Submitted to *INFORMS Journal on Computing*
manuscript JOC-2013-10-OA-191

Dynamic Programming Driven Memetic Search for the Steiner Tree Problem with Revenues, Budget and Hop Constraints

Zhang-Hua Fu, Jin-Kao Hao* (Corresponding author)
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
{fu,hao}@info.univ-angers.fr

We present a highly effective dynamic programming driven memetic algorithm for the Steiner tree problem with revenues, budget and hop constraints (STPRBH), which aims at determining a subtree of an undirected graph, so as to maximize the collected revenue, subject to both budget and hop constraints. The main features of the proposed algorithm include a probabilistic constructive procedure to generate initial solutions, a neighborhood search procedure using dynamic programming to significantly speed up neighborhood exploration, a backbone-based crossover operator to generate offspring solutions, as well as a quality-and-distance updating strategy to manage the population. Computational results based on four groups of 384 well-known benchmarks demonstrate the value of the proposed algorithm, compared to the state of the art approaches. In particular, for the 56 most challenging instances with unknown optima, our algorithm succeeds in providing 45 improved best known solutions within a short computing time. We additionally provide results for a group of 30 challenging instances that are introduced in the paper. We provide a complexity analysis of the proposed algorithm and study the impact of some ingredients on the performance of the algorithm.

Key words: Constrained Steiner trees; Evolutionary computation; Heuristics; Dynamic programming; Content distribution networks.

1. Introduction

Network design is an extremely challenging task in numerous resource distribution systems (e.g., electricity, telecommunication, heating, transportation, computer networks, etc.) (4, 8, 10, 32). These systems can conveniently be modeled as some variants of the Steiner (or spanning) tree problem (STP) (6, 27, 28). Given a graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ composed of two subsets V^t (called terminal vertices) and V^s (called Steiner vertices) and edge set $E = \{\{i, j\} \mid i, j \in V, i \neq j\}$ where each edge $\{i, j\} \in E$ is associated

with a cost $c_{ij} \geq 0$, the classic Steiner tree problem consists of determining a subtree spanning all the vertices of V^t and possibly some vertices of V^s, so as to minimize the total cost of the obtained tree. As one of Karp's 21 famous NP-complete problems (19), the Steiner tree problem is theoretically important and computationally challenging (16). Notice that the conventional spanning tree problem is a special case of the Steiner tree problem with $V^s = \emptyset$ which is polynomially solvable.

By considering different objectives or constraints, various variants of the Steiner or spanning tree problems can be defined. On the one hand, representative Steiner tree variants include the metric Steiner tree problem (MSTP) (7), the Steiner tree problem with profits (STPP) (11), the Steiner tree problem with hop-constraints (STPH) (29, 31). On the other hand, examples of interesting spanning tree variants are the degree-constrained spanning tree problem (DCSTP) (5), the quadratic minimum spanning tree problem (QMSTP) (9, 24), and the generalized minimum spanning tree problem (GMSTP) (14, 17).

In recent years, the so-called Steiner tree problem with revenues, budget and hop constraints (STPRBH) has received much attention (10, 12, 13, 15, 21, 30). Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E = \{\{i, j\} \mid i, j \in V, i \neq j\}$. Each vertex $i \in V$ has an associated revenue $r_i \geq 0$ (a vertex with $r_i > 0$ is called a *profitable vertex*, otherwise, it is a *non-profitable vertex*), and each edge $\{i, j\} \in E$ has an associated cost $c_{ij} \geq 0$. Let $B \in N$ be the allowed budget and $H \in N$ be the hop limit. Given a root vertex denoted by *root*, the STPRBH is to determine a rooted subtree T (with vertex set $v(T)$ and edge set $e(T)$) that maximizes the collected revenue, while guaranteeing that the total cost of the tree does not exceed B, i.e., $\sum_{\{i,j\} \in e(T)} c_{ij} \leq B$, and the number of edges between *root* and any vertex $i \in v(T)$ never surpasses H. According to this definition, only the root vertex is a terminal vertex ($V^t = \{root\}$) that must be included in any feasible solution, while the other vertices are Steiner vertices ($V^s = V \setminus \{root\}$).

As a generalization of both the STPP and the STPH, the STPRBH can be used to model a number of relevant real-life systems where the objective is to maximize the collected revenue within a given budget, while guaranteeing the reliability of the system. Since 2006, a number of exact or heuristic approaches have been proposed for the STPRBH. Exact algorithms like exemplified in (13, 21, 30) are applicable to solve small or mid-sized problem instances (with up to 500 vertices and 625 edges). For large instances, heuristics or metaheuristics are naturally preferred. For instance, the greedy, destroy-and-repair and tabu

search (TS) algorithms described in (10, 12) have found high quality suboptimal solutions for many STPRBH benchmark instances while the very recent breakout local search (BLS) algorithm (15) has improved a number of previous best known solutions.

In this study, we introduce a highly effective dynamic programming driven memetic algorithm for the STPRBH. The proposed algorithm is based on the general memetic search framework for discrete optimization (18, 23) and integrates a number of distinguished features. The main contributions of the work can be summarized as follows.

From the perspective of the algorithm, the proposed approach incorporates several specifically designed and original ingredients. First, we introduce two move operators for generating neighboring solutions, which are related to, but different from the existing operators. Second, we develop an original dynamic programming driven estimation criterion based on a related 0-1 knapsack problem, which is able to identify and discard a large number of useless neighboring solutions, so as to significantly speed up the local optimization procedure. Third, we devise a backbone-based tree crossover operator for generating offspring solutions. Fourth, we define a quality-and-distance based strategy for a healthy management of the solutions pool. Finally, we provide a complexity analysis of the proposed algorithm and study the impact of some ingredients on the performance of the algorithm. To our knowledge, this is the first evolutionary algorithm proposed for the STPRBH, whose main ideas could be helpful to design effective heuristics for other STP variants.

From the perspective of the computational results, we show 45 improved solutions (new lower bounds, obtained within a short time ranging from several seconds to several minutes) out of 56 most challenging STPRBH benchmarks with unknown optima, and report very competitive results on the remaining 328 benchmarks with known optima compared to the best reference STPRBH algorithms. Additionally, we define a new group of 30 difficult instances and use them to further evaluate the performance of our memetic algorithm. Both the improved solutions reported in the paper and the new problem instances can be used for the purpose of evaluating new STPRBH methods.

The remainder of this paper is organized as follows: After providing some preliminary definitions in Section 2, Section 3 presents the details of the proposed memetic approach. Section 4 is dedicated to computational results and comparisons. Section 5 investigates two key elements of the proposed algorithm, before concluding the paper in Section 6. The online supplement provides a worst case complexity analysis and additional results.

2. Preliminary Definitions

We provide at first some preliminary definitions that are useful for the description of the proposed memetic algorithm.

Definition 1. A *budget and hop constrained Steiner tree (BHS-tree)* is a rooted subtree of graph G meeting both the budget and hop constraints. A BHS-tree is also called a *feasible solution* of the problem.

Comment: Following the compact solution representation method used in (15), we uniquely represent each BHS-tree by a one-dimensional vector $T = \{t_i, i \in V\}$, where t_i denotes the parent vertex of vertex i if $i \in v(T)$ (excluding the root vertex, $v(T)$ denotes the vertex set of T), or $t_i = Null$ otherwise.

Definition 2. Given a BHS-tree T, a *feasible candidate path* with respect to T is a path originating at a vertex $i \in v(T)$ and connecting to an uncollected profitable vertex j ($j \notin v(T), r_j > 0$), such that even after inserting this path into T, the obtained solution is still a BHS-tree, i.e., satisfying both the budget and hop constraints.

Definition 3. A *saturated BHS-tree* is a BHS-tree for which no feasible candidate path exists. Otherwise, the BHS-tree is an *unsaturated (or partial) BHS-tree*.

Definition 4. The *constrained search space* Ω is composed of all BHS-trees (including saturated and unsaturated ones). The *saturated constrained search space* $\overline{\Omega} \subset \Omega$ is composed of all saturated BHS-trees.

Comment: As detailed below, our memetic algorithm restricts its search within the saturated constrained search space $\overline{\Omega}$, thus focusing on the most promising candidate solutions.

Definition 5. Let $L(i, j, l)$ represent the cost of the hop-constrained shortest path between vertex i and vertex j, containing at most l edges, then vertex i is said to be a *reachable vertex* if $L(root, i, H) < \infty$, and *an unreachable vertex* otherwise. Following (15), all the needed values of $L(i, j, l), i, j \in V, r_j > 0, 1 \leq l \leq H$ are pre-calculated by a pre-processing procedure using dynamic programming (20) and stored for a fast access of these values. The unreachable vertices will never be considered during the search process.

Comment: If all the reachable profitable vertices are connected by a BHS-tree T, i.e., $\sum_{i \in v(T)} r_i = \sum_{L(root,i,H)<\infty} r_i$, T corresponds to an optimal solution. This criterion is used as one of the termination criteria of our algorithm (see Section 3).

3. Memetic Algorithm for the STPRBH

Population-based evolutionary algorithms have been successfully applied to many graph and network design problems, including several STPs (3, 6, 8, 26). In this paper, we present a novel dynamic programming driven memetic algorithm (as outlined in Algorithm 1) for solving the STPRBH. To the best of our knowledge, this is the first population-based evolutionary algorithm designed to solve the STPRBH.

