ELSEVIER

European Journal of Operational Research 142 (2002) 231–241

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

# Diversification strategies in local search for a nonbifurcated network loading problem

Bernard Gendron [a,b,*], Jean-Yves Potvin [a,b], Patrick Soriano [a,c]

[a] *Centre de Recherche sur les Transports, Université de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal, Qué., Canada H3C 3J7*

[b] *Département d'informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal, Qué., Canada H3C 3J7*

[c] *Service des Méthodes Quantitatives de Gestion, Ecole des Hautes Etudes Commerciales, 3000, Chemin de la Côte-Sainte-Catherine, Montréal, Qué., Canada H3T 2A7*

Received 1 October 2000; accepted 1 June 2001

**Abstract**

This paper examines a variant of the network loading problem, a network design problem found in the telecommunications industry. In this problem, facilities of fixed capacity must be installed on the edges of an undirected network to carry the flow from a central vertex to a set of demand vertices. The objective is to minimize the total installation costs. In this work, the nonbifurcated version of the problem is considered, where the demand at any given vertex must be satisfied through a single path. The proposed heuristics alternate between a construction phase and a local search phase. Each new construction phase, except the first one, is part of a diversification strategy aimed at providing a new starting point for the following local search phase. Different diversification strategies are tested and compared on large-scale instances with up to 500 vertices.


*Keywords:* Network loading problem; Metaheuristics; Local search; Diversification

## 1. Introduction

Network design models find applications in various fields such as computer networks, transportation, manufacturing and telecommunications [1,11,22,24]. In this paper, we study a variant of the network loading problem [21] where facilities of fixed capacity must be installed on the edges of an undirected network to carry the flow from a central vertex to a set of demand vertices. The objective is to minimize the total installation costs. In this work, the nonbifurcated version of the problem is considered, where the demand at any given vertex must be satisfied through a single path. We assume that a fixed and limited number of types of facilities are available (the application considered has nine types of facilities). Each type has some capacity and installation cost. The latter increases with

* Corresponding author. Tel.: +1-514-343-7479; fax: +1-514-343-7121.

*E-mail address:* bernard@crt.umontreal.ca (B. Gendron).

0377-2217/02/$ - see front matter 
PII: S0377-2217(02)00263-1

capacity but with substantial economies of scale. For example, if two types of facilities are available with capacity 1 and $C > 1$, the cost of the second facility would be significantly less than $C$ times the cost of the first facility. This problem is known to be NP-hard, even if simple special cases are considered [19].

The literature on similar problems has mostly focused on exact approaches based on mathematical programming techniques, such as cutting planes and Lagrangean relaxation [2,4,9,10,13,19–21,25]. These approaches are restricted to instances of relatively small size (typically, networks with less than 50 vertices). Heuristic methods are thus indicated to address realistic large-scale instances.

In [3], a tabu search heuristic was proposed to obtain good approximate solutions to this problem. The search was based on a 1-opt neighborhood structure, where the path leading to a given demand vertex in the current solution is replaced by an alternative one. New starting points for the tabu search were provided through a long term adaptive memory scheme [27]. Although the tabu search outperformed a local descent heuristic based on a more complex 2-opt neighborhood (where two paths associated with two different demand vertices are replaced by two new paths), the adaptive memory mechanism was not measured against alternative diversification strategies. Previous studies have however shown that well adapted diversification strategies are an essential element in the development of efficient heuristic approaches [18,29]. The objective of the present paper is twofold: (1) to propose other diversification approaches aimed at providing new starting points for the local search and (2) to compare their performance on large-scale telecommunications instances with up to 500 vertices. We also investigate the behavior of the proposed diversification strategies when a 1-opt local descent is used instead of the tabu search.

The paper is organized as follows. Section 2 presents a mathematical formulation of the problem. In Section 3, the search framework is presented. The two main components of this framework, local search, and diversification, are described in Sections 4 and 5, respectively. Finally, in Section 6, computational results are reported on typical telecommunications data.

## 2. Problem formulation

Let $G = (V, E)$ be an undirected graph where $V$ is the vertex set of cardinality $n$, $E$ is the edge set of cardinality $m$ and $D \subseteq V$ is the subset of demand vertices of cardinality $r$. We further introduce the following notation:

| | |
|---|---|
| $T$ | set of facility types, |
| $P_i$ | set of paths from the central vertex to demand vertex $i$, |
| $d_i$ | demand at vertex $i$, |
| $c_t$ | cost of a facility of type $t$ (per unit of length), |
| $w_t$ | capacity of a facility of type $t$, |
| $l_e$ | length of edge $e$, |
| $u_e$ | initial capacity on edge $e$, |
| $\delta_{ej}$ | 1 if edge $e$ is on path $j$, 0 otherwise. |

