shares common vertices both with $p$ and $q$. The lightpath $r$ can be determined by choosing any endpoint of $p$ and $q$ and making these two to be the endpoints of $r$. We have shown that a basic operator can be replaced by one or a sequence of two twist operators. Hence, any configuration $X'$ can be altered to the feasible solution $X$ by performing a finite number of twist moves. \hfill \Box

3.2.5. Evaluation of the neighborhood

Given a move $m \in M(S)$, where $M(S)$ is the set of the candidate moves of configuration $S$, it is important to evaluate its effect in an efficient way. Of course we can just calculate the cost function $f(X \oplus m)$ of the resulting
solution to see whether the move is good or not. However, computing time would become prohibitive. Different mechanisms have been proposed in the literature for neighborhood evaluation in a local search algorithm. The fast incremental evaluation is one of them (see, e.g., [14, 26, 27, 29]). It calculates the score of $m$, i.e. its impact on the cost of the configuration. Then the move with the least score will be considered to be the best move in the case of a minimization problem. However, the fast incremental evaluation is usually effective only for problems with simple solution structures. In NDGP, the solution structure consists of the topology of the network and the routes for each traffic. It is hard to design a fast evaluation mechanism for the neighborhood introduced before.

In this paper, an estimation method is used to evaluate each move. Here, the cost function of the configurations is evaluated rather than the score of the moves. In other words, we estimate $f(X \oplus m)$ to check whether $m$ is good or not. The grooming subproblem formulation described in Section 3.1 is used to evaluate each neighboring configuration. However, instead of solving the MIP model we solve the linear programming relaxation of this model for each configuration in the neighborhood. The configuration with minimum $F$ is chosen to be the next solution.

We now explain how the evaluation works in detail. For the following illustration, we denote by $\Theta$ an empty vector, i.e., the vector with zero elements, and define two auxiliary functions described as follows:

$map(g, A)$ is a function that applies a given function $g$ to each element of a row vector $A = [a_1 \ a_2 \ ...]$, returning a row vector of results in the same order, defined as

$$map(g, A) = \begin{cases} \Theta & \text{if } A = \Theta, \\ ([g(a_1)] | map(g, [a_2 \ a_3 \ ...])) & \text{otherwise}, \end{cases}$$

(14)

cfun(g, x)$ takes two parameters, of which $g$ is a function receiving two
parameters and \( x \) is a variable with the same type as the first parameter of \( g \). It returns another function, which takes only one parameter, with the same functionality as \( g \) but fixes its first parameter to \( x \).

With the notation defined above, the candidate move operator set can be calculated as

\[
\mathcal{M}(L^o) = \begin{cases} 
\Theta & \text{if } L^o = \Theta, \\
\left([\text{map}(\text{cfun}(\text{move}, l_1^o), \mathcal{N}(l_1^o))] | \mathcal{M}([l_2^o ...])\right) & \text{otherwise},
\end{cases}
\]

where \( L^o = [l_1^o l_2^o ...] \) represents the vector of overloaded lightpaths and \( \mathcal{N}(l_1^o) \) represents the vector of lightpaths that is neighbored to \( l_1^o \). Here, \( \text{move} \) is a function that takes two parameters \( l_1 \) and \( l_2 \), and produces a twist move that will twist lightpath \( l_2 \) from one endpoint of \( l_1 \) to another endpoint of \( l_1 \). It is defined as

\[
\text{move}(l_1, l_2) = \langle \text{lp}(s\text{End}(l_1, l_2), d\text{End}(l_1, l_2)) \rangle,
\]

where function \( \text{lp}(u, v) \) creates a lightpath from \( u \) to \( v \), function \( s\text{End}(l_1, l_2) \) gives the vertex that is shared by \( l_1 \) and \( l_2 \), and function \( d\text{End}(l_1, l_2) \) gives the vertex that is in \( l_2 \) but not in \( l_1 \).

In the end, the best move operator can be obtained as follows:

\[
\text{best}(M) = \begin{cases} 
m_1 & \text{if } M = [m_1], \\
\text{best}([m_1 m_2 ...]) & \text{else if } \mathcal{F}(X \oplus m_1) < \mathcal{F}(X \oplus m_2), \\
\text{best}([m_2 m_3 ...]) & \text{otherwise},
\end{cases}
\]

where \( M = [m_1 m_2 ...] \) is the vector of candidate moves.