Algorithm 1 Dynamic programming driven memetic search algorithm for the STPRBH

REQUIRE: Graph $G(V, E)$, limited budget B, limited hops H, population size Q, maximum number M of non-improving consecutive generations
RETURN: The best solution T^* found meeting both the budget and hop constraints
/* Initialize the population with a sample of Q diversified individuals (saturated BHS-trees $\in \overline{\Omega}$), Section 3.1*/
$Pop = \{T^1, ..., T^Q\} \leftarrow Init\_Population()$
for Each solution T^i belongs to Pop **do**
    /* Apply a dynamic programming based neighborhood search procedure to improve T^i to a local optimum, Section 3.2 */
    $T^i \leftarrow Local\_Search(T^i)$
end for
/* T^* denotes the best solution (collecting the highest revenue) found so far */
$T^* \leftarrow Get\_Best(Pop)$
$\omega \leftarrow 0$
/* Iterate the search until all the reachable profitable vertices are connected, or the best found solution cannot be further improved after M consecutive generations, or the allowed maximum time has been elapsed */
while $\sum_{i \in v(T^*)} r_i < \sum_{L(root,i,H)<\infty} r_i$ and $\omega < M$ and the allowed time is not elapsed **do**
    Randomly select two solutions T^i and T^j from Pop
    /* Apply a backbone based crossover operator to generate an offspring T^0, Section 3.3 */
    $T^0 \leftarrow Crossover(T^i, T^j)$
    /* Improve T^0 to a local optimum, Section 3.2 */
    $T^0 \leftarrow Local\_Search(T^0)$
    /* If T^0 dominates T^*, update T^* and reset ω to 0 */
    if $\sum_{i \in v(T^0)} r_i > \sum_{i \in v(T^*)} r_i$ **then**
        $T^* \leftarrow T^0$
        $\omega \leftarrow 0$
    else
        /* Otherwise, increase ω by 1 */
        $\omega \leftarrow \omega + 1$
    end if
    /*Update Pop with T^0 by a quality-and-distance criterion, Section 3.4 */
    $Pop \leftarrow Update\_Pool(Pop, T^0)$
end while

Our memetic algorithm operates within the saturated constrained search space $\overline{\Omega}$. It starts with a population $Pop \subset \overline{\Omega}$ composed of Q saturated BHS-trees. Each initial solution of Pop is generated by the probabilistic greedy constructive procedure introduced in (15) and further improved by the local optimization procedure which is explained in Section 3.2. Then the algorithm enters into the main optimization process which is mainly driven by the solution recombination operator (backbone based tree crossover, Section 3.3) and local improvement (dynamic programming driven neighborhood search procedure, Section 3.2). At each generation of the memetic algorithm, two parents (randomly chosen) are mated to generate a new offspring solution which is further improved by the local optimization procedure. Finally, a quality-and-distance based criterion is applied to possibly update the population with the improved offspring solution (Section 3.4).

The evolution process terminates if (1) all the reachable profitable vertices are connected by the incumbent solution tree, meaning that an optimal solution is obtained, or (2) the best found solution T^* cannot be further improved for a certain number M of consecutive generations, or (3) the maximum allowed time is elapsed. The first condition depends only on the problem instance to solve, while the other two conditions can influence the quality of the solution found and the running time of the algorithm. Intuitively, a larger M or a longer allowed time imply more generations (thus more computational efforts), but may help discover better solutions. One can possibly set these two conditions to reach the desired trade-off between the solution quality and the computational time.

3.1. Construction of Saturated BHS-trees for Population Initialization

To initialize the population Pop with a given number Q of feasible solutions of reasonable quality, we employ the *probabilistic constructive procedure* introduced in (15) to generate Q saturated BHS-trees, and then improve each of these Q solutions to a local optimum by a dynamic programming driven neighborhood search procedure (detailed in the next subsection). These Q local optima form the first generation of Pop.

3.2. Dynamic Programming Based Fast Neighborhood Search

For local optimization, we devise a neighborhood search (NS) procedure (Algorithm 2) based on two specifically designed move operators (Move_1(T, i) and Move_2(T, i, j)) to explore neighboring saturated BHS-trees. To accelerate the neighborhood exploration by the NS procedure, we develop a dynamic programming based estimation criterion to identify and discard a large number of hopeless neighboring solutions.

Algorithm 2 Dynamic programming based neighborhood search for local optimization

REQUIRE: An initial saturated BHS-tree T

RETURN: A local optimal saturated BHS-tree

Step 1: Create a closely related 0-1 knapsack problem

Let V_{urp} denote the set containing all the reachable profitable vertices uncollected by T

for Each each vertex $j \in V_{urp}$ **do**

Create a knapsack item with item index k, gain v_k and weight w_k

$v_k \leftarrow r_j$ /* The item gain v_k is the revenue of vertex j */

$w_k \leftarrow \min(L(i,j,H-h_i), \forall i \in v(T)$ /* w_k is the cost of the hop-constrained shortest path from vertex j to the BHS-tree T, where h_i being the number of edges between a vertex $i \in v(T)$ and the root vertex */

end for

Set the knapsack capacity of the 0-1 knapsack problem to W_{max}, which is the maximum available budget after deleting any two paths connecting some leaf vertices

Step 2: Solve the 0-1 knapsack problem by dynamic programming

Let $Opt[k,W]$ be the optimal value of the knapsack problem with capacity W, using items up to index k

Calculate, by the dynamic programming procedure of Eq. (2), all the values of $Opt[k,W], 1 \leq k \leq |V_{urp}|, 1 \leq W \leq W_{max}$, and store these values in a $|V_{urp}| \times W_{max}$ table.

Step 3: Local optimization within neighborhood $N(T,1)$

for Each leaf vertex $i \in v(T)$ (examined in random order) **do**

Let W_i be the available budget after deleting the path connecting vertex i

if $r_i < Opt[|V_{urp}|, W_i]$ **then**

$T^\sharp \leftarrow T \bigoplus$ Move_1(T,i)

if $\sum_{k \in v(T^\sharp)} r_k > \sum_{k \in v(T)} r_k$ **then**

$T \leftarrow T^\sharp$ and goto Step 1

end if

end if

end for

Step 4: Local optimization within neighborhood $N(T,2)$

for Each pair of leaf vertices $i,j \in v(T), i<j$ (examined in random order) **do**

Let W_{ij} denote the available budget after deleting the paths connecting i and j

if $r_i + r_j < Opt[|V_{urp}|, W_{ij}]$ **then**

$T^\sharp \leftarrow T \bigoplus$ Move_2(T,i,j)

if $\sum_{k \in v(T^\sharp)} r_k > \sum_{k \in v(T)} r_k$ **then**

$T \leftarrow T^\sharp$ and goto Step 1

end if

end if

end for

return T

Basically, given the incumbent solution T, the operator Move_1(T,i) (respectively, Move_2(T,i,j)) works as follows: delete the path(s) connecting leaf vertex i (respectively, leaf vertices i and j) from T (see (15) for how to delete a path connecting a leaf vertex), and then insert a certain number of new feasible candidate paths into T (determined by a dynamic programming subroutine, as detailed in subsection 3.2.3) to obtain a neighboring saturated BHS-tree. Let $lv(T)$ be the set containing all the leaf vertices of T and let $T \oplus$Move_k ($k=1,2$) be the neighboring tree obtained by applying operator Move_k to T, then we define two neighborhoods of T corresponding to these two move operators.

$$\begin{aligned} N(T,1) &= \{T \oplus \text{Move\_1}(T,i), i \in lv(T)\}, \\ N(T,2) &= \{T \oplus \text{Move\_2}(T,i,j), i,j \in lv(T), i<j\}. \end{aligned} \tag{1}$$

At each iteration of the neighborhood search procedure, the algorithm first examines in random order the neighboring solutions belonging to $N(T,1)$ and replaces the incumbent solution with the first met improving neighboring solution (i.e., first improvement strategy). If no improving solution exists in $N(T,1)$, the search continues with $N(T,2)$ until no improving solution is found in $N(T,1)$ and $N(T,2)$, meaning that a local optimum is reached.

To accelerate the neighborhood examination and avoid useless computation, we try to identify and discard the neighboring solutions that are considered irrelevant. For this purpose, we introduce below an original and effective pre-examination technique based on solving a closely related 0-1 knapsack problem.

3.2.1. Estimation Criterion for Pre-examination of Neighboring Solutions For a given solution, there are usually a large number of possible candidate neighboring solutions. However, it is unnecessary to actually generate at each iteration all candidate neighboring solutions, since many of them are (or are highly likely to be) irrelevant to the improvement of the incumbent solution (collecting more revenue). To explore this observation, we implement the following estimation criterion, with the help of solving a closely related 0-1 knapsack problem, to identify and discard these irrelevant neighboring solutions.

a) Given a saturated BHS-tree T, there are usually some reachable and profitable vertices uncollected by T (otherwise T can be returned as an optimal solution). Let V_{urp} denote the set containing all the uncollected reachable profitable vertices, i.e., for each vertex $j \in V_{urp}$, we have $j \notin v(T)$, $L(root,j,H) < \infty$, and $r_j > 0$. Now, we consider the

following sub-problem: if we delete one or two paths connecting some leaf vertices from T (to release some occupied budget), and then further insert some new paths connecting some vertices $j \in V_{urp}$ to T, which vertices should be chosen for insertion, so that the extra collected revenue would be maximized, subject to both the budget and hop constraints?

b) We transform this sub-problem to a 0-1 knapsack problem as follows: for each vertex $j \in V_{urp}$, we know its revenue r_j, as well as the cost of the hop-constrained shortest path between vertex j and the tree T, i.e., $\min(L(i, j, H - h_i), i \in v(T))$, where h_i is the number of edges between vertex $i \in v(T)$ and the root vertex. Now consider each vertex $j \in V_{urp}$ as an item of a 0-1 knapsack problem (for convenience, each item is assigned an individual index $k \in [1, |V_{urp}|]$) with a weight $w_k = \min(L(i, j, H - h_i), i \in v(T))$ and a gain $v_k = r_j$. Furthermore, let W denote the available budget after path deletion (up to two paths are allowed to be deleted), corresponding to the knapsack capacity, we get our 0-1 knapsack problem which requires to determine the items to add to the knapsack, so as to maximize the total gain (additionally collected revenue), while guaranteeing that the total weight (increased cost) does not exceed the given capacity W (available budget after deletion, in our case, W is always a positive integer). Relative to this transformed knapsack problem, we also need to consider some special cases which are discussed in Section 3.2.2.

c) We solve the transformed 0-1 knapsack problem to optimality by the following dynamic programming procedure (1). Let $Opt[k, W]$ be the optimal value that can be attained with total weight less than or equal to W, using items up to index k ($k \in [1, |V_{urp}|]$), then $Opt[k, W]$ can be calculated recursively as follows:

$$\begin{aligned} &Opt[0, W] = 0, \\ &Opt[k, 0] = 0, \\ &Opt[k, W] = Opt[k-1, W], \text{ if } w_k > W, \\ &Opt[k, W] = \max(Opt[k-1, W], Opt[k-1, W - w_k] + v_k), \text{ if } w_k \leq W. \end{aligned} \tag{2}$$

Now, for each iteration of our NS procedure, before actually generating any neighboring solution, we first run the above dynamic programming procedure to obtain all the possible values of $Opt[k, W], 1 \leq k \leq |V_{urp}|, 1 \leq W \leq W_{max}$, with a computational complexity of $O(|V_{urp}| \times W_{max})$, where W_{max} denotes the maximum available budget after deleting any two paths (since only up to two paths are allowed to be deleted, hence W_{max} is generally quite small). Then, these values of $Opt[k, W]$ are stored into a $|V_{urp}| \times W_{max}$ table, which

can be fetched directly during the current iteration of NS, instead of recalculating them repeatedly.