We also introduce two sets of variables: the variables $x_{ij}$ which are equal to 1 if path $j \in P_i$ is used to service demand vertex $i$, 0 otherwise; and the design variables $y_{te}$ which are equal to the number of facilities of type $t$ installed on edge $e$. The problem is then formulated as follows:

$$\text{Minimize} \quad \sum_{e \in E} \sum_{t \in T} l_e c_t y_{te}$$

subject to

$$\sum_{i \in D} \left( \sum_{j \in P_i} x_{ij} \delta_{ej} \right) d_i \leqslant u_e + \sum_{t \in T} w_t y_{te} \quad \forall e \in E,$$

$$\sum_{j \in P} x_{ij} = 1 \quad \forall i \in D,$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in D,\ \forall j \in P_i,$$

$$y_{te} \text{ integer} \quad \forall t \in T,\ \forall e \in E.$$

In this model, the first set of constraints specifies that there must be enough capacity on the network to handle the flow on each edge. The second set of constraints requires nonbifurcated flows (i.e., a single path from the central vertex to

each demand vertex). The objective is to minimize the installation cost of any additional capacity to satisfy the demand at each vertex.

## 3. The algorithmic framework

The proposed heuristics alternate between a construction and a local search phase. The first solution is provided by a classical greedy construction heuristic. Thereafter, the construction phase is used for diversification purposes. In this case, the information gathered during the search is used to create a starting solution for the next local search phase (see [5,23] for similar approaches within the ant system framework).

The local search is an iterative neighborhood search which is either a pure descent to the first local optimum or a tabu search (TS). To avoid cycling with TS, a short term memory, known as the tabu list, stores previously visited solutions or components of previously visited solutions. It is then forbidden or tabu to come back to these solutions for a certain number of iterations. The interested reader is referred to [12] for details about the tabu search heuristic.

The algorithmic framework of our heuristics is sketched below.

1. while *stopping criterion of main loop* is not met do:
   **construction phase**
   a. if we are at the start of the algorithm then generate an initial solution $s$ with a constructive heuristic;
   $s^{\text{best}} \leftarrow s$;
   otherwise
   generate $s$ with a diversification procedure;
   **local search phase**
   b. $s^* \leftarrow s$;
   c. while *stopping criterion of local search loop* is not met do:
   generate a neighborhood of $s$ and select the best solution $s'$ in this neighborhood;
   store information about $s'$ for use in the diversification procedure,
   if indicated (see Section 5);
   if $s'$ is better than $s^*$ then $s^* \leftarrow s'$;
   $s \leftarrow s'$;
   d. if $s^*$ is better than $s^{\text{best}}$ then $s^{\text{best}} \leftarrow s'$;
2. output $s^{\text{best}}$.

In this algorithm, variables $s^*$ and $s^{\text{best}}$ correspond to the best solution of the current local search phase and the best overall solution, respectively. The algorithm terminates when the total CPU time spent in the main loop exceeds some value $t_{\max}$. The stopping criterion of the local search loop depends on the method, descent or TS. When descent is used, the loop is stopped when $s$ does not improve over $s^*$. In the case of TS, it is stopped when the number of consecutive iterations without improving $s^*$ has reached a user-supplied value iter$^*$.

In the following section, the neighborhood structure used for the local search phase is presented. Then, Section 5 describes the construction procedures used for initialization and diversification purposes.

## 4. Local search phase

The local search is based on a 1-opt neighborhood structure. Here, a new solution is obtained by replacing the path leading to a given demand vertex $i$ by an alternative one. Each neighbor is evaluated as follows. First, the flow on the selected path is removed and the capacity on the corresponding edges is reduced accordingly (with the flow on the $r-1$ paths leading to the other demand vertices left unchanged). Then, a least-cost, or shortest, path is computed from the central vertex to the demand vertex $i$, where the cost of each edge corresponds to the installation cost of any additional capacity needed to carry the flow $d_i$. Since there is a limited number of facility types, it is easy to determine a priori, even before the start of the algorithm, which facilities must be installed for any flow between 0 and the total demand $\sum_{i=1}^{r} d_i$ (see Section 6.1). Given this information and the flow on the $r-1$ remaining paths, one can then easily determine the additional capacity cost for each edge.

If the least-cost path thus obtained is different from the current one, it is the best alternative path. Otherwise, a second shortest elementary (or loopless) path must be computed. Polynomial-time

exact algorithms are reported in the literature for this problem [14]. However, a heuristic approach based on a simple adaptation of Dijkstra's algorithm [6] is indicated in our context, given that the latter is particularly efficient on sparse networks [28], which are typical in telecommunications applications.

