Adaptive Memory in Multistart Heuristics for Multicommodity Network Design

Daniel Aloise[1] and Celso C. Ribeiro[2]

[1] Department of Production Engineering,
Universidade Federal do Rio Grande do Norte,
Natal, RN 59072-970, Brazil.
aloise@gmail.com

[2] Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.
celso@ic.uff.br

Abstract. This paper focuses on the use of different memory strategies to improve multistart methods. A network design problem in which the costs are given by discrete stepwise increasing cost functions of the capacities installed in the edges is used to illustrate the contributions of adaptive memory and vocabulary building strategies. Heuristics based on shortest path and maximum flow algorithms are combined with adaptive memory in order to obtain an approximate solution to the problem in the framework of a multistart algorithm. Furthermore, a vocabulary building intensification mechanism supported by the resolution of a linear program is also explored. Numerical experiments have shown that the proposed algorithm obtained the best known solutions for some instances in the literature. These results show the contribution of each memory component and the effectiveness of their combination.

Keywords: Adaptive memory, vocabulary building, multistart, multicommodity flows, network design, heuristics.

1 Introduction

Flexible and adaptive memory techniques have been the source of a number of developments to improve multistart methods, which otherwise would simply resort to random restarts. Different strategies have been devised to intelligently explore memory information in this context. Their main underlying principle consists in exploring information collected and updated along the search to improve the performance of different constructive and local search methods. Among the different strategies used to explore memory techniques, we may find adaptive memory [9, 27, 34, 35], vocabulary building [6, 16, 18], and path-relinking [17, 24, 31], among others. In this paper, we show how two of these principles can be blended and combined into more efficient implementations of multistart metaheuristics.

A network design problem in which the costs are given by discrete stepwise increasing cost functions of the edge capacities is used to illustrate the contributions of adaptive memory and vocabulary building strategies.

Multicommodity network design problems are basic models in a wide range of applications such as communication networks, transportation problems, and logistics [10, 14, 25, 28]. These problems are widely studied for linear [22], convex [26], fixed charge or nonlinear concave, continuous and differentiable cost functions [5, 28]. We consider a multicommodity network design problem with discrete stepwise cost functions of the edge capacities, whose decision version is NP-complete even if there is only one single commodity [8]. Cutting plane methods were firstly investigated by [37, 38]. Gabrel et al. [11] proposed a Benders decomposition approach to the problem, leading to a heuristic implementation described in [12]. Upper bounds can also be obtained by a D.C. programming approach proposed in [4].

We propose a memory-based multistart heuristic exploring large neighborhoods for the problem above. A multistart heuristic is an iterative two-phase procedure consisting of the repeated application of the following steps: (i) construction of a solution and (ii) improvement of the latter by local search, until some stopping condition is met. Basic multistart algorithms are memoryless and do not take full advantage of information gathered during the search. The use of memory mechanisms already proposed in the literature (e.g. [9, 18]), but not yet sufficiently explored in practice, can strongly improve their performance, as illustrated by the computational results reported in this paper.

The mathematical formulation of the problem is given in the next section. Constructive and local search heuristics are described in Section 3. The framework of a memory-based multistart algorithm and its main memory components are presented in Section 4. Computational results are reported in Section 5 and compared with those in the literature. Concluding remarks are drawn in the last section.

2 Problem formulation

Let $G = [N, U]$ be an undirected graph, where N is the set of nodes and U is the set of edges. Furthermore, let $D = \{D_1, \ldots, D_K\}$ be a set of commodities or traffic demands. Demand $D_k = \{s_k, t_k, f_k\}$ is characterized by a triple where f_k is the flow to be sent from the source node s_k to the destination node t_k, for each $k = 1, \ldots, K$.

For every edge $u = (i, j) \in U$, we denote by x_u the installed capacity in edge u and by $\phi_u(x_u)$ its associated installation cost. Let $V_u = \{v_u^0, \ldots, v_u^{q(u)}\}$ be a finite set of values, with $0 = v_u^0 < \ldots < v_u^{q(u)} = \beta_u$. For every $\ell = 0, \ldots, q(u)$, let $\gamma_u^\ell = \phi_u(v_u^\ell)$, with $0 = \gamma_u^0 < \ldots < \gamma_u^{q(u)}$. Therefore, for every edge $u \in U$:

$$\phi_u(x_u) = \begin{cases} 0, \text{ if } x_u = 0 \\ \gamma_u^i, \text{ if } x_u \in (v_u^{i-1}, v_u^i] \text{ for some } i = 1, \ldots, q(u), \end{cases}$$

as illustrated in Figure 1.

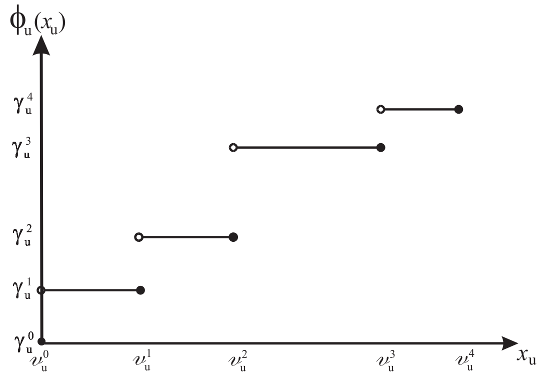

Fig. 1. Cost function $\phi_u(x_u)$ with $q(u) = 4$.

The multicommodity flow network design problem considered in this work consists in determining the capacities $x_u \geq 0$ to be installed in each edge $u \in U$, so as that all commodities can be simultaneously routed through the network, satisfying capacity, demand, and flow conservation constraints and minimizing the sum of the installation costs over all edges.

Let $\Gamma(i) = \{j : (i,j) \in U\}$ be the set of nodes adjacent to i in G. For each edge $u = (i,j) \in U$, let $\psi_{ij}^k \geq 0$ (resp. $\psi_{ji}^k \geq 0$) be the flow of commodity D_k from node i to j (resp. from node j to i) through edge $u = (i,j) \in U$. The problem can be modeled by the following undirected flow formulation as follows:

$$\min \phi(x) = \sum_{u=(i,j)\in U} \phi_u(x_u) \tag{1}$$

subject to:

$$\sum_{j\in\Gamma(i)} (\psi_{ij}^k - \psi_{ji}^k) = \left\{ \begin{array}{ll} +f_k, & i = s_k \\ 0, & i \neq s_k, i \neq t_k \\ -f_k, & i = t_k \end{array} \right\} \qquad \forall i \in N \quad \forall k = 1,\ldots,K \tag{2}$$

$$0 \leq x_u \leq \beta_u \qquad \forall u = (i,j) \in U \tag{3}$$

$$x_u = \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k) \qquad \forall u = (i,j) \in U \tag{4}$$

$$\psi_{ij}^k, \psi_{ji}^k \geq 0 \qquad \forall u = (i,j) \in U, \quad \forall k = 1,\ldots,K. \tag{5}$$

The objective function (1) consists in minimizing the total installation costs over all edges. Constraints (2) represent flow conservation and demand satisfaction, for every node and every commodity. Constraints (3) establish the upper bounds to the installed capacity in each edge. Constraints (4) define that the installed capacity in each edge is equal to the sum of the flows going through it over all commodities. Inequalities (5) are the non-negativity constraints on the flows. We notice that once the variables x_u are fixed for every edge $u \in U$, then the existence of a feasible routing can be checked in polynomial time by solving a linear program.

Without loss of generality, constraints (3) and (4) in the above model (1) to (5) can be simultaneously replaced by

$$x_u \in V_u = \{v_u^0, \ldots, v_u^{q(u)}\} \qquad \forall u = (i,j) \in U \tag{6}$$

$$x_u \geq \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k) \qquad \forall u = (i,j) \in U, \tag{7}$$

due to the stepwise increasing nature of the cost functions. In fact, for any feasible solution and for every edge $u \in U$, an installed capacity $x_u \in (v_u^{i-1}, v_u^i)$ for some $i = 1, \ldots, q(u)$ can be rounded up to $x_u = v_u^i$ preserving its cost and feasibility (cf. [13], page 128). The heuristics proposed in this paper will consider model (1) to (5) with constraints (3) and (4) replaced by (6) and (7), therefore restricting the search to solutions where the capacity x_u of each edge $u \in U$ is restricted to assume a value in the finite set $V_u = \{v_u^0, \ldots, v_u^{q(u)}\}$ of capacity steps. Thereafter, the resulting problem is referred to as the *discrete cost multicommodity flow problem* (DCMFP).

The results reported in the literature [29] show that the application of exact algorithms to this problem is limited in practice. Gabrel et al. [11] presented an integer programming formulation of DCMFP and proposed an exact constraint generation algorithm based on inequalities

$$\sum_{u \in U} \lambda_u x_u \geq \sum_{k=1}^{K} d_k l_k^*(\lambda),$$

where $l_k^*(\lambda)$, for $\lambda \in \mathbb{R}_+^{|U|}$, is the length of the shortest path joining s_k and t_k in G when the length of edge u is given by $\lambda_u \geq 0$, for each edge $u \in U$. These inequalities are often referred to as *metric inequalities*. They are indeed valid since if a linear design cost λ_u is associated with each edge $u \in U$, then a lower bound on the cost of routing demand D_k is given by the length of the shortest path between s_k and t_k multiplied by f_k. Therefore, $\sum_{k=1}^{K} d_k l_k^*(\lambda)$ is a valid lower bound for $\sum_{u \in U} \lambda_u x_u$.