\section*{3.2.6. Applying the move}

After the best move is chosen, it should be applied to the current configuration. Applying the move involves making changes to the network and reassigning the routes for some traffic demands. As explained in Section 3.1,
the grooming solution is obtained by solving the mixed integer programming problem. Although the model is tractable, it is still time consuming to solve it in each iteration.

However, during the local search procedure, only the information of the overload on each lightpath is needed for the evaluation. The details of the traffic grooming only make sense when there is no overload on any lightpath. Fortunately, the gap between the continuous grooming model and the one with binary variables is rather small (as we will see in Section 4.4). Thus it is possible to solve the linear programming relaxation at each iteration to get the estimated overload information when $F$ is greater than zero, and solve the integer program when $F$ equals zero to verify whether the network can satisfy all the traffic demands.

3.3. The matheuristic

In this section, we detail the tabu search procedure of the proposed algorithm. Tabu search (TS) was first proposed by Glover [10, 11] and has since been applied to numerous combinatorial optimization problems. A TS algorithm typically incorporates a tabu list as a recency-based memory structure to guarantee that solutions visited within a certain span of iterations will not be revisited. The algorithm then restricts its attention to moves not forbidden by the tabu list. Additional techniques frequently used in tabu search are diversification and intensification. In particular, the goal of diversification is to escape from the region currently visited by the algorithm, so as to explore new regions in the search space. The intensification is usually achieved by a local search algorithm.

3.3.1. Tabu search procedure

The TS_kNDG procedure has two input parameters: the initial configuration $X_0$ and the number of iterations $nbIter$. We denote by $k$ the number of lightpaths.
The goal of the procedure is to explore the set of configurations by applying a series of moves so as to discover a configuration of minimum cost. As already mentioned in Section 3.2, a configuration $X$ corresponds to any network with $k$ lightpaths. The cost of this configuration is the total overload amount on the network and the set of moves applicable to it is defined by the twist neighborhood.

In order to escape from local optima, the TS$_k$NDG procedure employs a tabu mechanism based on a simple idea. This idea is to forbid a newly added lightpath to be deleted for a short period. A lightpath belonging to the tabu list is said to be tabu. Also, any move $<x,y>$ such that $y$ belongs to the tabu list is declared tabu. After a move $m=<x_m,y_m>$ is performed, lightpath $y_m$ is inserted into the tabu list for $tt$ iterations, where $tt$ (the so-called tabu tenure) is determined by two parameters named $ttMin$ and $ttMax$.

In addition, the tabu status of a move $<x,y>$ can be overridden (by the so-called aspiration criterion) if applying this move leads to a configuration whose cost is smaller than the cost $fBest$ of the best configuration. Finally, we name candidate move any move that is non-tabu or that satisfies the aspiration criterion. In each iteration, the tabu algorithm will select the best candidate move, i.e., the one having the lowest score.

The TS$_k$NDG procedure also employs a diversification technique, which implements several random moves to give more diversity to the algorithm. The pseudocode of the TS$_k$NDG is given in Algorithm 2.

In Algorithm 2, $X$ represents the current configuration of the tabu search procedure. $X$ is first set to the initial configuration $X_0$, which is a parameter of the procedure. In each iteration, a candidate move $<x,y>$ is selected. The selected move is the best candidate move applicable to $X$, i.e., one that has the lowest score (ties are broken randomly). Then, move $<x,y>$ is applied to configuration $X$, i.e., $x$ is removed from $X$ while $y$ is added to $X$. 

17
Algorithm 2: Tabu search procedure for the $k$-NDGP

1: INIT()
2: for iter = 1 \ldots nbIter do
3: \quad < x, y > ← SELECT_MOVE(X)
4: \quad APPLY_MOVE(< x, y >)
5: \quad tt ← RAND(ttMin, ttMax)
6: \quad RENDER_MOVE_TABU(< x, y >, tt)
7: \quad if f(X) = 0 then
8: \quad \quad return (true, X, iter)
9: \quad end if
10: \quad APPLY_DIVERSIFICATION()
11: end for
12: return (false, X, iter)

and the linear programming or mixed integer programming problem is solved depending on the current $F$ (see Section 3.2). Finally, lightpath $y$ is made tabu for $tt$ iterations, where $tt$ is an integer randomly chosen in the range $[ttMin, ttMax]$, and $ttMin$ and $ttMax$ are two parameters of the algorithm. The procedure stops when a zero-cost configuration is discovered or at the end of $nbIter$ iterations.