In our adaptation, the scalar label associated with every vertex is replaced by a vector of size 2 whose $p$th element is an upper bound on the length of the $p$th shortest path to that vertex. The method proceeds similarly as in Dijkstra's algorithm, by selecting first the vertex with the smallest label and by updating the labels of its adjacent vertices. This label setting strategy is modified in a simple way to forbid paths with cycles. Once the vertex with minimum temporary label is chosen, any adjacent vertex already found on the tentative path is excluded from further consideration. Clearly, this approach always generates elementary paths. However, it does not necessarily produce the second shortest path, as some paths may be overlooked. It thus constitutes a heuristic approach to the 2-shortest path problem. The complexity of the current implementation of this method is $O(m \log n)$, where $n$ is the number of vertices and $m$ is the number of edges. A more detailed description of this procedure is found in [3].

In spite of the efficiency of this method, the exploration of the entire 1-opt neighborhood remains computationally expensive. Thus, instead of computing the best 1-opt move for each demand vertex, only $\lceil pct_1 \times r \rceil$ demand vertices are considered, where $pct_1$ is a user-supplied parameter.

Both the descent and TS share the same 1-opt neighborhood structure. In the case of TS, however, a tabu list is used to further restrict the subset of demand vertices to which a 1-opt move is applied. At every iteration, the demand vertex associated with the best move is declared tabu for a number of iterations randomly chosen in the interval $[\text{tabu}_{\min}, \text{tabu}_{\max}]$. Hence, the path leading to that vertex cannot be changed for that number of iterations. By associating a tabu status with the vertices rather than the paths themselves, a more restrictive approach is favored, thus reducing the risks of cycling. However, a tabu move may still be applied if it produces a solution which is better than the best one obtained thus far (which is the usual aspiration criterion).

## 5. Construction phase

During the construction phase, a new solution is obtained with one of the two following approaches:

- *Sequence independent construction (SI)*: For each demand vertex $i$, this approach identifies one path to carry the flow from the central vertex to $i$. The order in which demand vertices are considered does not influence the determination of these paths, which are simply combined to obtain a solution.
- *Sequence dependent construction (SD)*: A solution is incrementally constructed by considering the demand vertices in some predefined order. The least-cost path to the current demand vertex is computed, based on the flows and capacities already installed for the previous demand vertices.

In the following subsections, these two methodologies are used to generate new starting solutions for the local search. In Section 5.1, the construction of the first starting solution is presented. Then, Section 5.2 describes four diversification strategies aimed at periodically restarting the search process.

### 5.1. Initialization

An example of an SI procedure is the initialization approach implemented in [3], which consists in selecting the shortest paths, with respect to the edge lengths, from the central vertex to each demand vertex. This procedure suffers from a serious drawback since, even if it may find the best path to reach each vertex individually, the solution obtained will typically be quite far from the optimum, as many paths are likely to share common edges. Consequently, we implement an initialization procedure that exploits the SD methodology. Here, the demand vertices are sorted in non increasing order of demand. This order is motivated by the cost structure, which suggests that substantial econo-

mies of scale can be achieved if higher capacity facilities are installed first. In this way, it is expected that smaller demand vertices will be connected for "free", by exploiting the unused portion of the installed capacity on the network. It is important to note that the edge costs are adjusted at each iteration of the SD procedure, since the paths chosen for the previous demand vertices determine what facilities should be installed to carry the flow to the current demand vertex.

### 5.2. Diversification strategies

Four different diversification strategies will be presented in the following. The first one is the original method reported in [3]. The three others are alternative approaches motivated by the SD construction procedure.

#### 5.2.1. Adaptive memory

The adaptive memory (AM) diversification method, proposed in [3], is an example of an SI procedure. Here, the adaptive memory is partitioned into $r$ compartments, one for each demand vertex. A compartment stores different paths from the central vertex to the associated demand vertex, where the paths are collected from the best solutions visited during the search. The "score" of a path corresponds to the objective value of the best solution containing that path. When a new current solution is obtained by the local search, the paths leading to each demand vertex are stored in their corresponding compartment if: (1) the compartment is not full or (2) the path is better than the worst path in the compartment, in which case the new path takes its place. To generate a new starting solution, a path is taken from each compartment through a probabilistic selection process. In each compartment, the selection probability of a path is proportional to its score. Consequently, paths with better scores are more likely to be selected. This biased selection process, which is also used in Sections 5.2.2 and 5.2.4, is described in detail in Appendix A.

#### 5.2.2. Greedy multistart

The SD initialization procedure described in Section 5.1 can be slightly modified to provide different starting points during the search. For example, instead of selecting the least-cost path associated with the current vertex, a path could be selected at random from a list of paths that are not "too far" from the best one (using, for example, mechanisms found in greedy randomized adaptive search procedures or GRASP [7,8,26]). A variant of this idea, which is easier to implement, is to randomize the selection of the next demand vertex. That is, during the iterative SD construction procedure, the selection of the next vertex is probabilistically biased towards those with higher demands, using the procedure described in Appendix A.