Later, Mrad and Haouari [30] proposed improvements to the constraint generation algorithm in [11] in order to accelerate its convergence. Such improvements were based on efficiently finding violated bipartition inequalities, i.e., metric inequalities with $\lambda_u = 1$ for all edges u separating two subsets of nodes in G and $\lambda_u = 0$ for all other edges, as well as on the use of an exact algorithm based on LP duality for separating metric inequalities. These improvements led to the resolution of instances with up to 50 nodes and 100 edges, with three steps in each edge cost function. However, real-life applications to power systems optimization presented in [32] typically involve much larger instances. In [12], Gabrel et al. proposed greedy heuristics as well as a heuristic implementation of their constraint generation method in order to tackle instances not solvable by exact methods in reasonable computation times.

3 Heuristics

3.1 Construction

The constructive heuristic determines the capacities x_u to be installed in each edge $u \in U$. For every edge $u \in U$, we define $i(u) = \alpha$ if $x_u = v_u^\alpha$ for some $\alpha \in \{0, 1, \ldots, q(u)\}$. In other words, $x_u = v_u^{i(u)}$ denotes the installed capacity in edge $u \in U$.

Algorithm 1 presents the pseudo-code of the proposed heuristic HC which builds a feasible solution by solving a sequence of maximum flow problems. The algorithm takes as input an initial seed (*seed*) for the pseudo-random number generator. It starts by the loop in lines 2–4, which randomly assigns an initial capacity $x_u = v_u^i$ to each edge $u \in U$, for some $i = 1, \ldots, q(u)$, with probability $p_u^i = r_u^i / (\sum_{k=1,\ldots,q(u)} r_u^k)$ proportional to the ratio $r_u^i = v_u^i / \gamma_u^i$ between the installed capacity v_u^i and its associated cost γ_u^i. We note that $x_u > v_u^0 = 0$ after this step, for every $u \in U$.

1 **Algorithm** HC($seed$)
2 **forall** $u \in U$ **do**
3   Set $x_u \leftarrow v_u^i$ with i randomly selected from $\{1, \ldots, q(u)\}$ with probability $p_u^i = r_u^i / \sum_{k=1,\ldots,q(u)} r_u^k$;
4 **end**
5 $UR \leftarrow \emptyset$; $\mathcal{K} = \{1, \ldots, K\}$;
6 **while** $\mathcal{K} \neq \emptyset$ **do**
7   Randomly select a demand D_k, with $k \in \mathcal{K}$;
8   Compute the maximum flow $fmax_k$ from s_k to t_k using the residual capacities;
9   **if** $fmax_k \geq f_k$ **then**
10   Route the flow f_k from s_k to t_k with a minimum cost flow algorithm and update the flows ψ;
11   **else**
12         $UR \leftarrow UR \cup \{k\}$;
13   **end**
14     $\mathcal{K} \leftarrow \mathcal{K} - \{k\}$;
15 **end**
16 **while** $UR \neq \emptyset$ **do**
17   Randomly select a demand D_k, with $k \in UR$;
18   Compute the maximum flow $fmax_k$ from s_k to t_k using the residual capacities;
19   **while** $fmax_k < f_k$ **do**
20   Determine the minimum cut separating s_k and t_k;
21   **if** the flow in every edge u of the minimum cut is equal to $v_u^{q(u)}$ **then**
22   stop: the current solution is infeasible;
23   **else**
24   For every edge $u \in U$, let $i(u) \in \{1, \ldots, q(u)\}$ be such that $x_u = v_u^{i(u)}$;
25   Let $\bar{u}$ be the edge maximizing $(v_u^{i(u)+1} - v_u^{i(u)})/(\gamma_u^{i(u)+1} - \gamma_u^{i(u)})$ over all edges u in the minimum cut separating s_k and t_k;
26   Increase the capacity of edge $\bar{u}$ by setting $x_{\bar{u}} \leftarrow v_{\bar{u}}^{i(\bar{u})+1}$;
27   **end**
28   Update the maximum flow $fmax_k$ from s_k to t_k;
29   **end**
30   Route the flow f_k from s_k to t_k with a minimum cost flow algorithm and update the flows ψ;
31     $UR \leftarrow UR - \{k\}$;
32 **end**
33 $x_u \leftarrow \min\{v_u^t, t = 0, \ldots, q(u) : v_u^t \geq \sum_{k=1}^{K} \psi_{ij}^k + \psi_{ji}^k\}, \ \forall u = (i,j) \in U$;
34 **return** (x, ψ)

Algorithm 1: Pseudo-code of the constructive heuristic HC.

The loop in lines 6–15 attempts to route each demand $D_k, k = 1, \ldots, K$ using a maximum flow algorithm [7] with the current installed capacities. If the maximum flow $fmax_k$ from s_k to t_k is larger than flow f_k, then the latter is routed from s_k to t_k by a minimum cost flow algorithm [19]. Demands that cannot be entirely routed are saved in the set UR in line 12 to be routed in a later step.

Edge capacities are augmented in the loop in lines 16–32 until all demands can be routed and a feasible solution is found. At each iteration, a demand D_k is randomly selected in line 17 from the set of unrouted demands and the maximum flow $fmax_k$ from the origin s_k to the destination t_k is computed in line 18. Capacity increases are made in the loop in lines 19–29 until demand D_k can be successfully routed. The minimum cut separating s_k and t_k is determined in line 20. If all edges in the minimum cut are at their upper bounds, then the solution under construction is infeasible. Otherwise, the minimum cut edge providing the largest capacity/cost ratio is selected in lines 24–25 and its capacity is increased by one step in line 26. The maximum flow $fmax_k$ is updated in line 28 and the loop in lines 19–29 is interrupted when the flow f_k can be entirely routed. Then, demand D_k is routed in line 30 by the same aforementioned minimum cost flow algorithm and a new iteration of the outer loop resumes.

Finally, an attempt is made in line 33 to remove unused capacities after all demands have been routed. All edges are checked one at a time to verify if there are unused capacity steps to be eliminated. If the flow traversing an edge $u \in U$ with capacity $x_u = v_u^{i(u)}$ is less than or equal to $v_u^{i(u)-1}$, then its capacity can be reduced to at least $v_u^{i(u)-1}$. We remark that edges with null capacities may now exist in the solution after this step. A feasible solution defined by the installed capacities x and the flows ψ is returned in line 34.

3.2 Local search

Solutions originated from constructive heuristics are not necessarily optimal and usually can be improved by local search procedures. The local search heuristics presented in this section find local optima with respect to large neighborhoods that are explored by network flow algorithms in the same spirit of [1]. Two different neighborhood structures were used: *link-rerouting* and *demand-rerouting*.

Link-rerouting. A solution to DCMFP is defined by the installed capacities x_u, for every edge $u \in U$, and by a feasible routing using these capacities, mapped into variables ψ_{ij}^k and ψ_{ji}^k, for every edge $u = (i, j) \in U$ and every demand $D_k, k = 1, \dots, K$.

The basic principle of the link-rerouting heuristic [12] consists in performing successive improvements in the solution by rerouting part of the flow traversing each edge, so as to make it possible to reduce its installed capacity. The link-rerouting heuristic confines the neighborhood exploration to reroutings using a unique path.

Let (x, ψ) be a feasible solution and $y_u = \sum_{k=1}^{K}(\psi_{ij}^k + \psi_{ji}^k) \in (v_u^{i(u)-1}, v_u^{i(u)}]$ be the exact flow traversing edge $u = (i, j) \in U$, for some $i(u) = 1, \dots, q(u)$ if $x_u > 0$, $y_u = 0$ otherwise. Basically, for every edge $u \in U$, the heuristic progressively investigates the rerouting of $y_u - v_u^{i(u)-1}$, $y_u - v_u^{i(u)-2}$, $\dots$, $y_u - v_u^1$, and, finally, y_u units of flow. The amount of flow that leads to the largest reduction in the cost of the current solution is rerouted and a new solution is obtained. Local search halts when no improving solution can be found.