3.3.2. Diversification technique

In our tabu procedure, a perturbation consists in a series of random moves and the strength of the perturbation is measured by the number of random moves to be performed. Therefore, the perturbation can be seen as a jump in the search space, and the strength of the perturbation as the amplitude of the jump.

The perturbation mechanism is governed by two parameters: strength and perturbPeriod. Parameter strength is used to set the value of the perturbation strength, while parameter perturbPeriod is used to determine when the perturbation is triggered. Also, the algorithm manipulates two values named $X_{best}$ and iterStagnation. $X_{best}$ denotes the best configuration; more precisely, it is the last best configuration visited by the procedure. Therefore, $X_{best}$ is updated whenever $f(X) \leq f(X_{best})$. iterStagnation represents the number of iterations elapsed since the last improvement of the best configu-
ration or since the last perturbation was triggered.

In the tabu algorithm, a perturbation is triggered when $perturbPeriod$ iterations have been performed without improving the best configuration. In this case, a perturbation of strength $strength$ is applied to $X_{best}$.

3.3.3. Parameters of the matheuristic

In addition to the input traffic demands, the input of the matheuristic algorithm consists in the following parameters:

- the parameters $ttMin$ and $ttMax$ used to set the tabu list;
- the parameter $strength$ used to set the value of perturbation strength;
- the parameter $perturbPeriod$ used to determine when a perturbation is triggered;
- the parameter $nbIter$ used for the stopping criterion.

Parameter $nbIter$ depends on the available computing time. Tests performed in order to set the other parameters will be presented in Section 4.2.

4. Computational results

In this section, we present experiments performed in order to evaluate the performance of our algorithm. The matheuristic algorithm was programmed in the Java language. All experiments have been executed on a Linux Server with a 3 GHz CPU and 94 GB of RAM. CPLEX 12.61 was used to solve the linear programming and the integer programming subproblems inside the algorithm.
4.1. Benchmark and algorithms used for comparison

For our tests, we have selected the benchmark proposed in [28]. The benchmark consists of 22 large instances with up to 100 nodes and 500 traffic demands. For each instance, half of the demands have a bandwidth of 1, and the remaining demands have a bandwidth of 2. For all the instances, the lightpath capacity is set to be 32. These instances were randomly generated according to real world scenarios from one of the biggest telecommunication companies in China. The first number in each instance name indicates the size of the node set, and the second number indicates the size of the demand set.

We also provide a comparison between the proposed matheuristic and the two-level iterated local search (TL-ILS) algorithm proposed by [28]. The TL-ILS algorithm was programmed in C++ and run on a PC running Windows 7 with a 3.1 GHz CPU and 6.0 GB of RAM.

4.2. Calibration of the parameters of the tabu search algorithm

In this section, we present preliminary experiments conducted to set the values of the key parameters of the matheuristic:

- Parameters \( ttMin \) and \( ttMax \) determine the range of the tabu tenure \( (tt) \). We have tested two possible values for these parameters: \([1, 5]\) and \([5, 10]\).
- Parameter \( strength \) is used for setting the strength of the perturbation. We have tested two possible values for this parameter: \( 0.1|L| \) and \( 0.2|L| \).
- Parameter \( perturbPeriod \) corresponds to the period used to apply a perturbation. We have tested two values for this parameter: \( 0.5|L| \) and \( |L| \).

For this experiment, a subset of seven instances was used: NDG20_t100.1, NDG20_t200.1, NDG20_t300.1, NDG40_t200.1, NDG40_t200.5, NDG40_t400
and NDG100_t500. The algorithm was run 10 times on each instance and for each parameter setting. The time limit was set to 4 hours for each run. The results of these experiments are presented in Table 1. Each row in the table corresponds to a particular combination of the parameters. The first columns indicate the combinations of the parameters. The last two columns display, for the considered combination, the gap between the lower bound and the average score of the solutions returned by the algorithm over the 10 runs, and the average running time to reach the best solution.