We have implemented the latter idea within our greedy multistart (GM) diversification strategy. Starting from the best solution found during the current local search phase, $s^*$, a number of randomly chosen paths are first removed from the solution (i.e., the flows and capacities on the corresponding edges are reduced accordingly). In the current implementation, $\lceil pct_{GM} \times r \rceil$ paths are removed, where $pct_{GM}$ is a user-supplied parameter. Then, the partial solution is reconstructed through the SD greedy construction procedure, where the choice of the next vertex, among those which are not yet connected to the central vertex, is probabilistically biased towards vertices with higher demands. The solution obtained represents the new starting point for the next local search phase. This partial removal/reconstruction process provides a means for perturbing $s^*$ while, at the same time, preserving the general characteristics of this solution. The fraction of the paths that are removed represents a trade-off between the *exploitation* of good characteristics of $s^*$ and the *exploration* of new solutions. At both extremes, $s^*$ is kept as it is (total exploitation) or a new solution is reconstructed from scratch (total exploration).

#### 5.2.3. 2-Opt neighborhood (2N)

Starting from $s^*$, the best solution obtained during the current local search phase, a relatively similar solution can be obtained through a 2-opt move, where two paths associated with two demand vertices $i$ and $j$ are replaced by two alternative paths, as in the following procedure:

1. find the best alternative path to reach vertex $i$ and update the flows and capacities on the edges accordingly;
2. find the best alternative path to reach vertex $j$ and update the flows and capacities accordingly;
3. store the solution obtained in $s_1$;
4. repeat steps 1 and 2 by interchanging vertices $i$ and $j$;
5. store the solution obtained in $s_2$;
6. take the best solution between $s_1$ and $s_2$.

It is possible to explore the entire 2-opt neighborhood of $s^*$, obtained by considering all pairs of demand vertices $i$ and $j$. However, since our goal is to perform diversification, only a small fraction $pct_{2N}$ of the 2-opt neighborhood is considered, where $pct_{2N}$ is a user-supplied parameter. Basically, for each demand vertex, $\lceil pct_{2N} \times (r-1) \rceil$ vertices are chosen among the other vertices to form the required pairs. A move is then performed by selecting the best solution in this reduced neighborhood. Although a single move could be done, it is good practice to perform a fixed (usually small) number $iter_{2N}$ of such 2-opt moves (each time applied to the solution obtained from the previous move) to obtain a solution which is sufficiently different from $s^*$. It is worth noting that this 2-opt neighborhood (2N) diversification strategy, combined with the 1-opt local search, is reminiscent of variable neighborhood search [15–17].

This diversification strategy could be extended in a similar way to higher order $k$-opt neighborhoods. However, preliminary experiments have shown that even the evaluation of a very small fraction of the 3-opt neighborhood is too time-consuming with regard to its potential benefits. These higher order neighborhoods can nevertheless be exploited with profit, as illustrated in the following subsection.

*5.2.4. Greedy k-opt (Gk)*

The idea of using general $k$-opt moves is interesting, especially if only one such move is performed, as in the following procedure:

1. choose a value for $k$ in the interval $[\lceil pct^1_{Gk} \times r \rceil, \lceil pct^2_{Gk} \times r \rceil]$;
2. remove $k$ randomly chosen paths from $s^*$ and update the flows and capacities on the edges accordingly;
3. reinstall $k$ alternative paths to form a complete solution.

In this procedure, $pct^1_{Gk}$ and $pct^2_{Gk}$ are user-defined parameters. In step 3, the SD construction procedure is used, where the selection of the next vertex (among those not yet connected to the central vertex) is probabilistically biased towards those with higher demands. Note that this procedure, unlike the 2-opt move procedure described in Section 5.2.3, makes no attempt at forbidding the reconstruction of the same solution $s^*$. This could be easily fixed, but this situation is unlikely to occur given the probabilistic order used during the reconstruction. Furthermore, even if the same solution is visited twice, the subsequent trajectories are likely to be different, due to the randomization.

In its basic version, this greedy $k$-opt (G$k$) strategy involves the evaluation of a single move, which identifies one neighbor of $s^*$. However, it is certainly possible (although computationally more expensive) to consider a fixed, but relatively small, number of $k$-opt neighbors of $s^*$. Then, the search could restart from the best solution among these neighbors. It is even possible to go one step further. A pool containing the best solutions generated in this way could be maintained. As the search proceeds, this pool would typically contain solutions obtained from previous diversification steps. By using the best solution found in this pool to restart the search (which is not necessarily the best solution obtained from $s^*$), the entire history of the search, not only the recent one, would be considered. This approach can be seen as a form of adaptive memory [30], but restricted to diversification moves. In the remainder, it is assumed that the number of moves applied to $s^*$ and the size of the pool are both equal to *size*, a user-supplied parameter. Clearly, this does not need to be the case in general, but this simplification has little impact, if any, on the performance of the approach.