Algorithm 2 presents the pseudo-code of the link-rerouting heuristic. It starts with a solution (x, ψ). The loop in lines 2–38 attempts to find an improving neighbor by rerouting the flow in the current solution, until no further improvement is possible. The cost $\phi(x)$ of the current solution is saved into ϕ' in line 3. The loop in lines 5–37 investigates flow rerouting for each edge $u = (i, j) \in U$ with $y_u > 0$. The flow $y_u \in (v_u^{i(u)-1}, v_u^{i(u)}]$ through all edges $u \in U$ is computed in line 4. The cost $\phi(x)$ of the current solution is saved into ϕ_{min} in line 6. The reduced graph $G_u = (N, U \setminus \{u\})$ is built in line 7 The index $h = i(u) - 1$ initialized in line 8 is used in the loop in lines 9–18 to control all possible reductions in the flow through edge u. Each iteration of this loop investigates the rerouting of $\sigma_u = y_u - v_u^h$ units of flow (line 10) from edge $u = (i, j)$ using a shortest path between nodes i and j (line 12) in the reduced graph $G_u = (N, U \setminus \{u\})$ with lengths $\Delta_{u'} = \phi_{u'}(y_{u'} + \sigma_u) - \phi_{u'}(y_{u'})$ associated to each edge $u' \in U \setminus \{u\}$ (line 11). If the cost of the new solution obtained by rerouting σ_u units of flow from edge $u = (i, j)$ (line 13) is better than the cost ϕ_{min} of the best neighbor, then ϕ_{min} is updated in line 14 and the corresponding flow reduction is saved in line 15. The index used to investigate all possible flow amounts to be rerouted is decreased in line 17. If the test in line 19 detects that the cost of the best neighbor obtained by rerouting flow currently traversing edge $u = (i, j)$ is smaller than the cost of the current solution, then the latter is updated in lines 20 to 35.

The total amount $\bar{\sigma}$ of flow to be rerouted is stored in line 20, while the amount f of flow rerouted is initialized in line 21. Demands $D_1, \ldots, D_K$ are investigated in lines 22 to 34, until $\bar{\sigma}$ units of flow through edge $u = (i, j)$ have been rerouted. Due to the structure of the cost function and update procedures, at least one of the flows ψ_{ij}^k or ψ_{ji}^k is always null. Lines 25 to 27 handle the case $\psi_{ij}^k > 0$. Let $\sigma = \min\{\psi_{ij}^k, \bar{\sigma} - f\}$ in line 26 be the maximum amount of flow of demand D_k traversing edge $u = (i, j)$ to be rerouted in line 27 using the shortest path from node i to j in G_u. Similarly, lines 28 to 30 handle the case $\psi_{ji}^k > 0$, for which $\sigma = \min\{\psi_{ji}^k, \bar{\sigma} - f\}$ in line 29 is equal to the maximum amount of flow of demand D_k traversing edge $u = (i, j)$ from node j to i to be rerouted in line 30 using the shortest path from node j to i in G_u. The total rerouted flow is updated in line 32 and a new demand is taken into consideration due to operation in line 33. Finally, installed capacities in edges $u \in U$ are updated in line 35.

Algorithm LSLink(x, ψ)
repeat
  $\phi' \leftarrow \phi(x)$;
  $y_u \leftarrow \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k), \ \forall u = (i,j) \in U$;
  forall $u = (i,j) \in U$ **do**
    $\phi_{min} \leftarrow \phi(x)$;
    Construct the reduced graph $G_u = (N, U \setminus \{u\})$;
    $h \leftarrow i(u) - 1$;
    while $h \geq 0$ **do**
      $\sigma_u \leftarrow y_u - v_u^h$;
      Set lengths $\Delta_{u'} = \phi_{u'}(y_{u'} + \sigma_u) - \phi_{u'}(y_{u'})$ associated to the edges of G_u, $\forall u' \in U \setminus \{u\}$;
      Compute the length L of the shortest path from node i to j in G_u;
      if $\phi_{min} > \phi(x) - \phi_u(y_u) + \phi_u(y_u - v_u^h) + L$ **then**
        $\phi_{min} \leftarrow \phi(x) - \phi_u(y_u) + \phi_u(y_u - v_u^h) + L$;
        $h_{min} \leftarrow h$;
      end
      $h \leftarrow h - 1$;
    end
    if $\phi_{min} < \phi(x)$ **then**
      $\bar{\sigma} \leftarrow y_u - v_u^{h_{min}}$;
      $f \leftarrow 0$;
      $k \leftarrow 1$;
      Set lengths $\Delta_{u'} = \phi_{u'}(y_{u'} + \bar{\sigma}) - \phi_{u'}(y_{u'})$ associated to the edges of G_u, $\forall u' \in U \setminus \{u\}$;
      while $f < \bar{\sigma}$ **do**
        if $\psi_{ij}^k > 0$ **then**
          $\sigma \leftarrow \min\{\psi_{ij}^k, \bar{\sigma} - f\}$;
          Route σ additional units of flow through the shortest path from node i to j in G_u with lengths Δ and remove σ units of flow of demand D_k from edge $u = (i,j)$, i.e. set $\psi_{ij}^k \leftarrow \psi_{ij}^k - \sigma$;
        else if $\psi_{ji}^k > 0$ **then**
          $\sigma \leftarrow \min\{\psi_{ji}^k, \bar{\sigma} - f\}$;
          Route σ additional units of flow through the shortest path from node j to i in G_u with lengths Δ and remove σ units of flow of demand D_k from edge $u = (j,i)$, i.e. set $\psi_{ji}^k \leftarrow \psi_{ji}^k - \sigma$;
        end
        $f \leftarrow f + \sigma$;
        $k \leftarrow k + 1$;
      end
      $x_{u'} \leftarrow \min\{v_{u'}^t, t = 0, \ldots, q(u') : v_{u'}^t \geq \sum_{k=1}^{K} \psi_{ij}^k + \psi_{ji}^k\}, \ \forall u' = (i,j) \in U$;
    end
  end
until $\phi' = \phi(x)$;
return (x, ψ)

Algorithm 2: Pseudo-code of the link-rerouting heuristic.

Demand-rerouting. Instead of rerouting aggregated flow traversing a single edge, this local search heuristic evaluates neighbors of the current solution by rerouting each individual demand in the network. The neighborhood is composed by all possible path and cycle flows that reroute the considered demand. As before, let (x, ψ) be the current solution and $y_u = \sum_{k=1}^{K}(\psi_{ij}^k + \psi_{ji}^k)$ the exact flow through edge $u = (i, j) \in U$.

For any given demand D_ℓ, with $\ell = 1, \ldots, K$, let

$$\bar{y}_u = \sum_{\substack{k=1 \\ k \neq \ell}}^{K} (\psi_{ij}^k + \psi_{ji}^k)$$

be the flow through edge $u = (i, j) \in U$ after the removal of demand D_ℓ. The increase in the cost of the current solution due to rerouting an additional amount of flow δ_u through edge $u \in U$ is given by $\tilde{\phi}_u(\delta_u) = \phi_u(\overline{y}_u + \delta_u) - \phi(\overline{y}_u)$. The new flows associated with demand D_ℓ can be obtained by solving the following single commodity discrete minimum cost flow problem $SP(\ell, \overline{y})$, where the right hand side of inequality (11) is the residual capacity of an edge u considering that the other demands $D_k, k \in \{1, \ldots, K\} \setminus \{\ell\}$ have been routed:

$$\min \sum_{u=(i,j) \in U} \tilde{\phi}_u(\delta_u) \tag{8}$$

subject to:

$$\sum_{j \in \Gamma(i)} (\psi_{ij}^\ell - \psi_{ji}^\ell) = \left\{ \begin{array}{ll} +f_\ell, & i = s_\ell \\ 0, & i \neq s_\ell, i \neq t_\ell \\ -f_\ell, & i = t_\ell \end{array} \right\} \qquad \forall i \in N \tag{9}$$

$$\delta_u = \psi_{ij}^\ell + \psi_{ji}^\ell \qquad \forall u = (i, j) \in U \tag{10}$$

$$0 \leq \delta_u \leq \beta_u - \overline{y}_u \qquad \forall u = (i, j) \in U \tag{11}$$

$$\psi_{ij}^\ell, \psi_{ji}^\ell \geq 0 \qquad \forall u = (i, j) \in U. \tag{12}$$

Problem (8) to (12) above is NP-hard [8]. Knippel [23] developed a specialized branch-and-bound algorithm for its exact solution. We propose to solve it approximately by a heuristic based on maximum flow and minimum cost flow algorithms [7, 19] which allows to speedup neighborhood exploration.