Inversely, after calculating and storing all the needed values of $Opt[k, W]$, given a capacity W of the knapsack, it is easy to determine the items belonging to the optimal solution of the corresponding 0-1 knapsack problem, by applying a backtrack procedure, within a computational complexity of $O(|V_{urp}|)$.

d) Based on the above preliminaries, before actually generating any candidate neighboring solution, we use the following criterion to estimate its prospect. Take a neighboring solution corresponding to deleting the path connecting leaf vertex i for example (the case is similar for deleting two paths). With the released budget W_i (corresponding to the lost revenue r_i), $Opt[|V_{urp}|, W_i]$ calculated by dynamic programming is the maximum extra revenue that can be expected. Thus if $r_i \geq Opt[|V_{urp}|, W_i]$, it is useless to generate the neighboring solution corresponding to deleting vertex i (only with few exceptions discussed below).

3.2.2. Special Cases While generating a neighboring solution of the incumbent solution T, we first delete one or two paths connecting some leaf vertices, and then attempt to insert a number of feasible candidate paths connecting some vertices belonging to V_{urp}, until T becomes saturated. During this process, the following three special cases (see Fig. 1 for some examples) may occur.

- Case 1: While inserting a path connecting vertex $j \in V_{urp}$, the pre-calculated (used in dynamic programming) originating at vertex $i \in v(T)$, with minimum $L(i, j, H - h_i)$, may have already been deleted. In this case, we need to re-calculate a hop-constrained shortest path between vertex j and the incumbent solution, whose cost is at least the pre-calculated cost. Here the values of $Opt[|V_{urp}|, W]$ obtained by dynamic programming may be overestimated.

- Case 2: If more than one path can be inserted sequentially using the released budget, for example, paths connecting vertices j_1 and j_2, then, when inserting the path connecting j_2, there may exist a shorter hop-constrained path than the pre-calculated one, between j_2 and a recently inserted vertex. Naturally, in this case, it is preferable to choose the shorter path in order to save budget, leading to a possible underestimation of the values of $Opt[|V_{urp}|, W]$.

- Case 3: As explained in (12), due to the hop-constraints, cycles may occur after inserting a path. In this case, we need to detect cycles and eliminate the edges causing cycles. Obviously, the budget saved by the eliminated edges could be used for further insertion, meaning that the values of $Opt[|V_{urp}|, W]$ may be somewhat underestimated.

These three special cases lead to possible wrong estimations, i.e., generating useless (non-improving) neighboring solutions in case 1, or rejecting improving neighboring solutions in cases 2 and 3. In the next subsection, we describe the technique to handle these special cases and present the acceptance criterion to ensure that the generated neighboring solutions are always saturated BHS-trees (to guarantee feasibility even in the worst case) and only the improving neighboring solutions have a chance to be accepted.

3.2.3. Generation of Neighboring Solutions Using the stored information obtained by dynamic programming, we implement as follows our Move_1(T, i) and Move_2(T, i, j) operators for generating promising neighboring solutions of the incumbent solution T.

Step1: Delete the path connecting leaf vertex i from T (for Move_1(T, i)), or delete the paths connecting leaf vertices i and j from T (for Move_2(T, i, j)).

Step 2: Denote by W_i (for Move_1(T, i)) or W_{ij} (for Move_2(T, i, j)) the available budget after deletion. Then, for each item k belonging to the optimal solution of the transformed 0-1 knapsack problem with capacity W_i or W_{ij}, we attempt to insert the path connecting the corresponding vertex $k \in V_{urp}$ into the incumbent solution. These items are identified by backtrack technique of 0-1 knapsack problem and sorted in an increasing order by the item index. Note that before insertion, we should update the hop-constrained shortest path between vertex k and the incumbent solution T (to ensure that the hop constraint is always satisfied), and then examine again if it violates the budget constraints, to guarantee feasibility even in the worst case like special case 1. If there is no budget constraint violation, we insert the selected path into the incumbent solution, with the renewed hop-constrained shortest path. Otherwise, we discard it and skip to the next item belonging to the optimal solution of the corresponding 0-1 knapsack problem, until no such item exists.

Step 3: Examine again whether the obtained solution is saturated. If this is not the case, we use the probabilistic construction procedure (15) to insert some extra paths into the incumbent solution, until the obtained tree is saturated in order to fully utilize the budget saved if special case two and/or three occurred.

We point out that the above two move operators are related to, but essentially different from the operators developed in (15). Indeed, in (15), after a path deletion, the new feasible candidate paths are probabilistically selected for insertion. By contrast, in this work, the new paths chosen for insertion are determined by dynamic programming, guided by the optimal solution of the corresponding 0-1 knapsack problem (step 2).

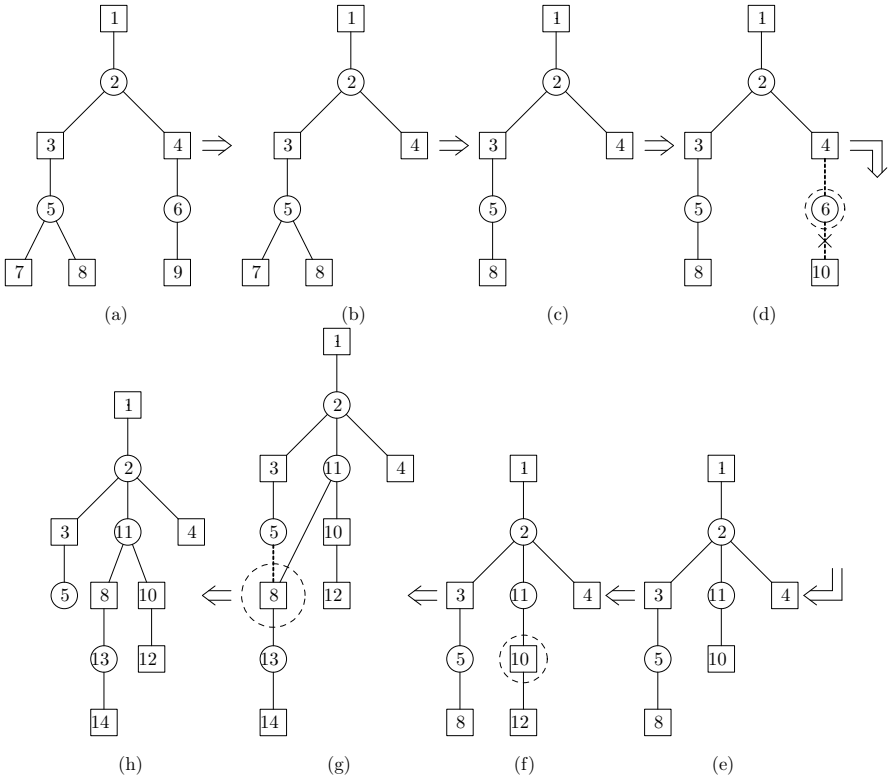

Figure 1 **The detailed process for generating a neighboring solution, corresponding to deleting two paths and inserting three new paths, taking into account the special cases**

Fig. 1 illustrates the process for generating a neighboring solution (with hop limit H=5), where the profitable vertices with $r_i > 0$ are drawn in boxes (i.e., vertices 1, 3, 4, 7, 8, 9, 10, 12, 14), and the others are drawn in circles (i.e., vertices 2, 5, 6, 11, 13). From the original solution Fig. 1a, the paths connecting leaf vertices 9 and 7 are deleted sequently, to obtain Fig. 1b and Fig. 1c. Subsequently, three new paths, respectively connecting profitable vertices 10, 12, 14, are inserted into Fig. 1c. Specifically, when inserting the path connecting vertex 10, the pre-calculated hop-constrained shortest path, originating at an already deleted vertex 6, is no longer available (special case 1, see Fig. 1d), hence we re-calculate a new path ($2 \rightarrow 11 \rightarrow 10$) for insertion (see Fig. 1e). Furthermore, while inserting

the path connecting vertex 12, a new path ($10 \rightarrow 12$) shorter than the pre-calculated one, is detected and chosen for insertion (special case 2, see Fig. 1f). Finally, after inserting the hop-constrained path connecting vertex 14 ($11 \rightarrow 8 \rightarrow 13 \rightarrow 14$, see Fig. 1g), a cycle occurs (special case 3). To destroy this cycle, we identify the vertex with two incoming edges, i.e., vertex 8, and eliminate the previously inserted incoming edge, i.e., edge $5 \rightarrow 8$, to get a feasible and saturated neighboring solution (a saturated BHS-tree) of the original solution (see Fig. 1h).

3.2.4. Acceptance Criterion After applying Move_1(T, i) or Move_2(T, i, j) to the incumbent solution T, a neighboring solution denoted by $T^\sharp$ is generated, which is always a feasible saturated BHS-tree and a possibly improved solution over T. Due to the possible wrong estimations of special case 1, we use the following criterion to decide whether $T^\sharp$ is accepted (see Algorithm 2, step 3 and step 4): if $T^\sharp$ collects a higher revenue than T, we accept $T^\sharp$ to replace T. Otherwise, we discard $T^\sharp$ and search another improving neighboring solution within the current neighborhood. When no improving solution can be found in $N(T, 1)$ and $N(T, 2)$, a local optimum is reached. At this point, the search turns to a recombination phase, as detailed in Section 3.3.

Generally, with the help of the dynamic programming driven estimation technique, each iteration of the neighborhood search procedure only generates very few neighboring solutions (in many cases only one, unless misjudgment of special case 1) instead of all the $O(|lv(T)|^2)$ candidates. As we show in Section 5.1, this technique drastically improves the efficiency of local optimization.

3.3. Backbone Based Crossover Operator

In addition to the neighborhood search procedure for local optimization, our memetic algorithm relies on a dedicated crossover operator to generate new solutions. The proposed crossover operator recombines two parent trees selected (at random) from the current population to generate one offspring solution (a saturated BHS-tree). It is based on the idea of transmitting the shared subtree (backbone) of parent trees to the offspring and using the probabilistic constructive procedure to obtain a saturated BHS-tree.

Definition 6: Given two parent trees T^1 and T^2, let T^s denote the subtree shared by both T^1 and T^2. The *backbone* T^b of T^1 and T^2 is the subtree of T^s such that each leaf vertex of T^b is necessarily a profitable vertex.

One notices that the backbone definition excludes non-profitable leaf vertices even if they are shared by both parents. The reason is that these vertices contribute nothing to the objective function while wasting some cost.

Fig. 2 shows an example where tree Fig. 2c is the backbone of trees Fig. 2a and Fig. 2b. One notices that each path in the backbone is shared by both parents, and every vertex of the backbone not only belongs to both parent trees, but also is connected to the root vertex by the same path in the two parents. In addition, note that although edge $\{5, 10\}$ belongs to the subtree shared by both parents, it is excluded from the backbone since vertex 10 is a non-profitable vertex.