Precisely, the pool of elite solutions is managed as follows. Initially, the pool is empty. At the first diversification step, *size* new solutions are inserted

in the pool, and the best one is selected and removed from it to restart the search. At subsequent diversification steps, the *size* new solutions produced with $k$-opt moves are merged with the *size*-1 solutions already in the pool, and only the *size* best are kept. Then, the best one is selected and removed from the pool to provide a new starting point for the local search.

## 6. Computational results

The diversification strategies presented in the previous section were tested and compared on randomly generated telecommunications networks with characteristics frequently observed in practice. Before reporting the numerical results, the cost data and network topologies used in the experiments are first introduced.

### 6.1. Cost data

The installation costs for the different types of facilities come from a real-world application. Since these costs are confidential, they are shown in generic form in Table 1. As observed in this table, it is not always optimal to use the smallest single facility which can carry the flow. For a flow of 100, for example, it is better to use facilities 7 and 1 with a total capacity of 102 and a total cost of 3.58, rather than facility 8 with capacity 144 and a cost of 4.37. This difficulty can be alleviated through the following observations:

- except for facility 9, it is never profitable to use the same facility twice (for example, installing facility 1 twice provides 12 units of capacity, while the same capacity is obtained at lower cost using facility 2);
- facility 9 is mandatory for flows of 216 or more, as it is less expensive than any other combination of smaller facilities.

Table 1
Facility types

| Facility type | Capacity | Cost (per unit of length) |
|---|---|---|
| 1 | 6 | 0.55 |
| 2 | 12 | 0.73 |
| 3 | 24 | 1.03 |
| 4 | 36 | 1.39 |
| 5 | 8 | 1.67 |
| 6 | 72 | 2.31 |
| 7 | 96 | 3.03 |
| 8 | 144 | 4.37 |
| 9 | 216 | 6.33 |

The special structure of these costs can thus be exploited to a priori determine the facilities to be installed for any flow between 0 and the total demand $\sum_{i=1}^{r} d_i$:

- for flows over 216, facility 9 is installed as many times as needed until a residual flow under 216 is obtained;
- for flows between 1 and 215, all possible combinations of *different* facilities are considered (except facility 9), and the combination of lowest cost is kept.

### 6.2. Randomly generated networks

General undirected networks with $n = 200$ and 500 vertices were randomly generated to test our algorithms. These networks are sparse and quasi-planar, like those found in practice, with edge densities (i.e., fraction of all possible edges) varying between 1% and 10%. More precisely, four types of networks were considered and 10 different instances were generated for each type. The four network types are the following:

- $n = 200$, $r = 100$ and edge density $= 1\%$;
- $n = 200$, $r = 100$ and edge density $= 10\%$;
- $n = 500$, $r = 250$ and edge density $= 1\%$;
- $n = 500$, $r = 250$ and edge density $= 10\%$.

A demand is thus associated with 50% of the vertices. The demand vertices fall into three categories: category 1 with 10–20 units of demand; category 2 with 40–60 units; category 3 with 80–100 units. The demand vertices are evenly distributed among the three categories. The problems come with some equipments already installed. The total capacity of the network, which is the sum of the capacities over all edges of the network, is twice the total demand. This capacity is

then uniformly distributed over all edges, with a random perturbation on each edge (i.e., ±0–20%).

### 6.3. Experiments

Preliminary experiments were conducted to determine appropriate parameter values for the algorithms. Since our objective is to compare different diversification strategies, the parameter values for the local search phase were the same in all experiments. Preliminary tests allowed us to identify the following values:

- fraction of 1-opt neighborhood at each descent iteration, $pct_1 = 25\%$;
- fraction of 1-opt neighborhood at each TS iteration, $pct_1 = 10\%$;
- tabu tenure: $\text{tabu}_{\min} = 5$, $\text{tabu}_{\max} = 9$;
- TS stopping criterion: maximum number of consecutive iterations without improvement, $\text{iter}^* = 10$.

Note that the values of $pct_1$ are different for descent and TS. Although the two methods share the same neighborhood structure, their search mechanisms are different, and therefore, the "best" fraction of the neighborhood to be considered at each iteration ($pct_1$) may also differ significantly. We comment further on this issue at the end of this section.