1 **Algorithm** LSRoute(x, ψ)
2 **repeat**
3 $\quad \phi' \leftarrow \phi(x)$;
4 $\quad U = \{1, \ldots, K\}$;
5 $\quad$ **forall** $j = 1, \ldots, K$ **do**
6 $\quad\quad$ Randomly select a demand $k \in U$;
7 $\quad\quad U \leftarrow U - \{k\}$;
8 $\quad\quad \bar{x} \leftarrow x; \bar{\psi} \leftarrow \psi$;
9 $\quad\quad \bar{\phi} \leftarrow \phi(x)$;
10 $\quad\quad \psi_{ij}^k, \psi_{ji}^k \leftarrow 0, \forall u = (i, j) \in U$;
11 $\quad\quad x_u \leftarrow \min\{v_u^t, t = 0, \ldots, q(u) : v_u^t \geq y_u - (\psi_{ij}^k + \psi_{ji}^k)\}, \forall u = (i, j) \in U$;
12 $\quad\quad$ Compute the maximum flow $fmax_k$ from s_k to t_k using the residual capacities;
13 $\quad\quad$ **while** $fmax_k < f_k$ *and* $\phi(x) < \bar{\phi}$ **do**
14 $\quad\quad\quad$ Determine the minimum cut separating s_k and t_k;
15 $\quad\quad\quad$ For every edge $u \in U$, let $i(u) \in \{0, \ldots, q(u)\}$ be such that $x_u = v_u^{i(u)}$;
16 $\quad\quad\quad$ Let $\bar{u}$ be the edge maximizing ratio $(v_u^{i(u)+1} - v_u^{i(u)}) \big/ (\gamma_u^{i(u)+1} - \gamma_u^{i(u)})$ over all edges u in the minimum cut separating s_k and t_k;
17 $\quad\quad\quad$ Increase the capacity of edge $\bar{u}$ by setting $x_{\bar{u}} \leftarrow v_{\bar{u}}^{i(\bar{u})+1}$;
18 $\quad\quad\quad$ Update the maximum flow $fmax_k$ from s_k to t_k;
19 $\quad\quad$ **end**
20 $\quad\quad$ **if** $\phi(x) < \bar{\phi}$ **then**
21 $\quad\quad\quad$ Route the flow f_k from s_k to t_k with the minimum cost flow algorithm and update the flows ψ;
22 $\quad\quad$ **else**
23 $\quad\quad\quad x \leftarrow \overline{x}; \psi \leftarrow \overline{\psi}$;
24 $\quad\quad$ **end**
25 $\quad$ **end**
26 **until** $\phi' = \phi(x)$;
27 **return** (x, ψ)

Algorithm 3: Pseudo-code of the demand-rerouting heuristic.

Algorithm 3 presents the pseudo-code of the demand-rerouting heuristic. The heuristic attempts to reroute the flows in lines 2–23, until no further improvement is possible. The cost $\phi(x)$ of the current solution is saved into ϕ' in line 3. The loop in lines 5–25 attempts to reroute at each of its iterations the flow of some demand $D_k, k = 1, \ldots, K$ in a random order. The current solution (x, ψ) is saved into $(\bar{x}, \bar{\psi})$ in line 8, while its cost $\phi(x)$ is saved into $\bar{\phi}$ in line 9. The flows associated with demand D_k are zeroed in line 10. The residual installed capacities after the removal of demand D_k are updated in line 11. The maximum flow $fmax_k$ from s_k to t_k using the residual capacities is computed in line 12. Capacity increases are made in the loop in lines 13–19 until demand D_k can be successfully routed or until the cost of rerouting demand D_k is proved to be greater than that of its previous routing. The minimum cut separating s_k and t_k is determined in line 14. The minimum cut edge providing the largest capacity/cost ratio is selected in lines 15–16 and its capacity is increased by one step in line 17. The maximum flow $fmax_k$ from s_k to t_k is updated in line 18. If the test in line 20 determines that the successive capacity augmentations yielded an improving solution, then demand D_k is routed in line 21 by the minimum cost flow algorithm. Otherwise, the original solution $(\bar{x}, \bar{\psi})$ saved in line 5 is restored as the incumbent in line 23. In both cases, a new iteration of the outer loop resumes. Finally, a solution (x, ψ) is returned in line 27.

4 Memory-based multistart framework

A multistart algorithm was designed to DCMFP using the heuristics described in the previous section. Its main building blocks are the randomized construction and local search heuristics. They are applied until some stopping condition is met. We propose two memory-based mechanisms to improve its solution quality: adaptive memory and vocabulary building. They are both based on an adaptive set P of elite solutions, which is continuously updated during the search.

4.1 Adaptive memory

We describe in this section the first memory mechanism used to improve the performance of the multistart algorithm for DCMFP. It allows the method to assess the quality of the choices made during its construction phase. This memory mechanism was first proposed in [9, 18].

Each solution is represented with a capacity vector x, with one component $x_u = v_u^{i(u)}$ for each edge $u \in U$, for some $i(u) = 0, \ldots, q(u)$. The capacity assignments are guided by a purely greedy criterion which favors the steps with larger capacity/cost ratios, for each edge $u \in U$. We define the normalized capacity/cost ratio as

$$\hat{r}_u^i = \begin{cases} r_u^i / \max\{r_u^k : k = 0, \ldots, q(u)\}, & \text{if } i \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

The relative weight of each elite solution $(x^\ell, \psi^\ell), \ell = 1, \ldots, |P|$, is given by

$$w(\ell) = \frac{\min\{\phi(x^{\ell'}), \ell' = 1, \ldots, |P|\}}{\phi(x^\ell)}.$$

Furthermore, let

$$W_u^i = \sum_{\ell=1,\ldots,|P|: x_u^\ell = v_u^i} w(\ell)$$

be a measure of quality and frequency associated to the assignment $x_u = v_u^i$ in the elite solutions in P. Thus, W_u^i is larger for assignments $x_u = v_u^i$ which arise more often and with larger relative weights in the elite set P.

We define the attractiveness of setting $x_u = v_u^i$ in the memory-based multistart heuristic as

$$E(u, i) = \lambda \cdot \hat{r}_u^i + W_u^i,$$

which depends on both greedy and adaptive memory information. Parameter λ controls the importance of the memory information over the greedy criterion during the execution of the algorithm. In the beginning, this parameter is set to a large value, emphasizing the greedy information. Memory information prematurely gathered in the beginning of the search will not be used. The value of λ decreases along the time to focus the search in promising regions of the solution space.

For each edge $u \in U$, the capacity assignment $x_u^i = v_u^i$ will therefore be selected with probability

$$\bar{p}_u^i = \frac{E(u,i)}{\sum\limits_{k=1,\ldots,q(u)} E(u,k)},$$

for $i = 1, \ldots, q(u)$ in line 3 of the extended version of algorithm HC based on adaptive memory.

4.2 Vocabulary building

Vocabulary building is an intensification strategy proposed in [16, 18] to create new solutions from good fragments (or blocks) of high quality solutions previously found and stored in the elite set P. Solutions in the elite set and their fragments are uniquely represented by their (possibly incomplete) vectors of installed capacities.

We applied vocabulary building to DCMFP by using the same algorithms described in [18]. The first algorithm applies the *intersection* operator $Int(x', x'')$ to pairs of elite solutions x' and x'' to identify good solution fragments. The order in which the elite solutions are considered may lead to the identification of different fragments. In the vocabulary building heuristic proposed here, $|P|$ random solution orderings are used by the intersection operator to identify fragments which are uniquely stored in the set of fragments Z. A parameter ℓ_{min} is used to set the minimum size of fragments extracted by the intersection operator, since very small fragments do not constitute sufficient information to guide the search.

The second algorithm is used to assemble new solutions from good fragments previously identified and stored in Z. An extended intersection operator $EInt(z', z'')$ is used to combine pairs of fragments z' and z'' in Z.

Solutions obtained by fragment recombination may be infeasible (i.e., the installed capacities may not be enough to route all demands) or incomplete (i.e., the capacities of some edges may not be defined). Given any incomplete solution $\overline{x}$, we define a neighborhood formed by all possible alternatives to complete its vector of installed capacities. An auxiliary minimum cost multicommodity flow problem is solved to efficiently explore this neighborhood. The best solution in the proposed neighborhood is given by the solution of the linear program (MCMFP)

$$\min \sum_{u=(i,j)\in U} c_u \cdot \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k) \tag{13}$$

subject to

$$\sum_{j\in\Gamma(i)} (\psi_{ij}^k - \psi_{ji}^k) = \left\{ \begin{array}{ll} +f_k, & i = s_k \\ 0, & i \neq s_k, i \neq t_k \\ -f_k, & i = t_k \end{array} \right\} \qquad \forall i \in N \quad \forall k = 1, \ldots, K \tag{14}$$

$$\sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k) \leq \alpha_u \quad \forall u = (i,j) \in U \tag{15}$$

$$\psi_{ij}^k, \psi_{ji}^k \geq 0 \qquad \forall u = (i,j) \in U, \quad \forall k = 1, \ldots, K, \tag{16}$$

with costs and capacities given by

$$c_u = \begin{cases} 0, & \text{if} \quad \overline{x}_u = v_u^i \quad \text{for some } i = 0, \ldots, q(u) \\ M, & \text{if} \quad \overline{x}_u \text{ is undefined,} \end{cases}$$

$$\alpha_u = \begin{cases} \overline{x}_u, & \text{if} \quad \overline{x}_u = v_u^i \quad \text{for some } i = 0, \ldots, q(u) \\ \beta_u, & \text{if} \quad \overline{x}_u \text{ is undefined,} \end{cases}$$

for each edge $u \in U$.

The capacity constraints of the network flow model consider the values in $\bar{x}$, except for the undefined capacities for which the constraints have upper limits equal to β_u. A non zero cost M is incurred by each unit of flow passing through edges with undefined capacities, while no cost is incurred by flow passing through the other edges. Therefore, demands will preferably be routed through edges with existing installed capacities, using those with undefined capacities only if strictly necessary. Thus, if the above linear program is feasible, the vector of capacities $\bar{x}$ is determined as $\bar{x}_u = \min\{v_u^i, i = 0, \ldots, q(u) : v_u^i \geq \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k)\}, \forall u \in U$. The optimal values of variables ψ complete a feasible solution $(\overline{x}, \psi)$ for the problem.