Table 1: Calibration experiments for parameter setting

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Average</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>gap (%)</td>
<td>time (s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5, 10]</td>
<td>[L]</td>
<td>0.1</td>
<td>L]</td>
</tr>
<tr>
<td>[1, 5]</td>
<td>[L]</td>
<td>0.1</td>
<td>L]</td>
</tr>
<tr>
<td>[5, 10]</td>
<td>[L]</td>
<td>0.2</td>
<td>L]</td>
</tr>
<tr>
<td>[1, 5]</td>
<td>[L]</td>
<td>0.2</td>
<td>L]</td>
</tr>
</tbody>
</table>

From Table 1, we notice that the differences in the performance of the algorithm between the different settings are small, which indicates the stability of the matheuristic. However, one observes that the parameter setting ([5, 10], 1.0|L|, 0.1|L|) leads to the best performance.

4.3. Computational results

In this section we compare the proposed matheuristic algorithm (using parameter setting ([5, 10], 1.0|L|, 0.1|L|)) with the TL-ILS algorithm [28] using the full set of instances. The computational results are shown in Table 2. Column LB gives the upward rounded value of the lower bound for each instance while columns Best report the best results obtained by the two algorithms over 10 independent runs. The time limit was set to 8 hours. Column Average reports the average objective value obtained by the proposed
matheuristic algorithm. Columns Time report the average time consumed by each algorithm to get the best result. Column Iteration reports the average number of iterations for the matheuristic to get the best result.

Table 2: Computational results

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>Matheuristic</th>
<th>TL-ILS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>NDG20_t100.1</td>
<td>19</td>
<td>19</td>
<td>19.0</td>
</tr>
<tr>
<td>NDG20_t100.2</td>
<td>19</td>
<td>19</td>
<td>19.0</td>
</tr>
<tr>
<td>NDG20_t100.3</td>
<td>19</td>
<td>19</td>
<td>19.0</td>
</tr>
<tr>
<td>NDG20_t100.4</td>
<td>19</td>
<td>19</td>
<td>19.0</td>
</tr>
<tr>
<td>NDG20_t100.5</td>
<td>19</td>
<td>19</td>
<td>19.0</td>
</tr>
<tr>
<td>NDG20_t200.1</td>
<td>19</td>
<td>23</td>
<td>23.0</td>
</tr>
<tr>
<td>NDG20_t200.2</td>
<td>19</td>
<td>23</td>
<td>23.0</td>
</tr>
<tr>
<td>NDG20_t200.3</td>
<td>19</td>
<td>23</td>
<td>23.0</td>
</tr>
<tr>
<td>NDG20_t200.4</td>
<td>19</td>
<td>22</td>
<td>22.0</td>
</tr>
<tr>
<td>NDG20_t200.5</td>
<td>19</td>
<td>23</td>
<td>23.0</td>
</tr>
<tr>
<td>NDG20_t300.1</td>
<td>30</td>
<td>34</td>
<td>34.5</td>
</tr>
<tr>
<td>NDG20_t300.2</td>
<td>30</td>
<td>34</td>
<td>34.5</td>
</tr>
<tr>
<td>NDG20_t300.3</td>
<td>30</td>
<td>33</td>
<td>33.7</td>
</tr>
<tr>
<td>NDG20_t300.4</td>
<td>30</td>
<td>34</td>
<td>34.9</td>
</tr>
<tr>
<td>NDG20_t300.5</td>
<td>30</td>
<td>34</td>
<td>34.4</td>
</tr>
<tr>
<td>NDG40_t200.1</td>
<td>19</td>
<td>22</td>
<td>22.0</td>
</tr>
<tr>
<td>NDG40_t200.2</td>
<td>19</td>
<td>22</td>
<td>22.3</td>
</tr>
<tr>
<td>NDG40_t200.3</td>
<td>39</td>
<td>39</td>
<td>39.0</td>
</tr>
<tr>
<td>NDG40_t200.4</td>
<td>39</td>
<td>39</td>
<td>39.0</td>
</tr>
<tr>
<td>NDG40_t200.5</td>
<td>39</td>
<td>39</td>
<td>39.0</td>
</tr>
<tr>
<td>NDG40_t400</td>
<td>39</td>
<td>52</td>
<td>52.0</td>
</tr>
<tr>
<td>NDG100_t500</td>
<td>39</td>
<td>62</td>
<td>62.1</td>
</tr>
</tbody>
</table>

From Table 2, one can notice that the difference between the minimum score of the matheuristic algorithm and the TL-ILS algorithm ranges from 0 to 7. The difference tends to increase with the scale of the vertex set and the number of traffic demands. For the NDG20_t300 instances the results produced by the TL-ILS are 12% to 15% larger than the ones produced by the matheuristic. For the smaller instances, the matheuristic also improves the results by 1 or 2 compared to the TL-ILS except for NDG20_t200.4. This indicates that the matheuristic algorithm outperforms the TL-ILS by a very
large margin.