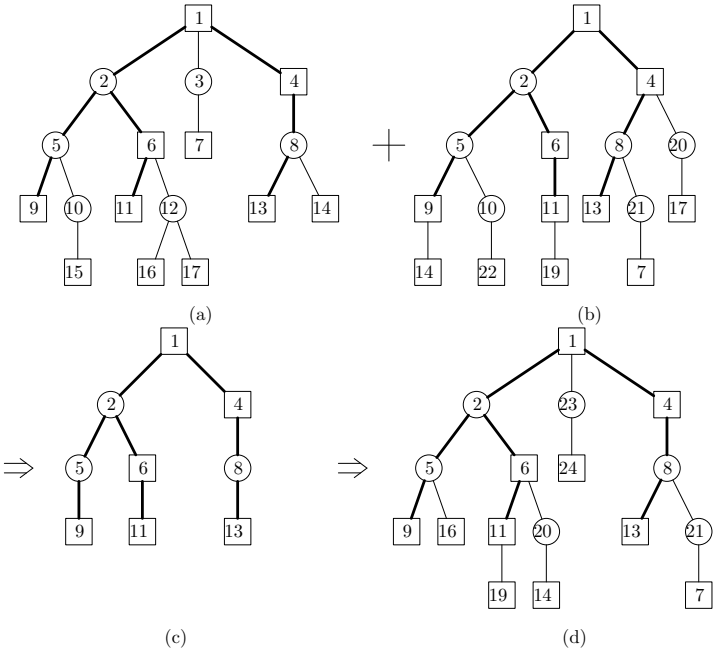

Figure 2 Backbone based crossover operator for generating an offspring solution which is a saturated BHS-tree

Based on the notion of backbone, our backbone-based crossover operator applies the following two steps to generate an offspring solution (i.e., a saturated BHS-tree).

Step 1: Identify the backbone T^b of the two selected parent trees at first. This can be done by examining all the profitable vertices and retaining the ones (associated with the corresponding paths) that not only belong to both the parents, but also are connected to the root vertex by the same path.

Step 2: Apply the probabilistic constructive procedure in (15) to iteratively add feasible candidate paths to T^b until it becomes saturated, leading to the offspring solution T^0.

In the example of Fig. 2, after the identification of the backbone Fig. 2c, applying the probabilistic constructive procedure leads to the offspring solution Fig. 2d (a saturated BHS-tree) including five newly inserted feasible paths.

For each new offspring solution, we apply again the NS procedure described above to improve its quality, and then use it to update the population by the quality-and-distance strategy described in the next subsection.

3.4. Pool Updating Strategy

To update the population with a given offspring solution, we apply the quality-and-distance procedure shown in Algorithm 3 which is inspired from the studies reported in (22, 25).

Algorithm 3 Quality-and-distance pool updating strategy

REQUIRE: Population $Pop = \{T^1, \ldots, T^Q\}$ and offspring solution T^0
RETURN: Updated population Pop
Insert T^0 into the population: $Pop \leftarrow Pop \cup \{T^0\}$
for each solution $T^a \in Pop$ **do**
    Calculate the distance between T^a and the population according to Eq. (5)
    Calculate the goodness score $g(T^a)$ of T^a according to Eq. (6)
end for
Identify the worst solution T^w with the lowest goodness score in Pop:
$T^w = arg\ \min\{g(T^a)|T^a \in Pop\}$
Remove the worst solution T^w from the population: $Pop \leftarrow Pop \backslash \{T^w\}$

Definition 7. Given two candidate solutions, coded by two one-dimensional vectors $T^a = \{t_1^a, \ldots, t_n^a\}$ and $T^b = \{t_1^b, \ldots, t_n^b\}$, the *distance* $D(T^a, T^b)$ *between* T^a *and* T^b is defined as:

$$D(T^a, T^b) = \sum_{i=1}^{n} d(t_i^a, t_i^b), \tag{3}$$

where $d(t_i^a, t_i^b)$ denotes the difference between the ith element of T^a and T^b such that:

$$d(t_i^a, t_i^b) = \begin{cases} 0, \text{ if } t_i^a = t_i^b, \\ 1, \text{ otherwise.} \end{cases} \tag{4}$$

Definition 8. Given a population $Pop = \{T^1, ..., T^Q\}$, as well as the distance $D(T^a, T^b)$ between any two solutions T^a and T^b $(T^a, T^b \in Pop, a \neq b)$, then the *distance between a solution* $T^a \in Pop$ *and the population* Pop, denoted by $D(T^a, Pop)$, is defined as the minimum distance between T^a and any other solution in the population, i.e.:

$$D(T^a, Pop) = \min\{D(T^a, T^b)|T^b \in Pop, b \neq a\}. \tag{5}$$

As shown in Algorithm 3, for population updating, the offspring T^0 is first inserted into the population ($Pop \leftarrow Pop \bigcup \{T^0\}$). Then, we use the following quality-and-distance goodness scoring function $g(T^a)$ from (22) to identify the worst solution T^w and remove it from Pop:

$$g(T^a) = \beta \times \widetilde{A}(f(T^a)) + (1-\beta) \times \widetilde{A}(D(T^a, Pop)), \tag{6}$$

where $f(T^a)$ is the objective value, i.e., the collected revenue and $\beta \in [0,1]$ is a parameter used to balance the relative importance of the quality criterion and the distance criterion. $\widetilde{A}(.)$ is the following normalized function:

$$\widetilde{A}(y) = \frac{y - y_{min}}{y_{max} - y_{min} + 1}. \tag{7}$$

Here y_{min} and y_{max} are respectively the minimum and maximum value of y, corresponding to all the solutions in the population. The term "+1" is used to avoid the possibility of a 0 denominator. With these definitions, the worst solution T^w is the one with the lowest quality-and-distance score $g(T^w)$. T^w is removed from Pop to make sure that Pop always contains Q (the population size) diversified solutions.

Notice the above quality-and-distance updating strategy ensures that an overall best solution (better than all the existing solutions in Pop) will always be added into Pop to replace some existing solution, since there is at least one solution in Pop (e.g., the solution closest to the overall best solution) with a lower quality-and-distance goodness score.

4. Computation Results and Comparisons

The proposed memetic algorithm is coded in C++. In order to assess its performance, we test it on a large set of STPRBH benchmark instances and compare the obtained results with the best known results published in the literature.

Typically, for each instance, we use the objective function, i.e., the collected revenue, as the main evaluation criterion, while including the CPU time for indicative purposes. Since the CPU times reported in the literature are based on different platforms, like in (15), we use the tool of the Standard Performance Evaluation Cooperation (SPEC, see www.spec.org) to harmonize the computing times. Our memetic algorithm is executed on the same platform as (15), i.e., an Intel Xeon E5440 2.83GHz processor (with a peak value of 25.5, according to the SPEC) and 2GB RAM, while an AMD Opteron machine with 2.39GHz CPU (with a peak value of 18.7) and 2GB RAM was used in (10, 12, 13), and

an Intel Xeon E5540 2.53GHz processor (with a peak value of 29.4) and 3GB RAM was used in (30). Respectively, in the following subsections, we multiply the CPU times of (10, 12, 13) by 0.73 ($\frac{18.7}{25.5}$) and the CPU times of (30) by 1.15 ($\frac{29.4}{25.5}$), while the CPU times of our computer serve as a baseline.

4.1. Benchmark Instances and Experimental Protocol

For our experiments, we use 384 well-known STPRBH instances (10, 12, 13, 15, 21, 30) and 30 newly generated challenging instances[1]. The 384 existing instances correspond to 58 adapted Steiner graphs from the series B and C of the OR-Library (2). For the sake of clarity, we classify the 384 existing instances into four groups G1-G4 as follows, and use our 30 new instances to form group G5.

- Group G1 contains 144 small or mid-sized instances, corresponding to the 18 graphs of series B, which have all been solved to optimality by previous exact algorithms (13, 30). A subset (54 out of 144) of these instances are also solved to optimality by (21).
- Group G2 contains 60 larger instances, corresponding to the first 10 graphs of series C (steinc1_10, ..., steinc5_10, steinc1_100, ..., steinc5_100, hereafter, we rename steinc$i$_j by Ci_j), which have also been solved to optimality by exact algorithms (13, 30). A small subset (10 out of 60) of these instances are solved to optimality by (21). Note that the existing exact algorithms can only solve the above two groups of instances to optimality. For the following two groups of larger instances, only solutions from heuristics (lower bounds) are available.
- Group G3 contains 124 instances of series C, which have all been solved to optimality (with all the reachable profitable vertices collected) by previous heuristics (10, 12, 15), due to their large enough allowed budget.
- Group G4 contains the remaining 56 instances of series C, for which the optimal solutions remain unknown, due to their large problem size (thus previous exact algorithms cannot terminate within reasonable time) and restrict budget constraint (thus it is difficult for heuristics to collect all the reachable profitable vertices). They are the most challenging instances among all the 384 existing instances.

[1] The adapted STPRBH graphs as well as the best results found in this paper on all these five groups of 414 instances are available at `http://www.info.univ-angers.fr/pub/hao/stprbh-memetic.html`, while the initial Steiner graphs are available at `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html`

• Group G5 contains the new set of 30 challenging instances that we generated. Among the 40 graphs of series C of the OR-Library, the 10 largest ones (C16_10,...,C20_10,C16_100,...,C20_100) with 500 vertices and 12500 edges have been used to create instances of group G3. However, these instances are easy due to their large allowed budget. To make these instances difficult, we can reduce the allowed budget. Based on this idea, from each of these 10 graphs, we generate three new instances by reducing the allowed budget and imposing different hop limits (H=5, 15, 25). Given that the allowed budget B is represented as $\frac{\sum_{\{i,j\}\in E} c_{ij}}{b}$, for graphs C16_10 and C16_100, we set b=10000; For graphs C17_10 and C17_100, we set b=5000; For the remaining six graphs (C18_10, C18_100, C19_10, C19_100, C20_10, C20_100), we set b=1000. Given that these new instances are of large size and have strict budget constraints, we believe that their optima are difficult to prove.

In the following subsections, we first summarize the results obtained by our memetic algorithm on the 328 instances with known optimal solutions (Groups G1-G3), and then report the detailed results on the 56 instances of group G4 with unknown optimal solutions. Subsequently, we provide the detailed results of our memetic algorithm with respect to the latest heuristic BLS (15) on the 30 new instances of group G5. Finally, we summarize the results of all the five groups of instances in a graphical way, in order to highlight the competitiveness of our memetic algorithm with respect to previous heuristics.

For the 384 instances of G1-G4, the reference algorithms (TS(2000) and TS(10000) in (10, 12), BLS in (15)) report their results on a basis of ten independent runs per instance. For a fair comparison, we also independently run our memetic algorithm ten times for these instances even though more runs offer more opportunity to find better solutions. Moreover, like (15), each run of our memetic algorithm is limited to a maximum of 12 minutes, thus at most two hours is allowed for solving each instance. Whenever we show computing times for our memetic algorithm, we give the accumulated total time (in seconds) of the ten independent runs. In (10, 12), the computing time of one run (more precisely, the average computing time over ten runs) of TS(2000) and TS(10000) was reported (confirmed by the author of (10)), thus we multiply their corresponding times by ten in our comparative studies.