The exploration of this neighborhood is also performed for complete solutions obtained by vocabulary building, since the LP solver might eliminate unused capacity steps. This happens to a complete solution x whenever $\min\{v_u^i, i = 0, \ldots, q(u) : v_u^i \geq \sum_{k=1}^{K} (\psi_{ij}^k + \psi_{ji}^k)\} < x_u$ for some edge $u \in U$.

4.3 Full memory-based multistart heuristic

Once all building blocks of the memory-based multistart algorithm have been described (constructive heuristic, local search procedures, adaptive memory, and vocabulary building), the latter is presented in detail by Algorithm 4.

```
1  Algorithm MS+MEM+VB
2  $\phi^* \leftarrow \infty$;
3  Let seed be any seed for the pseudo-random number generator;
4  while stopping condition not met do
5      $(x, \psi) \leftarrow$ HC(seed);
6      repeat
7          $\phi' \leftarrow \phi(x)$;
8          $(x, \psi) \leftarrow$ LSLink$(x, \psi)$;
9          $(x, \psi) \leftarrow$ LSRoute$(x, \psi)$;
10     until $\phi' = \phi(x)$;
11     if $\phi(x) < \phi^*$ then
12         $x^* \leftarrow x$; $\psi^* \leftarrow \psi$; $\phi^* \leftarrow \phi(x)$;
13     end
14     Update the set of elite solutions $P$ with $(x, \psi)$;
15     if vocabulary building condition is met then
16         $Z \leftarrow \emptyset$;
17         for $i = 1, \ldots, |P|$ do
18             Build a random ordering of the elite solutions;
19             Use the $Int$ operator to extract a fragment $\overline{z}$ of installed capacities of
               size at least $\ell_{min}$;
20             $Z \leftarrow Z \cup \{\overline{z}\}$;
21         end
22         for $j = 1, \ldots, |Z|$ do
23             Build a random ordering of fragments in $Z$;
24             Use the $EInt$ operator to assemble a solution $\overline{x}$ of installed capacities;
25             Solve MCMFP with costs and capacities defined by $\overline{x}$;
26             if MCMFP is feasible then
27                 $\bar{x}_u \leftarrow \min\{v_u^i, i = 0, \ldots, q(u) : v_u^i \geq \sum_{k=1}^{K}(\psi_{ij}^k + \psi_{ji}^k)\}, \forall u \in U$;
28                 if $\phi(\bar{x}) < \phi^*$ then
29                     $x^* \leftarrow \overline{x}$; $\psi^* \leftarrow \psi$; $\phi^* \leftarrow \phi(\bar{x})$;
30                 end
31                 Update the set of elite solutions $P$ with $(\overline{x}, \psi)$;
32             end
33         end
34     end
35     Generate a new value for seed;
36 end
37 return $(x^*, \psi^*)$
```

Algorithm 4: Pseudo-code of the memory-based multistart heuristic.

Algorithm MS+MEM+VB starts by the initializations of the cost of the best solution found in line 2 and of the seed used for the pseudo-random number generator in line 3. Lines 3–36 correspond to the main loop of the memory-based multistart algorithm. Each of its iterations starts by the construction of an initial solution (x, ψ) in line 5 using the adaptive memory criterion described in Section 4.1, expressed by means of the modified probabilities $\bar{p}_u^i$ for each $u \in U$ and each $i = 1, \ldots, q(u)$. Different initial solutions are obtained in each iteration by providing different seeds to algorithm HC.

The combined local search strategy following the VND model (see e.g. [20]) and exploring the two distinct neighborhoods described in Section 3.2 appears in the loop of lines 6–10. Local search stops when none of the two neighborhoods can lead to a solution improving the current solution ϕ'.

If the test in line 11 determines that solution (x, ψ) obtained by local search improves the best solution found, then the latter and its cost are updated in line 12. The set P of elite solutions is updated in line 14: the new solution x replaces the worst elite solution if it is better than the latter and differs from every solution in the pool by the installed capacities in at least d_{min} edges, where d_{min} is a parameter. We notice that only the installed capacities are stored for each elite solution, since the associated routings are not necessary for the operations performed by the algorithm with the elite set P.

Lines 14–34 consist in the application of the vocabulary building procedure. Each of the $|P|$ iterations of the loop in lines 17–21 applies the intersection operator in line 19 using a different random order of the elite solutions in P, as defined in line 18, such that a fragment of installed capacities of size at least ℓ_{min} is extracted. It is then uniquely stored in the set Z in line 20. Vocabulary building proceeds in the loop in lines 22–33 by applying the extended intersection operator to assemble $|Z|$ solutions from random orderings of the fragments stored in Z, which are created one-at-a-time in line 23. Once a solution $\overline{x}$ is obtained, problem MCMFP with costs and capacities defined by $\overline{x}$ is solved in line 25 to be tested for feasibility in line 26. If it is feasible, then its solution ψ is used to complete and update the current solution of installed capacities in line 27. If the test in line 28 determines that solution $(\bar{x}, \psi)$ improves the best solution found to DCMFP, then the latter and its cost are updated in line 29. The solution is also tested in line 31 to possibly update the elite set P. A new seed for the pseudo-random number generator is set in line 35. Finally, the best solution found is returned in line 37.

5 Computational results

The computational experiments were performed on a Pentium IV with a 2GHz clock and 512 Mbytes of RAM memory running under Linux. The algorithms were implemented in C++ and compiled by `gcc` 3.2. We have used our own implementation of the random number generator described in [36].

The 34 test instances used in the experiments were created by a problem generator proposed in [13] and taken from [12]. Table 1 shows for each instance its number of nodes, its number of edges, and the total number of steps over all edges in its cost function. Since there exists one demand for each possible pair of origin-destination nodes in N (i.e., $K = (|N|^2 - |N|)/2$), the number of nodes of an instance may be used as a measure of its hardness.

5.1 First experiment: constructive heuristic

The first computational experiment refers to the behavior of the routing strategy used by the constructive heuristic, in which each demand is routed by a minimum

Table 1. Characteristics of the test instances.

Instance	$\|N\|$	$\|U\|$	#steps	Instance	$\|N\|$	$\|U\|$	#steps
rn15_1	15	26	165	rn20_10	20	34	221
rn15_21	15	26	168	rn30_1	30	53	343
rn15_3	15	26	155	rn30_2	30	54	331
rn15_4	15	27	173	rn30_4	30	54	355
rn15_6	15	26	159	rn30_5	30	54	343
rn15_7	15	27	175	rn30_7	30	54	359
rn15_8	15	26	170	rn30_8	30	52	345
rn15_9	15	26	167	rn30_9	30	54	353
rn15_10	15	28	172	rn30_10	30	54	342
rn20_22	20	37	236	rn40_1	40	71	468
rn20_3	20	34	216	rn40_2	40	72	462
rn20_4	20	35	227	rn40_3	40	75	483
rn20_51	20	37	233	rn40_5	40	74	468
rn20_61	20	25	203	rn40_6	40	73	475
rn20_7	20	36	230	rn40_9	40	71	460
rn20_8	20	36	230	rn40_10	40	74	479
rn20_9	20	35	227	rn50_2	50	90	577

cost flow algorithm. Two different strategies were considered, based on distinct edge utilization costs:

1. *Unit utilization costs:* with a unit utilization cost associated with each edge, the minimum cost flow routing attempts to use a minimum amount of edge residual capacity and leads to solutions with small values of $\sum_{u \in U} x_u$ after all demands in UR have been routed. Although this is a good indication of solution quality, solutions using the minimum amount of capacity are not necessarily optimal.
2. *Utilization costs inversely proportional to residual capacities:* with this strategy, routing is preferably performed through edges with larger residual capacities, resulting in a more equitable use of the network resources. Each call to the minimum cost flow algorithm in lines 10 and 30 of heuristic HC considers the cost per unity of flow through edge $u \in U$ as equal to $\min\{1/(x_u - y_u), M'\}$, where M' is a sufficiently large value to handle the situation $x_u = y_u$.

For any run of heuristic HC, let Δ_1 (resp. Δ_2) denotes the total amount of unrouted flow associated with the demands in set UR after the loop in lines 6–15 is performed using the first (resp. second) cost strategy in line 10. Capacity resources are better used when less flow is left to be routed in the loop in lines 16–32. The superiority of the second strategy over the first may be graphically addressed by plotting the differences $\Delta = \Delta_1 - \Delta_2$ associated with any instance for a large number of runs of the constructive heuristic, using each one of the cost criteria. The initial edge capacities are always the same for both algorithm versions. The differences $\Delta = \Delta_1 - \Delta_2$ for 100 distinct runs of instances rn20_7 and rn40_2 are plotted in Figure 2.