These values are not necessarily "optimal" for each diversification strategy, but they exhibit a good overall performance and allow a fair comparison. Each diversification strategy has also been calibrated. For our tests, the following parameter values were used:

- AM
  - compartment size: 10;
- GM
  - fraction of paths removed, $pct_{\text{GM}} = 20\%$;
- 2N
  - fraction of neighborhood explored, $pct_{\text{2N}} = 0.5\%$;
  - number of 2-opt iterations, $\text{iter}_{\text{2N}} = 3$;
- G$k$
  - $k \in [\lceil pct^l_{\text{G}k} \times r \rceil, \lceil pct^2_{\text{G}k} \times r \rceil]$ with $pct^1_k = 5\%$, $pct^2_k = 20\%$;
  - pool size, $size = 1, 10$.

Our experiments were run on a 64-processor, 64-GB Sun Enterprise 10000, with each processor operating at 400 MHz. Because all heuristics contain stochastic features, three runs of each method were performed on each problem instance. Tables 2 and 3 compare the results obtained with the four diversification strategies on the four types of networks. For 200-vertex instances, the total CPU time was set to $t_{\max} = 1$ hour, while for 500-vertex, this value was increased to 2.5 hours. For each diversification strategy, three measures, averaged over the 10 corresponding instances and the three individual runs, are reported: $Z(s^{\text{best}})$, the average value of the best solution found; $\text{iter}_{\text{div}}$, the average number of diversification steps and $\text{iter}_{\text{loc}}$ the average number of local search iterations performed between two diversification steps.

In addition to the four diversification strategies presented in Section 5, the results of two other approaches are reported: (1) a pure TS method, with no diversification ("TS"), which is obtained by setting $\text{iter}^*$ to a very large number and (2) G$k$ with $k$ fixed to 2 ("G2"). The G2 strategy was introduced to better evaluate the impact of the $k$ value in G$k$ and also to provide a comparison with 2N, which exhibits a more "aggressive" approach by selecting the best move in a fraction of the 2-opt neighborhood (rather than choosing 2-opt moves at random). Thus, a total of 15 different heuristics are compared. In addition, Tables 2 and 3 show the average value of the first starting solution produced by the initialization procedure. Note that each identifier *XX–YY* in these tables refers to diversification strategy *XX* combined with local search *YY* (either descent or TS).

A number of conclusions may be drawn from these results:

- For a given amount of CPU time, it is often better to restart more frequently with a good diversification strategy, like G$k$, and to optimize less thoroughly using a simple descent (rather than the opposite, that is, restart less frequently and optimize more thoroughly with tabu search).

Table 2
Results on networks with 200 vertices (total CPU time of 1 hour)

| Heuristic | Density = 1% | | | Density = 10% | | |
|---|---|---|---|---|---|---|
| | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ |
| Initialization | 16,251 | – | – | 11,146 | – | – |
| TS | 15,910 | 1 | 45,208 | 10,896 | 1 | 31,219 |
| AM-descent | 15,917 | 1017 | 45 | 10,954 | 779 | 39 |
| AM-TS | 15,918 | 1449 | 77 | 10,933 | 1001 | 76 |
| GM-descent | 15,941 | 3897 | 11 | 10,966 | 2579 | 9 |
| GM-TS | 15,949 | 2912 | 37 | 10,956 | 1811 | 36 |
| 2N-descent | 15,839 | 4753 | 4 | 10,879 | 2880 | 5 |
| 2N-TS | 15,880 | 12,912 | 10 | 10,940 | 7879 | 10 |
| G2-descent (size = 1) | 15,870 | 47,865 | 1 | 10,914 | 30,677 | 1 |
| G2-TS (size = 1) | 15,855 | 13,150 | 10 | 10,893 | 8213 | 10 |
| G2-descent (size = 10) | 15,876 | 24,329 | 2 | 10,910 | 17,077 | 1 |
| G2-TS (size = 10) | 15,870 | 10,264 | 12 | 10,898 | 6622 | 11 |
| G*k*-descent (size = 1) | 15,800 | 16,786 | 3 | 10,851 | 10,871 | 3 |
| G*k*-TS (size = 1) | 15,788 | 8397 | 15 | 10,889 | 5036 | 15 |
| G*k*-descent (size = 10) | 14,661 | 12,091 | 1 | 10,407 | 7918 | 1 |
| G*k*-TS (size = 10) | 15,091 | 7082 | 10 | 10,639 | 4494 | 10 |