Table 1 Statistical results based on 20 sample instances corresponding to different parameter values

Parameter Q			Parameter M			Parameter β		
Parameter Value	Mean Rank	Mean Time(s)	Parameter Value	Mean Rank	Mean Time(s)	Parameter Value	Mean Rank	Mean Time(s)
10	6.5	70.37	10	9.9	9.69	0.1	5.9	80.38
20	**4.2**	**69.66**	20	8.8	10.38	0.2	5.6	69.57
30	6.7	49.38	30	7.85	11.30	0.3	5.9	70.66
40	5.5	49.44	50	7.15	12.81	0.4	4.5	67.46
50	5.9	50.71	100	5.7	20.69	0.5	4.75	71.50
60	4.45	51.46	200	4.55	31.05	**0.6**	**3.2**	**69.66**
70	5.05	77.16	300	4.5	44.88	0.7	5.05	84.11
80	4.9	98.16	**500**	**2.6**	**69.66**	0.8	6.1	78.51
90	5.95	97.28	1000	2	73.53	0.9	4.85	83.07
100	4.4	87.48	2000	1.5	138.92	1.0	6.2	78.84

4.2. Parameter Setting

Our memetic algorithm requires three parameters: Q - the population size (Section 3.1), M - served as one of the termination criteria (the search terminates once the best found solution cannot be further improved after M consecutive generations), and β - the coefficient used in the goodness function of Eq. (6) for pool management. These parameters are independent from each other, thus we study them separately.

For each parameter, we implement ten different scenarios of our memetic algorithm by varying the chosen parameter within a reasonable range, while fixing the other two parameters with their default values (Q=20, M=500, β=0.6 by default). Then we compare their statistical performances based on a sample of 20 instances randomly taken from the 56 most challenging instances. Specifically, we independently run each scenario (e.g., scenario i) ten times to solve each sample instance (e.g., instance j), and then record the average collected revenue associated with the accumulated CPU time (in seconds). After that we calculate the rank $\lambda_{ij} = \chi + 1$ of scenario i for solving instance j, where χ denotes the number of scenarios (among the ten compared scenarios of a particular parameter) which perform better than scenario i, in terms of average collected revenue. Based on this, the mean rank of scenario i for solving all the 20 sample instances is defined as $\lambda_i = \frac{\sum_{j=1}^{20} \lambda_{ij}}{20}$. Intuitively, a lower value of λ_i indicates a better statistical performance of scenario i for solving the 20 sample instances, with respect to other compared scenarios. Table 1 gives the mean rank of each test scenario (classified into three groups corresponding to each parameter), as well as the average accumulated CPU time for each instance, based on which we determine a proper value for each parameter.

Parameter Q: We vary Q within the range $[10, 100]$ with a step of 10 and statistic the mean rank and mean CPU time of each scenario in Table 1 (columns 1-3). Although it is not easy to draw a definitive conclusion from these comparisons, we observe that $Q = 20$

yields a good mean rank within a reasonable mean computing time, thus we adopt a population with 20 solutions.

Parameter M: We vary M within the range $[10, 2000]$ with variable steps and compare their performances. In general, a larger value of M could lead to a better result, but would require a high computing time. As shown in Table 1 (columns 4-6), there is no value of parameter M which completely dominates other values in terms of both solution quality and computing speed. To exploit a good trade-off between quality and efficiency, we set M=500 as the default value in our memetic algorithm, which leads to results of good quality, and requires competitive computational times for each group of instances. We also tested M=1000 during preliminary experiments and obtained slightly better results. However, the average computational time on group G1 became a little more than the latest heuristic BLS (executed on the same platform).

Parameter β: We vary β within the range $[0.1, 1]$ with a step of 0.1 to get ten test scenarios. Table 1 (columns 7-9) discloses that the scenarios with mid values of β ($\beta \in [0.4, 0.6]$) perform statistically better than those with large ($\beta \in [0.7, 1.0]$) or small ($\beta \in [0.1, 0.3]$) values. This indicates that both criteria (i.e., quality and distance) are important to maintain a health population. Following this observation, we choose 0.6 as the default value of β, which yields the best mean rank within a comparative mean CPU time.

4.3. Summarized Results of Groups G1-G3 with Known Optima

For the first three groups of 328 instances with known optima, we summarize in Table 2 the results of our memetic algorithm together with the results of six exact algorithms (S1, S3, S5 (13), BP1, BP2, BP3 (30)) and four heuristics (destroy-and-repair (D&R for short), TS(2000), TS(10000) (10, 12) and BLS (15)) in the literature. In Table 2. For each group of instances, column 'Match' indicates the number of cases for which the optimal value is reached by each method, and column 'Miss' indicates the number of cases where an exact approach cannot terminate within the allowed time, i.e., $7200 \times \frac{18.7}{25.5} \approx 5256$ seconds (after harmonizing by SPEC) in (13) and $10000 \times \frac{29.4}{25.5} \approx 11529$ seconds in (30), or a heuristic approach fails to reach the optimal value. Columns 'Mean Gap' and 'Mean Time' list the mean gaps (only for the instances missing optimality) between the optimal solutions and the best solutions of a heuristic algorithm (exact algorithms can always reach an optimal solution, unless they fail to terminate within the time limit) and the mean times in seconds used by a method (for all the test instances). The objective values of the reference

Table 2 Summarized comparative results for groups G1-G3 with known optimal solutions

Method	Group G1 (144 instances)				Group G2 (60 instances)				Group G3 (124 instances)			
	Match	Miss	Mean Gap	Mean Time(s)	Match	Miss	Mean Gap	Mean Time(s)	Match	Miss	Mean Gap	Mean Time(s)
S1	144	0	-	1.12	59	1	-	190.38	-	-	-	-
S3	144	0	-	1.05	59	1	-	250.26	-	-	-	-
S5	117	27	-	1093.95	36	24	-	2164.78	-	-	-	-
BP1	144	0	-	2.63	58	2	-	766.33	-	-	-	-
BP2	144	0	-	1.44	59	1	-	443.01	-	-	-	-
BP3	144	0	-	1.41	59	1	-	376.65	-	-	-	-
D&R	91	53	3.07%	0.06	34	26	8.09%	2.26	124	0	-	434.88
TS(2000)	97	47	1.59%	2.11	29	31	5.36%	30.05	122	2	6.24%	599.11
TS(10000)	108	36	0.96%	9.82	30	30	4.26%	145.55	122	2	6.24%	3180.89
BLS	133	11	1.00%	1.71	43	17	1.48%	79.51	124	0	-	113.53
Memetic	**144**	0	-	1.27	**50**	10	1.57%	14.46	124	0	-	4.73

algorithms are extracted from the corresponding papers (after some transformation corresponding to TS(2000) and TS(10000)), while the computation times are harmonized using SPEC. The unavailable items are marked as '-' in Table 2.

As shown in Table 2, the 144 small or mid-sized instances of group G1 can all be solved to optimality by the previous exact approaches (only except S5), with a short mean time ranging from 1.05s to 2.63s (excluding the mean time of S5). Meanwhile, the previous best heuristic BLS [2] misses 11 optimal solutions, with a mean gap of 1.00% and a mean time of 1.71s. For comparison, our memetic algorithm can solve all these 144 instances to optimality, with a competitive mean time of 1.27s.

For the 60 larger instances of group G2, the previous best exact approach S1 can solve 59 instances to optimality with a best mean time of 190.38s, while the current best heuristic BLS misses 17 optimal solutions, with a mean gap of 1.48% and a mean time of 79.51s. For comparison, our memetic algorithm misses ten optimal results out of these 60 instances, with a mean gap of 1.57% and a short mean time of 14.46s.

For the 124 instances of group G3, one observes that they cannot be solved by existing exact algorithms (marked as '-' in the top-right part of Table 2), due to their large size. However, these instances are not difficult for heuristics to reach optimality, due to their large allowed budget. The previous heuristics D&R and BLS can both reach all the optimal values, with a mean time of 434.88s and 113.53s respectively. For comparison, our memetic algorithm can also solve them to optimality, with a much shorter mean time of 4.73s.

Finally, we mention that the authors of (21) recently developed three MIP formulations (MTZ, MTZ-L, RLT) and tested them on a subset of group G1 (54 out of 144 instances) and G2 (10 out of 60 instances). Experiments based on an Intel core 2 Duo processor with

[2] In (15), the results of BLS for these 144 easy instances are omitted. For the sake of comparison, we produced the shown results using the source code available at `http://www.info.univ-angers.fr/pub/hao/stprbh.html`

3.0 GB RAM show that, their approach can solve the selected 64 instances to optimality, with an average CPU time of 5.78s for the 54 instances of G1 (using the RLT formulation), and 52.13s for the 10 instances of G2 (using the MTZ-L formulation).

4.4. Detailed Results of the 56 Challenging Instances of Group G4

We now turn our attention to group G4 which contains 56 most challenging instances with unknown optima. The obtained results are shown in Table 3, where the first five columns respectively indicate the graph name, the number of vertices $|V|$ and edges $|E|$, the budget limit $B = \frac{\sum_{\{i,j\}\in E} c_{ij}}{b}$ (given the b values) and the hop limit H, while the following eight columns list the results reported by the previous heuristics. Respectively, columns 6-7 indicate the collected revenue R and the consumed CPU time $t(s)$ of D&R. Columns 8-13 indicate the best collected revenue R^{best} among ten independent runs and the accumulated CPU time $t(s)$ of three other heuristics: TS(2000), TS(10000) and BLS. As mentioned above, the CPU times of D&R, TS(2000) and TS(10000) in (10, 12) have all been harmonized by SPEC (i.e., multiplied by $\frac{18.7}{25.5} \approx 0.73$). The last 5 columns show the results of our memetic algorithm, including the best (R^{best}) and the average (R^{avg}) of the collected revenue among ten independent runs, the times that our memetic algorithm improves (column $\surd$) or matches (column =) the best known result among the ten runs (in terms of best collected revenue), and the total accumulated CPU time $t(s)$ for the ten runs. The values in **bold** indicate the best results among all the results, while the values in *italic* indicate that our memetic approach matches the best known results.