These results show that $\Delta > 0$ for 67 runs of instance rn20_7 and 78 runs of instance rn40_2. This is an indication that the second cost strategy is likely to lead to better solutions, since it leaves less unrouted flow to be handled in lines 16–32. The statistical test t for paired observations [21] made it possible to conclude that for instances rn20_7 and rn40_2 the second strategy is better than the first one with 86.7% and 99.2% of confidence, respectively. This strategy will therefore be used in the sequel.

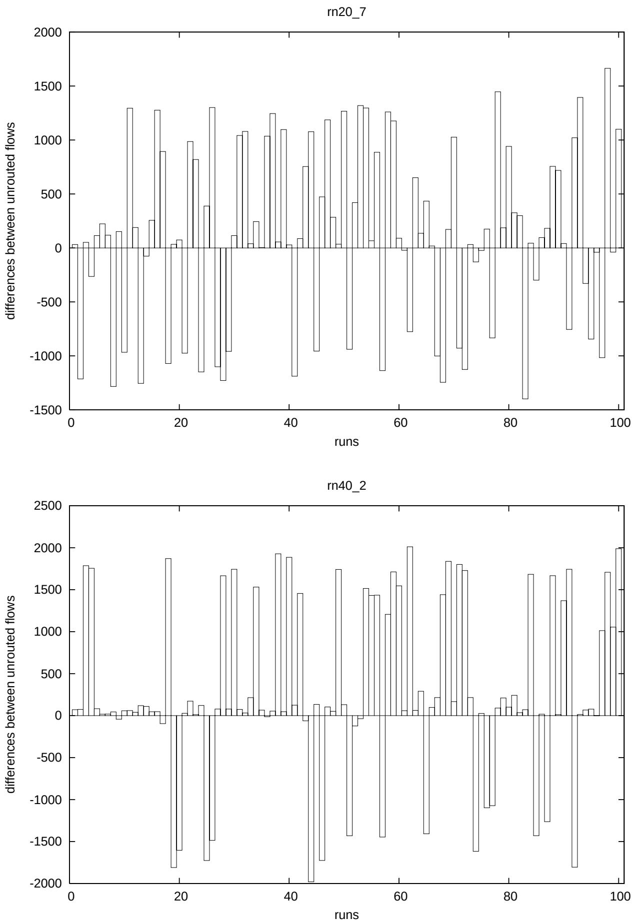

Fig. 2. Differences between the unrouted flows for 100 runs of instances rn20_7 and rn40_2.

5.2 Second experiment: parameter setting

We address the best settings for each of the following parameters used by the adaptive memory component of algorithm MS+MEM+VB: the size of the set P of elite solutions, the minimum dissimilarity d_{min} between any two elite solutions, and the weight λ of the greedy criterion over memory information to construct new solutions.

The elite set P stores valuable information with respect to the search space. Similarities between the elite solutions may be used to identify promising regions of the search space and to intensify the search on such regions. Adaptive memory exploits the information stored in the solutions of P to assign capacity steps in algorithm HC (see Section 4.1), while vocabulary building assemble new solutions from their qualified fragments (see Section 4.2).

The more similarities exist between the solutions in P, the closer the elite solutions are. In order to control the diversity of P and, consequently, the strength of intensification over promising regions of the search space, a minimum accepted dissimilarity d_{min} between any two solutions in P is imposed. Further studies reporting on the constitution of the elite set can be found e.g. in [15, 31, 33].

A small subset formed by instances `rn15_6`, `rn20_22`, `rn30_8`, `rn40_10`, and `rn50_2` is used for setting appropriate values for the size of the elite set and the minimum dissimilarity between any two elite solutions. Table 2 displays the sum of normalized averages over the above five instances for ten executions of algorithm MS+MEM+VB using different values of $|P|$ and d_{min}, without the vocabulary building phase, hereby denoted MS+MEM. For each instance, average normalization is performed as follows. Let $\phi^{a,b}$ be the average cost of the solutions obtained by algorithm MS+MEM with parameters fixed as $|P| = a$ and $d_{min} = b$. Let ϕ^* be the best average cost of the solutions obtained by MS+MEM over all tested parameter combinations. The normalized average values of the solutions found by MS+MEM with parameters $|P| = a$ and $d_{min} = b$ is defined as $\phi^{a,b}/\phi^* - 1$.

Although the results in Table 2 are similar, we choose the configuration with $|P| = 30$ and $d_{min} = 3$. The computation time required for maintaining P is directly proportional to its cardinality, discouraging the use of larger elite sets.

Table 2. Variation of the normalized average values with the parameters $|P|$ and d_{min}.

	d_{min}			
$\|P\|$	1	3	$\|U\|/10$	$\|U\|/5$
10	0.055	0.053	0.031	0.024
20	0.018	0.016	0.014	0.030
30	0.020	**0.013**	0.024	0.054
40	0.040	0.036	0.039	0.043
50	0.044	0.036	0.042	0.063

Parameter λ controls the relative weight of the greedy criterion over the information contained in the adaptive memory. Initially, the value of λ is set to $10 \cdot |P|$. This value is decreased by $|P|$ after every each 50 iterations, until it get equals to $|P|$ and the memory information turns out to have a more significant weight in the construction of new solutions. From this moment on, the value of λ is decreased by $0.05|P|$ after every each 50 iterations, until it gets equal to zero.

Figure 3 displays the cost of the feasible solutions obtained at each iteration of MS+MEM for one particular execution of the instances rn20_7 and rn40_2. They show a progressive improvement in solution quality as more iterations are performed. After 1,500 iterations, λ is made equal to zero and, therefore, the information contained in the adaptive memory stands alone in the construction of new solutions. This behavior clearly shows that the multistart heuristic benefits from the adaptive memory mechanisms.

The stopping criterion of both MS+MEM and MS+MEM+VB versions of the heuristic is set at 3,000 restart iterations in the main loop of lines 4–36 in Algorithm 4. Vocabulary building is applied for the first time after 2,000 iterations and thereafter every 250 iterations. Parameter ℓ_{min} is made smaller for large instances, since the set of elite solutions is more likely to be diversified in these cases: $\ell_{min} = |U| - 8$ for instances with 20 nodes, $\ell_{min} = |U| - 12$ for instances with 30 to 40 nodes, and $\ell_{min} = |U| - 15$ for those with 50 nodes.

CPLEX 10.0 is used to exactly solve the neighborhood search for MCMFP subproblems. The solver is allowed to run for 45 seconds for the instances with 40 nodes and for 60 seconds for those with 50 nodes.

Two experiments were performed with versions MS (pure multistart approach without any memory information), MS+MEM, and MS+MEM+VB of the heuristic on instances with 15, 20, 30, 40, and 50 nodes, in order to assess the gains obtained by the memory mechanisms. In the case of experiment A, each algorithm is allowed to run for 3,000 iterations. In experiment B, the stopping criterion for heuristics MS+MEM and MS+MEM+VB is a time limit which is set as the same computation time spent by version MS of the heuristic to perform 3,000 iterations on the same instance. Table 3 presents average deviation results in percent over ten executions of versions MS, MS+MEM, and MS+MEM+VB of the heuristic, always computed with respect to the best average results obtained in each experiment.

The values in Table 3 show that the full heuristic MS+MEM+VB obtained the best average results in both experiments. The gains obtained by the memory mechanisms with respect to the pure multistart approach clearly improve with the increase in problem size, showing that the approach scales well.

Improvements in solution quality implied by the use of adaptive memory are illustrated by the time-to-target plots associated with the MS and MS+MEM heuristics, following the methodology reported in [2, 3]. Two hundred runs of each heuristic were performed for instances rn20_7 and rn40_2. For each run $i = 1, \ldots, 200$, we collected the computation times t_i (in seconds) spent by each heuristic (MS and MS+MEM) to obtain a solution with cost smaller than or

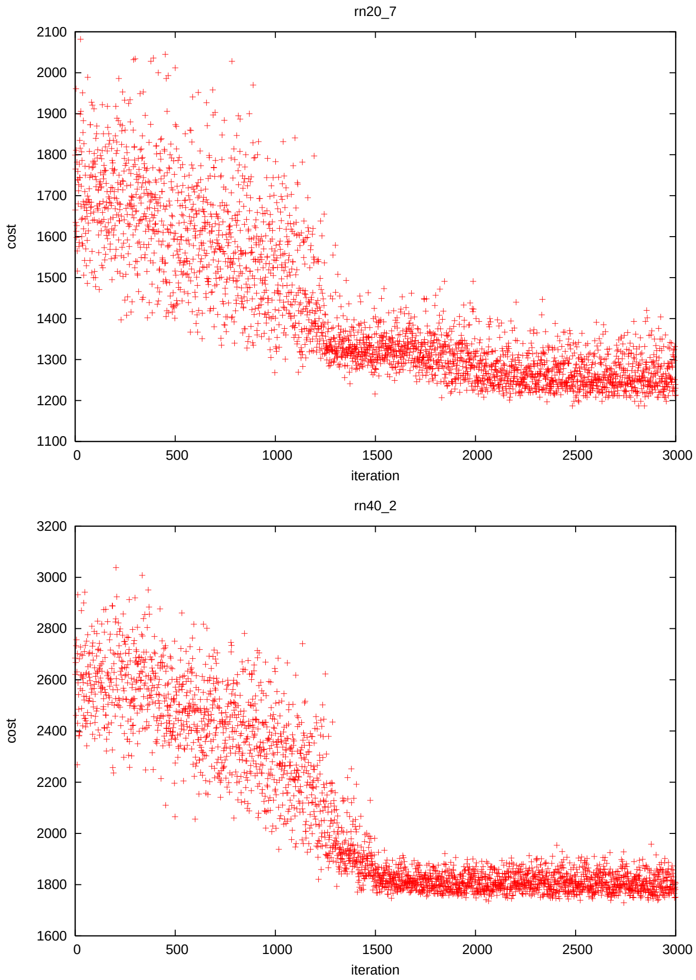

Fig. 3. Solution values along 3,000 iterations of heuristic MS+MEM.