Table 3
Results on networks with 500 vertices (total CPU time of 2.5 hours)

| Heuristic | Density = 1% | | | Density = 10% | | |
|---|---|---|---|---|---|---|
| | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ |
| Initialization | 43,492 | – | – | 24,310 | – | – |
| TS | 42,655 | 1 | 11,049 | 23,752 | 1 | 6018 |
| AM-descent | 42,994 | 101 | 109 | 23,928 | 57 | 96 |
| AM-TS | 42,835 | 169 | 163 | 23,845 | 88 | 157 |
| GM-descent | 42,944 | 437 | 30 | 23,970 | 150 | 35 |
| GM-TS | 42,857 | 396 | 82 | 23,885 | 131 | 101 |
| 2N-descent | 42,651 | 384 | 4 | 23,724 | 157 | 5 |
| 2N-TS | 42,756 | 3489 | 10 | 23,910 | 1482 | 10 |
| G2-descent (size = 1) | 42,670 | 13,871 | 1 | 23,826 | 6127 | 1 |
| G2-TS (size = 1) | 42,624 | 3671 | 10 | 23,823 | 1578 | 10 |
| G2-descent (size = 10) | 42,683 | 13,794 | 1 | 23,842 | 6092 | 1 |
| G2-TS (size = 10) | 42,630 | 3644 | 10 | 23,828 | 1563 | 10 |
| G*k*-descent (size = 1) | 42,726 | 1995 | 7 | 23,736 | 610 | 10 |
| G*k*-TS (size = 1) | 42,684 | 1262 | 27 | 23,694 | 386 | 38 |
| G*k*-descent (size = 10) | 42,312 | 3093 | 2 | 23,606 | 1256 | 2 |
| G*k*-TS (size = 10) | 42,463 | 1826 | 12 | 23,580 | 716 | 13 |

- The 1-opt neighborhood is not "rich" enough, on the long term, to identify improving solutions. This observation is supported by the values of $iter_{loc}$ observed for the best diversification strategies: they tend towards the minimum number of iterations performed during any single local search phase, which, for TS, is equal to $iter^* = 10$, and for descent, is equal to 1 (no improving solution is found at the very first iteration).
- The starting points provided by AM are not very effective, given the relatively large number of local search iterations ($iter_{loc}$) after each diversification step. This result was expected, since

AM is based on the SI construction approach, where independently computed paths are merged together to form a solution. Overall, the AM approach outperforms only the GM strategy.
- Among all diversification strategies, GM is the worst. It is also the only one which does not exploit the history of the search.
- The 2N-descent method is more effective than the G2-descent variant. However, it is the opposite when 2N is combined with TS. Since both 2N and G2 are based on the same 2-opt neighborhood structure, it is beneficial when using descent as the local search phase to spend some additional effort to determine good starting points through a partial exploration of the neighborhood. When TS is used, this effect vanishes as the latter is less sensitive to the starting point.
- The G*k* diversification strategy is significantly better than the others. In particular, G*k* outperforms both 2N and G2, which are based on 2-opt neighborhoods. Clearly, general *k*-opt moves differ more significantly than 2-opt moves from the 1-opt moves used in the local search phase, and provide new and better opportunities for diversification. It should also be noted that, unlike G2, the G*k* diversification strategy is greatly improved when a pool of elite solutions is used.

Note also that increasing the neighborhood size to 25% for the *XX*-TS methods, as for the corresponding *XX*-descent methods, does not provide any improvement over the values reported in Tables 2 and 3. In fact, it is exactly the opposite. On the 500-vertex problems with densities of 1% and 10%, for example, G*k*-TS with a pool size of 10 produces average solution values of 42,544 and 23,636, respectively. The same degradation is observed for the other methods based on TS, when the neighborhood size is similarly increased. Thus, increasing the neighborhood size is indicated only if the search is performed in a restricted region of the search space, as in pure descent methods.

In summary, the G*k* diversification approach is the most effective because it combines:

- complex moves based on large-scale neighborhoods;
- the generation of multiple solutions at each diversification step;
- the exploitation of the entire history of the search (not only the recent one), via a pool of elite solutions generated by previous diversification steps.

## 7. Conclusion

In this paper, new heuristics were reported for the nonbifurcated version of the network loading problem presented in [3]. These heuristics alternate between a construction and a local search phase. The construction phase is used to generate the initial starting solution and to provide restart points during the search process, for diversification purposes. Four diversification strategies were compared on large-scale instances with up to 500 vertices. The results indicate that the best diversification strategy combines three desirable features: (1) it generates solutions that are significantly different from those obtained during the local search phase, due to a more complex neighborhood structure, (2) it generates many different solutions at each diversification step and (3) it keeps track of the entire history of the search by maintaining a pool of elite solutions produced by previous diversification steps.

## Acknowledgements

Special thanks are due to François Guertin and Serge Bisaillon for their help in implementing and testing our algorithms. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is hereby gratefully acknowledged.

## Appendix A

In this appendix, we describe how a random selection of one element in a list of elements sorted from best to worst is performed. The best element gets value *Max*, while the worst gets value *Min*.