As shown in Table 3, for these 56 challenging instances, our memetic algorithm succeeds in improving **45** and matching four best known results (reported by all the compared heuristics), while missing seven best known results. The mean improvement over the best known results is 1.36%. On the other hand, the mean CPU time consumed by our memetic algorithm is 63.04s, while the CPU times of D&R, TS(2000), TS(10000) are 12.37s, 48.52s, 232.12s respectively (after harmonizing by SPEC), and the mean CPU time of BLS is 369.99s. Additionally, one can observe from Table 3 (column 16) that for 22 out of 56 instances, our memetic algorithm finds an improved solution during each of the ten independent runs. This indicates that for these 22 instances, one single run (i.e., with one tenth of the reported time) of our algorithm suffices to find an improved best known result.

Furthermore, we use the Friedman test to assess the observed differences between the memetic algorithm and the compared heuristics. The Friedman test reveals a p-value=7.24×10^{-14}, 1.21×10^{-13}, 8.90×10^{-13}, 2.91×10^{-8} between the best results from our

Table 3 Detailed results of the memetic algorithm on the 56 most challenging instances with unknown optima

Instance					D&R		TS(2000)		TS(10000)		BLS		Memetic				
Graph	$\|V\|$	$\|E\|$	b	H	R	$t(s)$	R^{best}	$t(s)$	R^{best}	$t(s)$	R^{best}	$t(s)$	R^{best}	R^{avg}	$\surd$	$=$	$t(s)$
$C8\_10$	500	1000	20	5	208	1.32	≤228	14.2	≤229	65.4	228	81.69	**230**	230.0	**10**	0	19.06
$C8\_10$	500	1000	50	5	104	0.12	**≤116**	13.6	**≤116**	65.2	**116**	30.95	*116*	116.0	0	*10*	10.80
$C8\_10$	500	1000	20	15	303	9.51	≤308	42.0	≤319	206.2	326	172.04	**330**	327.8	**8**	1	42.37
$C8\_10$	500	1000	50	15	152	1.91	≤161	41.6	≤166	199.1	167	34.63	**171**	168.5	**7**	3	23.03
$C8\_10$	500	1000	20	25	311	16.23	≤310	65.6	≤319	324.0	328	150.98	**330**	329.0	**7**	3	35.60
$C8\_10$	500	1000	50	25	151	3.19	≤160	63.7	≤166	311.6	171	42.21	**172**	171.6	**6**	4	31.50
$C8\_100$	500	1000	20	5	2261	1.28	≤2365	13.5	≤2368	70.3	2341	100.73	**2380**	2374.0	**6**	0	24.37
$C8\_100$	500	1000	50	5	1151	0.23	≤1204	13.5	**≤1220**	66.4	1201	31.26	1216	1216.0	0	0	12.30
$C8\_100$	500	1000	20	15	3269	9.24	≤3237	43.1	≤3306	202.8	3378	204.17	**3418**	3407.6	**10**	0	47.39
$C8\_100$	500	1000	50	15	1696	1.88	≤1675	41.8	≤1705	199.9	1763	41.04	**1774**	1759.6	**6**	0	26.46
$C8\_100$	500	1000	20	25	3310	16.87	≤3337	66.8	≤3340	315.3	3392	187.76	**3443**	3426.5	**10**	0	48.16
$C8\_100$	500	1000	50	25	1735	1.79	≤1742	64.3	≤1755	311.0	1780	42.24	**1792**	1784.2	**7**	2	33.34
$C9\_10$	500	1000	20	5	274	2.26	≤279	16.0	≤283	78.5	284	229.10	**302**	299.3	**10**	0	29.31
$C9\_10$	500	1000	50	5	143	0.26	≤145	16.1	≤146	76.9	147	32.67	**149**	149.0	**10**	0	16.04
$C9\_10$	500	1000	20	15	354	11.35	≤349	42.4	≤354	195.7	**376**	162.46	*376*	372.4	0	*1*	42.45
$C9\_10$	500	1000	50	15	172	2.36	≤165	39.3	≤169	192.9	181	49.24	**183**	180.8	**4**	1	34.77
$C9\_10$	500	1000	20	25	353	9.45	353	63.0	353	309.3	**379**	164.72	377	374.7	0	0	54.65
$C9\_10$	500	1000	50	25	168	3.70	≤167	59.1	≤172	301.2	183	47.65	**186**	184.8	**8**	2	43.77
$C9\_100$	500	1000	20	5	2864	2.24	≤2933	16.4	≤2954	77.8	2921	225.13	**3112**	3092.8	**10**	0	38.16
$C9\_100$	500	1000	50	5	1514	0.50	≤1509	15.8	≤1533	83.9	1536	42.98	**1563**	1563.0	**10**	0	17.57
$C9\_100$	500	1000	20	15	3642	11.15	≤3588	41.5	≤3631	194.3	**3904**	188.66	3873	3854.5	0	0	56.49
$C9\_100$	500	1000	50	15	1675	2.13	≤1742	38.6	≤1732	192.7	**1883**	54.23	1879	1863.5	0	0	36.32
$C9\_100$	500	1000	20	25	3602	17.26	≤3644	62.1	≤3673	309.2	**3926**	212.18	3906	3881.4	0	0	71.83
$C9\_100$	500	1000	50	25	1674	3.35	≤1745	59.4	≤1753	296.7	1892	51.18	**1909**	1898.6	**7**	1	50.11
$C10\_10$	500	1000	20	5	341	2.07	≤371	14.4	≤372	68.6	385	458.71	**388**	387.4	**10**	0	57.24
$C10\_10$	500	1000	50	5	156	0.39	≤173	13.9	≤175	67.7	184	62.49	**185**	184.1	**1**	9	26.66
$C10\_10$	500	1000	20	15	505	13.07	≤505	41.4	≤509	203.0	545	134.52	**553**	545.2	**6**	0	113.51
$C10\_10$	500	1000	50	15	211	2.54	≤221	40.4	≤226	198.8	245	71.89	**247**	247.0	**10**	0	61.70
$C10\_10$	500	1000	20	25	482	19.86	≤501	67.2	≤513	313.8	547	102.18	**561**	555.4	**10**	0	133.69
$C10\_10$	500	1000	50	25	219	3.83	≤218	62.9	≤229	311.2	**254**	80.70	*254*	249.6	0	*1*	78.51
$C10\_100$	500	1000	20	5	3530	1.88	≤3811	15.5	≤3863	68.3	4027	463.66	**4069**	4047.9	**8**	0	67.42
$C10\_100$	500	1000	50	5	1513	0.32	≤1836	14.0	≤1858	67.9	1937	84.66	**1940**	1939.3	**10**	0	34.83
$C10\_100$	500	1000	20	15	5163	13.30	≤5227	45.5	≤5253	202.0	**5687**	138.55	5686	5620.0	0	0	130.25
$C10\_100$	500	1000	50	15	2415	2.88	≤2365	39.7	2356	198.4	2555	67.17	**2601**	2552.1	**1**	0	71.46
$C10\_100$	500	1000	20	25	5287	22.97	≤5286	64.9	≤5331	321.3	5642	131.01	**5773**	5694.6	**9**	0	139.72
$C10\_100$	500	1000	50	25	2472	4.66	≤2432	63.8	2422	311.9	2511	86.82	**2632**	2576.1	**10**	0	92.41
$C13\_10$	500	2500	100	5	242	2.67	≤243	26.6	≤246	124.7	248	135.87	**253**	251.1	**10**	0	22.97
$C13\_10$	500	2500	100	15	303	12.57	≤296	72.8	≤302	327.4	308	56.55	**316**	312.0	**10**	0	33.71
$C13\_10$	500	2500	100	25	302	22.19	≤298	107.3	≤302	520.3	307	46.47	**314**	311.6	**10**	0	37.38
$C13\_100$	500	2500	100	5	2507	2.48	≤2526	25.9	≤2544	134.4	2558	168.50	**2622**	2593.0	**9**	1	29.87
$C13\_100$	500	2500	100	15	3064	11.64	≤3029	70.4	≤3111	326.4	3236	59.53	**3255**	3242.7	**7**	0	41.05
$C13\_100$	500	2500	100	25	3064	22.72	≤3057	109.8	≤3136	521.3	3248	64.68	**3260**	3241.2	**1**	0	42.62
$C14\_10$	500	2500	100	5	344	5.33	339	26.8	≤341	130.3	367	144.91	**370**	366.0	**3**	2	29.95
$C14\_10$	500	2500	100	15	377	17.72	≤370	71.4	≤371	312.7	**399**	47.99	398	394.7	0	0	55.08
$C14\_10$	500	2500	100	25	377	28.02	369	101.8	≤373	493.2	**397**	58.78	*397*	393.8	0	*2*	53.99
$C14\_100$	500	2500	100	5	3485	5.02	≤3416	26.3	≤3503	124.2	3824	158.45	**3847**	3823.7	**3**	0	36.42
$C14\_100$	500	2500	100	15	3846	17.32	≤3872	71.5	≤3929	312.9	4122	68.00	**4172**	4150.2	**9**	0	72.46
$C14\_100$	500	2500	100	25	3846	27.43	≤3856	102.2	≤3897	495.3	4120	72.65	**4150**	4121.6	**4**	0	76.56
$C15\_10$	500	2500	20	5	1174	79.48	≤1192	28.9	≤1211	130.0	1213	7295.82	**1221**	1218.2	**10**	0	188.94
$C15\_10$	500	2500	100	5	401	4.94	≤427	26.2	≤435	126.4	451	196.81	**462**	456.0	**8**	1	71.42
$C15\_10$	500	2500	100	15	515	23.48	≤501	71.0	≤502	324.1	548	93.39	**558**	557.5	**10**	0	92.59
$C15\_10$	500	2500	100	25	520	38.60	510	107.7	510	506.6	546	77.24	**558**	556.4	**10**	0	99.20
$C15\_100$	500	2500	20	5	12078	83.31	≤12198	28.5	≤12306	131.5	12213	7023.82	**12392**	12366.5	**10**	0	359.13
$C15\_100$	500	2500	100	5	4180	5.17	≤4358	27.1	≤4379	125.9	4378	271.20	**4807**	4758.0	**10**	0	97.01
$C15\_100$	500	2500	100	15	5355	24.04	≤5177	70.8	≤5159	326.6	5770	106.43	**5807**	5793.8	**10**	0	109.24
$C15\_100$	500	2500	100	25	5393	41.07	5243	106.7	≤5274	524.9	5703	110.16	**5822**	5795.7	**10**	0	127.45

memetic algorithm and those reported by D&R, TS(2000), TS(10000), BLS respectively, meaning that the differences are statistically significant. Precisely, our memetic algorithm dominates BLS (the latest and best performing heuristic) and TS(10000), in terms of both

Table 4 Detailed results of the memetic algorithm compared to BLS for the 30 newly generated instances