equal to a given target solution value. For each instance, the target solution value used in the experiments corresponds to the worst solution values obtained by running 200 executions of heuristic MS and by running 200 executions of heuristic MS+MEM, each of them for 3,000 iterations. A probability $\Omega_i = (i - \frac{1}{2})/200$ is associated with the i-th smallest running time t_i, and points $z_i = (\Omega_i, t_i)$ for $i = 1, \ldots, 200$ are plotted. The more to the left is a plot, the better is the algorithm corresponding to it. We observe that MS+MEM outperforms MS for both instances plotted in Figure 4.

In order to access the gains obtained by the use of vocabulary building, Figure 5 shows the evolution of the current minimum solution value found along

Table 3. Average deviations in percent from the best average results for experiments A and B, over ten runs of heuristics MS, MS+MEM, and MS+MEM+VB.

	Experiment A (3,000 iterations)			Experiment B (fixed time)		
$\|N\|$	MS	MS+MEM	MS+MEM+VB	MS	MS+MEM	MS+MEM+VB
15	6.5	0.5	0.0	6.5	0.4	0.0
20	8.2	0.7	0.0	8.1	0.6	0.0
30	15.3	1.4	0.0	14.7	0.9	0.0
40	19.2	2.8	0.0	18.0	1.9	0.0
50	21.0	2.9	0.0	20.4	2.6	0.0

one particular execution of heuristics MS+MEM and MS+MEM+VB for instances `rn20_7` and `rn40_2`. Each algorithm was allowed to run for 30 minutes. Solution values are sampled every 10 seconds and whenever the best solution value has improved.

The best solution values obtained by both algorithms are the same until the first application of vocabulary building. We notice that heuristic MS+MEM+VB performed better for both instances in the long term execution. We also observe that the vocabulary building procedure might spend a considerable amount of time for large instances. As an example, the first vocabulary building application to instance `rn40_2` took approximately 330 seconds, leaving to a significant improvement in the best solution value by 1.5%.

5.3 Third experiment: comparison with previous work

Solution values and computation times observed with heuristics `H1` and `H2` previously proposed in the literature were extracted from experiments performed on a Sun UltraSparc 10 system and reported in [12]. The authors considered that `H2` was too time consuming for the larger instances with 50 nodes, therefore only results obtained with `H1` were reported for these instances.

Table 4 presents average and best results for ten runs of heuristics MS+MEM and MS+MEM+VB with the parameters defined in the previous section, along with the results reported in [12] for `H1` and `H2`. For each instance, the value ϕ^* is the best previously known solution in the literature: it was obtained by the exact Bender's decomposition approach for instances with 15 and 20 nodes, while for larger instances it was provided either by the heuristic implementation derived from the exact Benders' decomposition approach or by the MS+MEM+VB heuristic proposed here. All results for which the best solution values obtained by MS+MEM+VB were better than or equal to the best among those of `H1` and `H2` are also outlined in bold in the table.

Heuristic MS+MEM+VB presented better average results than `H1` for 26 out of 34 instances, and better than `H2` for 14 out of 33 instances. Even heuristic MS+MEM, without the vocabulary building component, obtained better average results than `H1` for 22 out of 34 instances, showing that the memory-based multistart algorithm is already an effective approach.

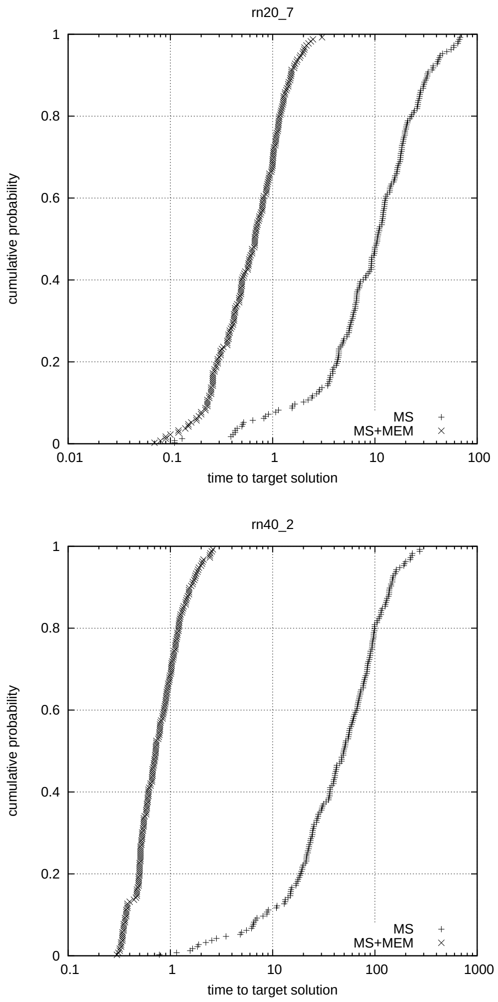

Fig. 4. Time-to-target (in seconds) plots of heuristics MS and MS+MEM for instances rn20_7 and rn40_2.

Furthermore, heuristic MS+MEM+VB improved the best known upper bounds at the time of writing for nine instances, also in bold in Table 4: rn30_2, rn30_4, rn30_5, rn30_7, rn30_8, rn30_9, rn30_10, rn40_5, and rn50_2.

Average computation times observed with heuristics H1, H2, MS+MEM, and MS+MEM+VB were calculated over all instances of the same size and plotted in Figure 6. Despite the use of different computational architectures in the experiments, we notice that the average times observed for heuristics H1 and H2 are smaller for the tested instances. However, the average times with heuristics

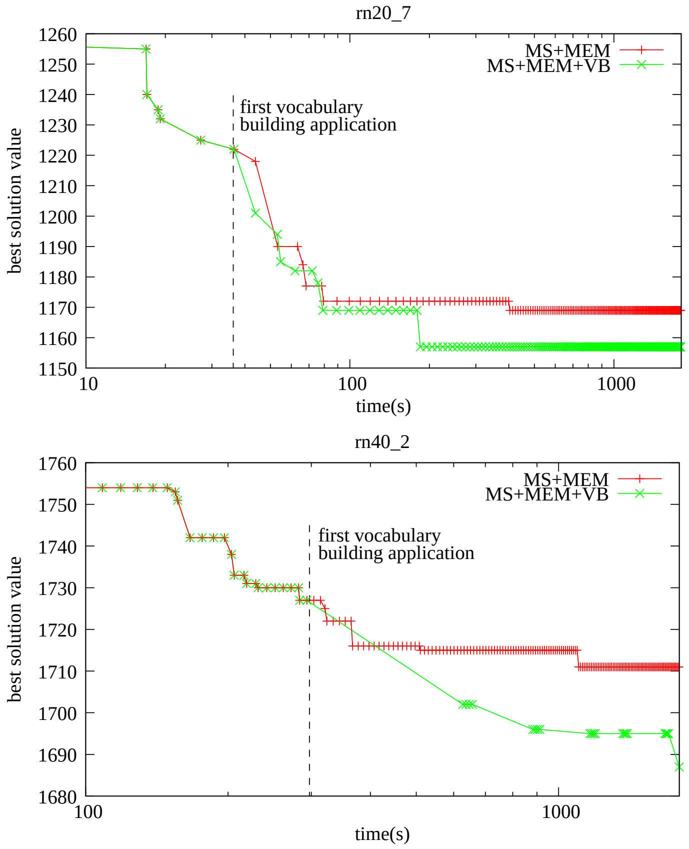

Fig. 5. Plots of minimum solution values as a function of CPU times for instances rn20_7 and rn40_2.

MS+MEM and MS+MEM+VB increase more moderately as the instance sizes augment. Furthermore, since heuristics MS+MEM and MS+MEM+VB work on the primal DCMFP problem, they are more likely to provide good feasible solutions in a reasonable amount of time when tackling very large instances of the problem. In fact, the heuristics based on Benders' approach spend a lot of computation time searching through infeasible solutions.

6 Concluding remarks

We consider the use of adaptive memory and vocabulary building to enhance solution quality in the context of multistart heuristics. Although the above principles are not new, they have not been extensively explored in the recent past. They are blended into an innovative strategy, which is customized for a specific

Table 4. Results obtained with heuristics `H1`, `H2`, MS+MEM, and MS+MEM+VB.