4.4. Analysis of the evaluation technique

For a neighborhood search method, it is particularly important to be able to rapidly evaluate the neighborhood. As described in Section 3.2.5, a relaxed linear programming model is used to estimate the effects of the twist operator. Two experiments were carried out to analyze this evaluation technique on some representative instances, which are NDG20_t100.1, NDG20_t200.1, NDG20_t300.1, NDG40_t200.1, NDG40_t200.5, and NDG40_t400. Note that similar results can be observed on other instances. The time limit for each run was set to 4 hours.

4.4.1. The accuracy of the evaluation technique

As described in Section 3.2.5, after each move the exact integer programming problem is solved only if the cost function reaches zero. In order to observe the accuracy of the evaluation, we have created a variant of the algorithm which solves the IP model after each move. The cost function values given both by the relaxed linear programming model and the integer programming model are recorded and compared to see the differences. The results are shown in Figure 3.

One can observe that for most cases, the linear programming model correctly evaluates the actual cost of the configuration. The plots almost coincide in most of the cases, especially for some easy instances like NDG20_t100.1 and NDG40_t200.5. For other instances we can see that the gap between the original problem and the relaxation is very narrow. These statistics indicate that our proposed evaluation technique works rather well with the twist operator.

4.4.2. The effectiveness of the evaluation technique

Solving the integer programming problem at each iteration will certainly give the algorithm more accuracy, however the computation is very time
Figure 3: Accuracy of the evaluation by the linear programming model consuming and most of the time the detailed information of the configuration is not important. As we mentioned in Section 3.2.5, only the overload amount on each lightpath is needed to produce a new solution of the problem. Thus, the integer programming problem is only solved when the evaluation method predicts a feasible solution, then the algorithm tries to prove that prediction by using integer programming.

Here, we estimate how much time is saved by solving only the relaxed model during the search procedure. The following two strategies are considered: the proposed algorithm which only solves the LP during the search process except when the cost function value equals zero (PAG); the algorithm used in the previous section which will solve an integer programming
One finds that the matheuristic with PAG is faster than that with EAG on difficult instances, but seems worse for simple instances. However, the difference on simple instances is negligible, for both algorithms can get the optimal solution in very short time. For difficult instances the two algorithms have similar performance at an early stage, but the gap gradually widens as \( k \) decreases.

Table 3 shows the detailed results of this experiment. One observes that PAG gets better average and best results than EAG. One also observes that PAG uses more CPU time. However, that is because of the time limit of four
hours. PAG happens to get better results in the time limit. For EAG, it may need much more time to get the same result as PAG.

Table 3: Computational performance comparison

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB</th>
<th>EAG</th>
<th>PAG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
<td>Time (s)</td>
</tr>
<tr>
<td>NDG20, t100.1</td>
<td>19</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>NDG20, t200.1</td>
<td>19</td>
<td>22</td>
<td>22.9</td>
</tr>
<tr>
<td>NDG20, t300.1</td>
<td>30</td>
<td>35</td>
<td>35.4</td>
</tr>
<tr>
<td>NDG40, t200.1</td>
<td>19</td>
<td>22</td>
<td>22.5</td>
</tr>
<tr>
<td>NDG40, t200.5</td>
<td>39</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>NDG40, t400</td>
<td>39</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>NDG100, t1500</td>
<td>39</td>
<td>63</td>
<td>63</td>
</tr>
</tbody>
</table>

From this experiment we learn that the $k$-NDGP gets more and more difficult as $k$ decreases, and the proposed evaluation technique is very helpful when $k$ is small.

5. Conclusion

In this paper, we have proposed a hybrid algorithm embedding LP and MIP into a tabu search for tackling the NDGP. The performance of the proposed algorithm was assessed on instances from the literature. Computational results show that the proposed algorithm is highly competitive in comparison with an existing iterated local search heuristic and it improves the upper bounds for all the instances in the benchmark set. Further analysis indicates that the neighborhood evaluation technique proposed in this paper is essential to the computational efficiency of the proposed algorithm.

Acknowledgments

The research was supported in part by the National Natural Science Foundation of China under grant number 61370183 and the program for New Century Excellent Talents in University (NCET 2013).
References