Instance			BLS			Memetic			Instance			BLS			Memetic		
Graph	b	H	R^{best}	R^{avg}	$t(s)$	R^{best}	R^{avg}	$t(s)$	*Graph*	b	H	R^{best}	R^{avg}	$t(s)$	R^{best}	R^{avg}	$t(s)$
$C16\_10$	10000	5	*19*	16.80	24.32	*19*	19.00	15.22	$C16\_100$	10000	5	*203*	179.48	28.88	*203*	203.00	15.18
$C16\_10$	10000	15	*19*	17.00	24.47	*19*	19.00	15.35	$C16\_100$	10000	15	*203*	180.90	35.15	*203*	203.00	15.30
$C16\_10$	10000	25	*19*	16.70	26.97	*19*	19.00	15.50	$C16\_100$	10000	25	*203*	175.15	32.22	*203*	203.00	15.48
$C17\_10$	5000	5	*47*	47.00	42.60	*47*	47.00	22.69	$C17\_100$	5000	5	*481*	481.00	46.09	*481*	481.00	24.18
$C17\_10$	5000	15	*50*	47.37	37.06	*50*	49.25	23.06	$C17\_100$	5000	15	*513*	484.43	38.84	*513*	491.24	23.09
$C17\_10$	5000	25	*50*	47.55	37.43	*50*	49.28	23.54	$C17\_100$	5000	25	*513*	484.66	39.04	*513*	495.08	23.75
$C18\_10$	1000	5	311	295.32	1430.08	**312**	307.90	196.02	$C18\_100$	1000	5	3245	3042.33	1489.26	**3252**	3229.20	215.36
$C18\_10$	1000	15	334	323.27	357.29	**338**	335.57	151.70	$C18\_100$	1000	15	3478	3383.57	384.12	**3526**	3499.54	202.73
$C18\_10$	1000	25	334	324.00	362.98	**338**	335.90	160.55	$C18\_100$	1000	25	3506	3379.53	382.05	**3526**	3496.48	191.35
$C19\_10$	1000	5	390	372.44	1242.58	**392**	386.80	208.06	$C19\_100$	1000	5	4021	3865.49	1403.18	**4080**	4011.77	286.92
$C19\_10$	1000	15	412	395.48	433.09	**417**	412.08	209.10	$C19\_100$	1000	15	4265	4131.19	562.51	**4355**	4291.64	296.54
$C19\_10$	1000	25	407	395.07	416.19	**418**	412.04	204.13	$C19\_100$	1000	25	4265	4129.64	570.23	**4337**	4288.82	283.50
$C20\_10$	1000	5	**441**	431.29	2156.18	436	429.70	364.05	$C20\_100$	1000	5	4527	4412.83	2530.84	**4565**	4518.63	503.57
$C20\_10$	1000	15	**486**	465.85	590.30	479	473.33	397.09	$C20\_100$	1000	15	**5124**	4878.30	827.61	5019	4970.74	487.43
$C20\_10$	1000	25	**487**	467.75	604.34	481	472.75	390.01	$C20\_100$	1000	25	5045	4882.87	767.82	**5054**	4975.50	497.47

solution quality and runtime. The memetic algorithm achieves solutions of much better quality than TS(2000) and D&R, while consuming some more average computation time.

In order to check whether our memetic algorithm can still achieve competitive results with a reduced computing effort, we re-run our memetic algorithm with $M = 50$ (instead of M=500). Experimental results [3] shown in Table 1 of the online supplement indicate that the memetic algorithm with $M = 50$ still yields results better than or equal to both TS(2000) and D&R on each of these 56 instances, with a mean improvement of 6.47% and 8.25% respectively and a mean CPU time of 11.88s, being less than the mean times of TS(2000) and D&R. This confirms that our memetic algorithm dominates TS(2000) and D&R, not only in terms of solution quality, but also in terms of computation time.

4.5. Computational Results of the 30 New Benchmark Instances of Group G5

For each of the 30 newly generated instances of group G5, we run our memetic algorithm (with its default parameters) and the current best heuristic BLS (using the source code available at `http://www.info.univ-angers.fr/pub/hao/stprbh.html`) 100 times, and then compare their performances in Table 4, including the best (R^{best}) and average (R^{avg}) collected revenues as well as the computing time $t(s)$. As shown in Table 4, the memetic algorithm respectively improves, matches, misses 14, 12, 4 best results obtained by BLS, with a mean improvement of 0.32%, while the Friedman test reveals a p-value=1.84×10^{-2} (indicating a significant statistical difference). Second, for the average results, the memetic algorithm respectively improves, matches, misses 27, 2, 1 average results compared to BLS, with a significant mean improvement of 4.94% and a p-value of 8.94×10^{-7} (again

[3] Available online at `http://www.info.univ-angers.fr/pub/hao/stprbh-memetic.html`

a statistically significant difference is detected). On the other hand, the mean computing time consumed by the memetic algorithm (182.60s) is much less than the computing time of BLS (564.12s). Thus we conclude that for these 30 new instances, the memetic algorithm performs much better than BLS both in terms of solution quality and computing time.

4.6. Overall Comparison with Respect to the Previous Heuristics

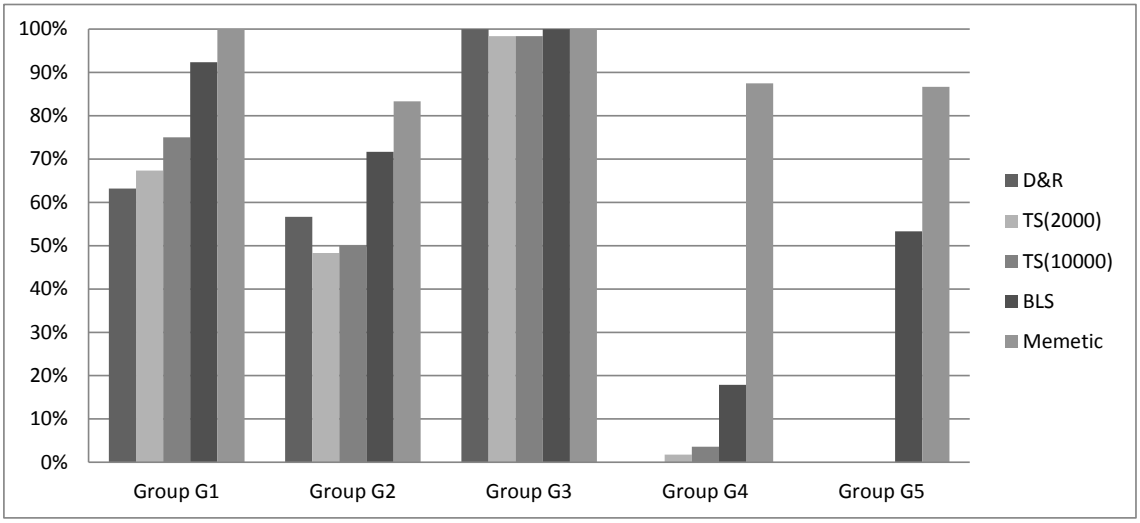

Figure 3 **For each compared heuristic, we show the percentage of instances for which the optimal solutions (groups G1-G3) or the best known results (including those updated by the memetic algorithm) are reached (groups G4 and G5). A higher percentage corresponds to a better performance.**

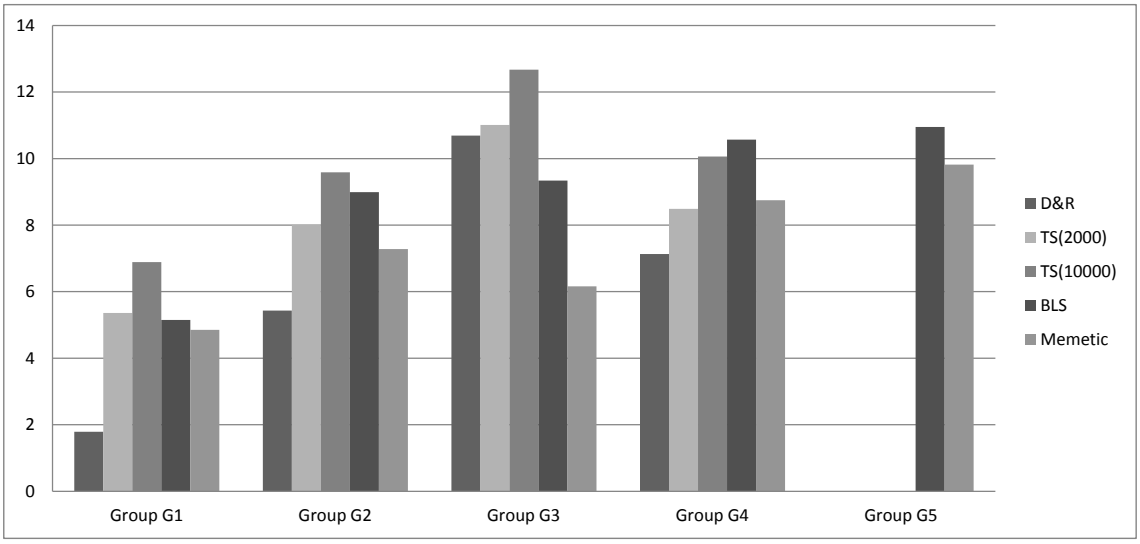

Figure 4 **For each compared heuristic, we show the average CPU time X in seconds (expressed as $ln(100X)$) for solving each group of instances. A lower value of $ln(100X)$ corresponds to a better performance.**

In this section, we try to provide an overall comparison between our memetic algorithm and the existing heuristics. For this purpose, we illustrate in a graphical way the overall performances (based on all the five groups of instances) of our memetic algorithm with respect to the previous heuristics D&R, TS(2000), TS(10000) and BLS. Respectively, Fig. 3 gives the percentage of instances for which each heuristic can reach the optimal solutions (for groups G1-G3) or the overall best known results (including those updated by the memetic algorithm) (for groups G4 and G5). Fig. 4 gives the average CPU time X in seconds (in logarithmic scale, expressed as $ln(100X)$) of each heuristic for solving each group of instances. The logarithmic scale $ln(100X)$ is used to convert the mean CPU times from a large range (from 0.06s to 3180.89s) to a reasonable range (from 1.79 to 12.67). Note that for group G5, only results of BLS and the memetic algorithm are available.

Fig. 3 and Fig. 4 show that our algorithm reaches the highest percentage of optimal or best known results on each group and thus outperforms the compared heuristics in terms of solution quality. Moreover, our algorithm consumes statistically much shorter mean CPU time than BLS and TS(10000) on each group, while requiring slightly more time than D&R and TS(2000) on groups G1, G2 and G4. As shown in Section 4.4, running the memetic algorithm with M=50 outperforms D&R and TS(2000) both in terms of solution quality and run time on the most challenging group G4. These comparisons clearly indicate the competitiveness of our memetic algorithm with respect to the existing heuristics.