Instance	ϕ^*	`H1`	`H2`	MS+MEM		MS+MEM+VB	
				average	*best*	*average*	*best*
rn15_1	401	481	433	434.6	428	429.7	**409**
rn15_21	794	873	822	814.6	798	814.2	**798**
rn15_3	558	601	558	572.0	569	559.8	**558**
rn15_4	583	587	583	583	583	583.0	**583**
rn15_6	646	651	658	658.2	657	655.0	655
rn15_7	678	765	711	703.2	695	688.4	**685**
rn15_8	905	947	905	939.4	930	935.0	913
rn15_9	699	747	711	729.4	729	728.0	728
rn15_10	678	678	713	678.0	678	678.0	**678**
rn20_22	799	883	824	826.8	817	817.2	**805**
rn20_3	1035	1104	1049	1044.9	1042	1039.0	**1035**
rn20_4	875	954	877	946.8	939	904.5	903
rn20_51	846	927	853	878.3	871	855.8	**848**
rn20_61	785	841	810	785.5	785	785.0	**785**
rn20_7	1157	1236	1229	1199.0	1171	1190.8	**1164**
rn20_8	1182	1299	1253	1226.2	1200	1224.6	**1202**
rn20_9	884	972	928	917.6	902	916.9	**896**
rn20_10	977	1011	1003	1019.9	1009	1001.9	**997**
rn30_1	1324	1462	1324	1388.8	1363	1363.9	1358
rn30_2	**1081**	1157	1084	1109.6	1096	1104.1	**1081**
rn30_4	**1537**	1678	1592	1566.1	1561	1545.2	**1537**
rn30_5	**1050**	1110	1077	1124.7	1113	1087.7	**1050**
rn30_7	**1506**	1580	1511	1559.6	1548	1540.6	**1506**
rn30_8	**1341**	1566	1353	1365.3	1353	1353.8	**1341**
rn30_9	**1310**	1447	1415	1344.1	1325	1328.0	**1310**
rn30_10	**1556**	1643	1568	1679.9	1651	1592.3	**1556**
rn40_1	2214	2279	2214	2377.3	2366	2278.0	2252
rn40_2	1650	1650	1659	1732.7	1696	1702.3	1660
rn40_3	2026	2099	2026	2240.0	2223	2197.1	2176
rn40_5	**1792**	1927	1803	1871.6	1847	1819.1	**1792**
rn40_6	1914	1927	1914	2212.1	2068	1982.5	1946
rn40_9	1471	1541	1471	1661.5	1651	1596.8	1584
rn40_10	2076	2076	2076	2275.8	2246	2161.4	2142
rn50_2	**1961**	2006	-	2119.7	2107	1985.9	**1961**

network design problem with discrete stepwise increasing cost functions of the edge capacities.

The computational experiments have shown that the full approach, combining both adaptive memory and vocabulary building with a multistart heuristic, performs better than the individual components. Each of the two principles brings an additional contribution to solution quality. Together, they speedup the time needed to find approximate solutions meeting a given target quality. The experiments also showed progressive improvements in solution quality as more

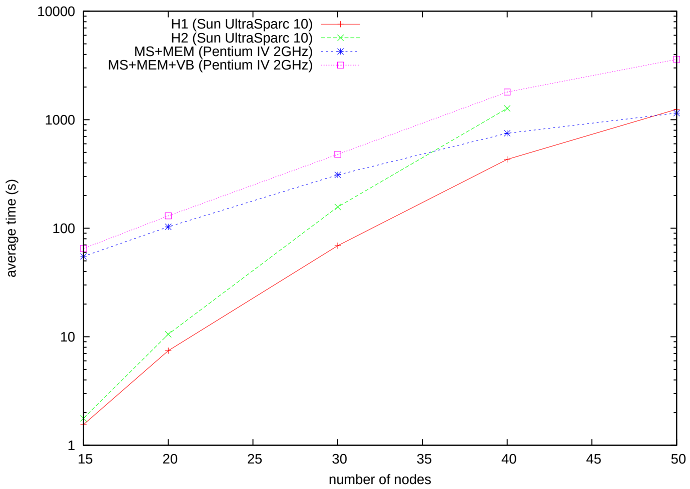

Fig. 6. Average computation times of heuristics `H1`, `H2`, MS+MEM, and MS+MEM+VB for each instance size.

iterations are performed, illustrating the fact that the adaptive memory contributes to drive the search towards better solutions and benefits the multistart heuristic.

The proposed strategy improved the best known solution in the literature for several among the benchmarking instances, and this by as much as 7.4%. Findings and results in this work reinforce other studies, confirming that the intelligent use of memory mechanisms provide valuable strategies for creating improved multistart heuristics.

Acknowledgments

We are thankful to Arnaud Knippel for providing us the test instances. Work of the first author was sponsored by CAPES, CNPq, and FAPERJ. The second author was supported by CNPq grants 301.694/2007-9 and 485.328/2007-0, as well as by FAPERJ grant E-26/102.805/2008.

References

1. R. Ahuja, O. Ergun, J. Orlin, and A. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.
2. R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.
3. R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.

4. L.T.H. An and P.D. Tao. D.C. programming approach for multicommodity network optimization problems with step increasing cost functions. *Journal of Global Optimization*, 22:205–232, 2002.
5. C.F. Bazlamaçci and F. Say. Minimum concave cost multicommodity network design. *Telecommunication Systems*, 36:181–203, 2007.
6. D. Berger, B. Gendron, J.-Y. Potvin, S. Raghavan, and P. Soriano. Tabu search for a network loading problem with multiple facilities. *Journal of Heuristics*, 6:253–267, 2000.
7. B. Cherkassky and A. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
8. S. Chopra, I. Gilboa, and S. Sastry. Algorithms and extended formulations for one and two facility network design. *Lecture Notes in Computer Science*, 1084:44–57, 1996.
9. C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptative memory. *INFORMS Journal on Computing*, 11:198–204, 1999.
10. L. Fratta, M. Gerla, and L. Kleinrock. The flow-deviation method: an approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.
11. V. Gabrel, A. Knippel, and M. Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25:15–23, 1999.
12. V. Gabrel, A. Knippel, and M. Minoux. A comparison of heuristics for the discrete cost multicommodity network optimization problem. *Journal of Heuristics*, 9:429–445, 2003.
13. V. Gabrel and M. Minoux. LP relaxations better than convexification for multicommodity networks optimization problems with step increasing cost functions. *Acta Mathematica Vietnamica*, 22:123–145, 1997.
14. G. Geoffrion and G. Graves. Multicommodity distribution system design by Benders decomposition. *Management Science*, 5:822–844, 1974.
15. I. Ghamlouche, T. Crainic, and M. Gendreau. Path-relinking, cycle-based neighborhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.
16. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problem. *Discrete Applied Mathematics*, 49:231–255, 1992.
17. F. Glover. Tabu search and adaptive memory programming - advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.
18. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.
19. A. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29, 1997.
20. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
21. R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling.* Wiley, 1991.
22. J. Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26:209–236, 1978.
23. A. Knippel. *Modèles et algorithmes de multiflots à cout discontinu pour l'optimisation de réseau de télécommunications.* PhD thesis, Université Paris VI, 2001.

24. M. Laguna, R. Martí, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26:1217–1230, 1999.
25. T. Magnanti and R. Wong. Network design and transportation planning-models and algorithms. *Transportation Science*, 18:1–55, 1984.
26. P. Mahey, A. Ouorou, L. LeBlanc, and J. Chifflet. A new proximal decomposition algorithm for routing in telecommunication networks. *Networks*, 31:227–238, 1998.
27. B. Melián, M. Laguna, and J.A. Moreno-Pérez. Capacity expansion of fiber optic networks with WDM systems: Problem formulation and comparative analysis. *Computers and Operations Research*, 31:461–472, 2004.
28. M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19:313–360, 1989.
29. M. Minoux. Discrete cost multicommodity network optimization problems and exact solution methods. *Annals of Operations Research*, 106:19–46, 2001.
30. M. Mrad and M. Haouari. Optimal solution of the discrete cost multicommodity network design problem. *Applied Mathematics and Computation*, 204:745–753, 2008.
31. M. G. C. Resende and C. C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Kluwer, 2005.
32. C. C. Ribeiro. Approximate algorithms for the automatic planning of power transmission systems (in Portuguese). Master's thesis, COPPE, Universidade Federal do Rio de Janeiro, 1978.
33. C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.
34. Y. Rochat and É. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.
35. E. Rolland, R.A. Patterson, and H. Pirkul. Memory adaptive reasoning and greedy assignment techniques for the CSMT. In S. Voss, S. Martello, I Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 487–498. Kluwer, 1999.
36. L. Schrage. A more portable FORTRAN random number generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.
37. M. Stoer and G. Dahl. A polyhedral approach to multicommodity survivable network design. *Numerische Mathematik*, 68:149–167, 1994.
38. M. Stoer and G. Dahl. A cutting-plane algorithm for multicommodity survivable network design problem. *INFORMS Journal on Computing*, 10:1–11, 1998.