The values associated with the other elements are equally spaced between *Min* and *Max*. More precisely, assuming that we have $d$ elements in the list (with $d > 1$), the value $V_i$ for the element of rank $i$ is

$$v_i = Max - (Max - Min) \times \frac{i-1}{d-1}, \quad 1 \leqslant i \leqslant d.$$

The selection probability $p_i$ of the element of rank $i$ is then

$$p_i = \frac{v_i}{\sum_{j=1}^{d} v_j}, \quad 1 \leqslant i \leqslant d.$$

Assuming that $Min + Max = 2$, the selection bias in favor of the best elements can be increased by setting the *Max* value closer to 2, or reduced by setting its value closer to 1. In our implementations, the values $Min = 0.5$ and $Max = 1.5$ were used.

## References


[1] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, Network design, in: M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, New York, 1998, pp. 311–334.

[2] F. Barahona, Network design using cut inequalities, SIAM Journal on Optimization 6 (1996) 823–837.

[3] D. Berger, B. Gendron, J.-Y. Potvin, S. Raghavan, P. Soriano, Tabu search for a network loading problem with multiple facilities, Journal of Heuristics 6 (2) (2000) 253–267.

[4] D. Bienstock, O. Günlük, Capacitated network design – polyhedral structure and computation, INFORMS Journal on Computing 8 (1996) 243–259.

[5] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ant colonies, in: Proceedings of the European Conference on Artificial Life, Paris, France, 1991, pp. 134–142.

[6] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959) 269–271.

[7] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.

[8] P. Festa, M.G.C. Resende, GRASP: An annotated bibliography, Technical Report, AT&T Labs Research, Florham Park, NJ, 2000.

[9] B. Gavish, I. Neuman, A system for routing and capacity assignment in computer communication networks, IEEE Transactions on Communications (1989) 360–366.

[10] B. Gavish, A. Altinkemer, Backbone network design tools with economic tradeoffs, ORSA Journal on Computing 2 (1990) 236–252.

[11] B. Gendron, T.G. Crainic, A. Frangioni, Multicommodity capacitated network design, in: B. Sansó, P. Soriano (Eds.), Telecommunications Network Planning, Kluwer Academic Publishers, Dordrecht, 1998, pp. 1–19.

[12] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.

[13] O. Günlük, A branch-and-cut algorithm for capacitated network design problems, Mathematical Programming A 86 (1999) 17–39.

[14] E. Hadjiconstatinou, N. Christofides, An efficient implementation of an algorithm for finding $K$ shortest simple paths, Networks 29 (1999) 88–101.

[15] P. Hansen, N. Mladenović, Variable Neighborhood Search: Principles and Applications, Les Cahiers du GERAD G-98-20, Montréal, Canada, 1998.

[16] P. Hansen, N. Mladenović, D. Perez-Brito, Variable Neighborhood Decomposition Search, Les Cahiers du GERAD G-98-53, Montréal, Canada, 1998.

[17] P. Hansen, N. Mladenović, Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization, Les Cahiers du GERAD G-2000-03, Montréal, Canada, 2000.

[18] J.P. Kelly, M. Laguna, F. Glover, A study of diversification strategies for the quadratic assignment problem, Computers and Operations Research 21 (1992) 885–893.

[19] T.L. Magnanti, P. Mirchandani, Shortest paths, single origin–destination network design and associated polyhedra, Networks 23 (1993) 103–121.

[20] T.L. Magnanti, P. Mirchandani, R. Vachani, The convex hull of two core capacitated network design problems, Mathematical Programming 60 (1993) 233–250.

[21] T.L. Magnanti, P. Mirchandani, R. Vachani, Modeling and solving the two-facility capacitated network loading problem, Operations Research 43 (1995) 142–157.

[22] T.L. Magnanti, R.T. Wong, Network design and transportation planning: Models and algorithms, Transportation Science 18 (1984) 1–55.

[23] V. Maniezzo, Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, INFORMS Journal on Computing 11 (1999) 358–369.

[24] M. Minoux, Network synthesis and optimum network design problems: Models, solution methods and applications, Networks 19 (1989) 313–360.

[25] T. Ng, D. Hoang, Joint optimization of capacity and flow assignment in a packet-switched communication network, IEEE Transactions on Computing 35 (1987) 202–209.

[26] M.G.C. Resende, Greedy Randomized Adaptive Search Procedures (GRASP), Encyclopedia of Optimization, Kluwer Academic Publishers, Dordrecht, 2000.

[27] Y. Rochat, É.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.

[28] D.R. Shier, On algorithms for finding the $k$-shortest paths in a network, Networks 9 (1979) 195–214.

[29] P. Soriano, M. Gendreau, Diversification strategies in tabu search algorithms for the maximum clique problem, Annals of Operations Research 63 (1996) 189–207.

[30] É.D. Taillard, L.M. Gambardella, M. Gendreau, J.-Y. Potvin, Adaptive memory programming: A unified view of metaheuristics, European Journal of Operational Research 135 (1) (2001) 1–16.