5. Discussions

5.1. Impact of the Dynamic Programming Driven Estimation Criterion

To assess the impact of the dynamic programming driven estimation criterion, we create a variant of our memetic algorithm by disabling this technique (call this variant Memetic/DP). In Memetic/DP, we use a basic neighborhood search procedure which, at each iteration, may actually generate all the candidate neighboring solutions ($O(|lv(T)|^2)$ in total) to seek an improving neighboring solution. We compare Memetic/DP and our standard memetic algorithm on the base of the 56 most challenging instances of group G4 (each instance is solved ten times by each algorithm using the same parameters and under the same termination condition, i.e. with Q=20, M=500, β=0.6 and a time limit of 12 minutes per run). The detailed results are provided in Table 2 of the online supplement, where the meanings of the columns are the same as in Table 4.

The results show that Memetic/DP yields globally similar or slightly worse results in terms of solution quality compared to the standard memetic algorithm (without significant statistical difference according to the Friedman test). However, to achieve these results, Memetic/DP needs much more computing time than the standard memetic algorithm. In fact, for these 56 instances, the mean computing time consumed by Memetic/DP (903.36s) is about 14 times higher than the time of the standard memetic algorithm (63.04s). This experiment clearly demonstrates that the dynamic programming driven estimation criterion is extremely important to speed up the search process (by discarding a large number of non-promising candidate solutions), without sacrificing solution quality.

5.2. Impact of the Memetic Framework

We now analyze the impact of the memetic framework by disabling its crossover operator and removing its population mechanism. Precisely, we compare the memetic algorithm with an iterated local search (ILS) approach, which uses the same probabilistic constructive procedure for solution initialization, and the same neighborhood search procedure (including the dynamic programming driven estimation criterion) for local optimization. In order to run ILS under the same condition as our memetic algorithm, we reinforce ILS with a multi-start strategy: each time ILS reaches a local optimum, it is restarted from a newly generated initial solution and calls the neighborhood search procedure to reach another local optimum. This process is repeated until an optimal solution is reached (all the reachable profitable vertices are collected) or the best found solution cannot be improved after 500 consecutive restarts or the cutoff time (up to 12 minutes) is reached (being equivalent to the termination criteria of the memetic algorithm with M=500). We independently run ILS ten times to solve each of the 56 most challenging instances and compare the best and average results as well as the computing time with those of our memetic algorithm. The detailed results of this experiment are provided in Table 3 of the online supplement.

In terms of the best results, the memetic algorithm respectively improves, matches, misses 35, 13, 8 best results compared to ILS, with a mean improvement of 0.43% and a p-value=3.83×10^{-5}, indicating a statistically significant difference. Furthermore, the memetic algorithm respectively improves, matches, misses 45, 3, 8 average results compared to ILS, with a mean improvement of 0.35% and a p-value=3.73×10^{-7} indicating that the difference in terms of average quality is even larger. The mean time consumed by the memetic algorithm is 65.04s, which is a little more than the mean time required by ILS

(46.91s). This comparison provides thus evidences that the memetic framework is relevant to the performance of the whole algorithm.

6. Conclusion

The Steiner tree problem with revenues, budget and hop constraints (STPRBH) is a relevant model able to formulate a number of network design problems. The proposed memetic approach for the STPRBH relies on a probabilistic constructive procedure for initialization, a neighborhood search subroutine based on a dynamic programming procedure for local optimization, a backbone based crossover operator for generating offspring solutions and a quality-and-distance pool updating strategy for managing the solutions pool. Although all these components are indispensable for the proposed algorithm, we have shown the particular importance of the dynamic programming pre-examination technique for speeding up the neighborhood exploitation and the advantages of the population-based memetic framework compared to an iterated local search procedure for solving the STPRBH.

Experiments based on four groups of 384 STPRBH benchmarks from the literature demonstrate that the proposed algorithm is highly competitive, compared to the currently available methods. For the 328 cases with a known optimum, the memetic algorithm can attain the optimal result for 318 cases (96.95%) within a short computing time (from several seconds to several minutes). More importantly, for the 56 most challenging instances with unknown optimal solutions, our algorithm consistently finds improved solutions (new lower bounds) for 45 cases. Additionally, we have generated a new group of 30 benchmark instances and reported computational results which can serve as a basis for comparing new solution methods for the STPRBH. Finally, we hope that the main ideas of this research (i.e., pre-examination for neighborhood exploration, backbone based crossover, quality-and-distance pool updating criterion) could be useful for designing effective heuristics for other STP variants, including those encountered in real-life applications.

Acknowledgments

The work was supported by the following projects: RaDaPop and LigeRO (2009-2013, Pays de la Loire Region) and PGMO (2014-2015, Jacques Hadamard Mathematical Foundation). We are grateful to the referees and the editors for their insightful comments and suggestions. We are also grateful to Dr. A.M. Costa for kindly making the benchmark instances available to us and answering our questions.

References

[1] Andonov, R., V. Poirriez, S. Rajopadhye. 2000. Unbounded knapsack problem: dynamic programming revisited. *Eur. J. Oper. Res.* **123**(2) 168–181.

[2] Beasley, J. E. 1990. OR-Library: Distributing test problems by electronic mail. *J. Oper. Res. Soc.* **41**(11) 1069–1072.

[3] Benlic, U., J. K. Hao. 2011. A multilevel memetic approach for improving graph k-partitions. *IEEE Trans. Evolut. Comput.* **15**(5) 624–642.

[4] Botton, Q., B. Fortz, L. Gouveia, M. Poss. 2013. Benders decomposition for the hop-constrained survivable network design problem. *INFORMS J. Comput.* **25**(1) 13–26.

[5] Bui, T. N., X. H. Deng, M. Catherine. 2012. An improved ant-based algorithm for the degree-constrained minimum spanning tree problem. *IEEE Trans. Evolut. Comput.* **16**(2) 266–278.

[6] Carvalho, P. M. S., L. A. F. M. Ferreira, L. M. F. Barruncho. 2001. On spanning-tree recombination in evolutionary large-scale network problems-application to electrical distribution planning. *IEEE Trans. Evolut. Comput.* **5**(6) 623–630.

[7] Chakrabarty, D., N. R. Devanur, V. V. Vazirani. 2011. New geometry-inspired relaxations and algorithms for the metric Steiner tree problem. *Math. Program.* **230**(1): 1–32.

[8] Chou, H. H., G. Premkumar, C. H. Chu. 2001. Genetic algorithms for communications network design-an empirical study of the factors that influence performance. *IEEE Trans. Evolut. Comput.* **5**(3) 236–249.

[9] Cordone, R., G. Passeri. 2012. Solving the quadratic minimum spanning tree problem. *Appl. Math. Comput.* **218**(23) 11597–11612.

[10] Costa, A. M. 2006. Models and algorithms for two network design problems. PhD thesis, Ecole des Hautes Etudes Commerciales, Montreal, CA.

[11] Costa, A. M., J. F. Cordeau, G. Laporte. 2006. Steiner tree problems with profits. *INFOR* **44** 99–115.

[12] Costa, A. M., J. F. Cordeau, G. Laporte. 2008. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. *Eur. J. Oper. Res.* **190**(1) 68–78.

[13] Costa, A. M., J. F. Cordeau, G. Laporte. 2009. Models and branch-and-cut algorithms for the Steiner tree problem with revenues, budget and hop constraints. *Networks* **53**(2) 141–159.

[14] Ferreira, C. S., L. S. Ochi, V. Parada, E. Uchoa. 2012. A GRASP-based approach to the generalized minimum spanning tree problem. *Expert Syst. Appl.* **39** 3526–3536.

[15] Fu, Z. H., J. K. Hao. 2014. Breakout local search for the Steiner tree problem with revenue, budget and hop constraints. *Eur. J. Oper. Res.* **232**(1) 209–220.

[16] Garey, M. R., R. L. Graham, D. S. Johnson. 1977. The complexity of computing Steiner minimal trees. *SIAM J. Appl. Math.* **32**(4) 835–859.

[17] Golden, B., S. Raghavan, D. Stanojević. 2005. Heuristic search for the generalized minimum spanning tree problem. *INFORMS J. Comput.* **17**(3) 290–304.

[18] Hao, J. K. 2012. Memetic algorithms in discrete optimization. F. Neri, C. Cotta, P. Moscato, eds. *Handbook of Memetic Algorithms.* Springer.

[19] Karp, R. M. 1972. Reducibility among combinatorial problems. R. E. Miller, J. W. Thatcher, eds. *Complexity of Computer Computations.* New York.

[20] Lawler, E. 1976. Combinatorial Optimization: Networks and Matroids. New York.

[21] Layeb, S. B., I. Hajri, M. Haouari. 2013. Solving the Steiner tree problem with revenues, budget and hop constraints to optimality. *Proc. 5th Internat. Conf. Modeling, Simulation and Applied Optimization*, IEEE Publisher, Hammamet, Tunisia, 1–4.

[22] Lü, Z. P., F. Glover, J. K. Hao. 2010. A hybrid metaheuristic approach to solving the UBQP problem. *Eur. J. Oper. Res.* **207**(3) 1254–1262.

[23] Moscato, P., C. Cotta. 2003. A gentle introduction to memetic algorithms. F. Glover, G. Kochenberger, eds. *Handbook of Metaheuristics.* Kluwer Academic Publishers, The Nederlands.

[24] Öncan, T., A. P. Punnen. 2010. The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm. *Comput. Oper. Res.* **37** 1762–1773.

[25] Porumbel, D. C., J. K. Hao, P. Kuntz. 2010. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Comput. Oper. Res.* **37**(10) 1822–1832.

[26] Potvin, J. Y. 2009. State-of-the art review - evolutionary algorithms for vehicle routing. *INFORMS J. Comput.* **21**(4) 518–548.

[27] Ribeiro, C. C., E. Uchoa, R. F. Werneck. 2002. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS J. Comput.* **14**(3) 228–246.

[28] Rothlauf, F. 2009. An encoding in metaheuristics for the minimum communication spanning tree problem. *INFORMS J. Comput.* **21**(4) 575–584.

[29] Santos, M., L. M. A. Drummond, E. Uchoa. 2010. A distributed dual ascent algorithm for the hop-constrained Steiner sree problem. *Oper. Res. Lett.* **38**(1) 57–62.

[30] Sinnl, M. 2011. Branch-and-Price for the Steiner tree problem with revenues, budget and hop constraints. Master Thesis, Faculty of Computer Science, Vienna University of Technology, Austria.

[31] Voβ, S. 1999. The Steiner tree problem with hop constraints. *Ann. Oper. Res.* **86** 321–345.

[32] Voβ, S. 2006. Steiner tree problems in telecomunications. P. Pardalos, M. G. C. Resende, eds. Handbooks of Optimization in Telecommunications. Springer, New York.