Breakout Local Search for the Steiner Tree Problem with Revenue, Budget and Hop Constraints

Zhang-Hua Fu and Jin-Kao Hao *

LERIA, Université d'Angers
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

Accepted to European Journal of Operational Research, June 2013.
DOI: http://dx.doi.org/10.1016/j.ejor.2013.06.048

Abstract

The Steiner tree problem (STP) is one of the most popular combinatorial optimization problems with various practical applications. In this paper, we propose a Breakout Local Search (BLS) algorithm for an important generalization of the STP: the Steiner tree problem with revenue, budget and hop constraints (STPRBH), which consists of determining a subtree of a given undirected graph which maximizes the collected revenues, subject to both budget and hop constraints. Starting from a probabilistically constructed initial solution, BLS uses a Neighborhood Search (NS) procedure based on several specifically designed move operators for local optimization, and employs an adaptive diversification strategy to escape from local optima. The diversification mechanism is implemented by adaptive perturbations, guided by dedicated information of discovered high-quality solutions. Computational results based on 240 benchmarks show that BLS produces competitive results with respect to several previous approaches. For the 56 most challenging instances with unknown optimal results, BLS succeeds in improving 49 and matching one best known results within reasonable time. For the 184 instances which have been solved to optimality, BLS can also match 167 optimal results.

Keywords: Steiner tree problems, network design, constrained combinatorial optimization, heuristic search, adaptive perturbation.

* Corresponding author.
Email addresses: fu@info.univ-angers.fr (Zhang-Hua Fu), hao@info.univ-angers.fr (Jin-Kao Hao).

1 Introduction

Many problems in network designing, e.g., electricity, telecommunication, heating, transportation, should determine a least cost tree spanning all or some of the vertices of a given graph (Avella et al., 2005; Voβ, 2006). These problems usually can be modeled as the Steiner tree problem (STP) or the minimum spanning tree problem (MSTP), which are generally formulated as follows: given a graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ which is partitioned into two sets: a set of terminal vertices and a set of Steiner vertices, and edge set $E = \{(i, j) : i, j \in V, i \neq j\}$ where each edge $(i, j) \in E$ has an associated cost $c_{ij} \geq 0$. In some cases, a specified vertex is chosen as the root vertex. The STP consists of determining a subtree spanning all terminal vertices (including the root vertex) and possibly some Steiner vertices, so as to minimize the total cost of the obtained tree. As a special variant of the STP, for the MSTP, all vertices are terminal which should be included in any feasible solution. Unlike the MSTP that can be solved to optimality within polynomial time (Prim, 1957), the STP has proven to be NP-hard (Garey et al., 1977).

In this paper, we study an important variant of the STP: the Steiner tree problem with revenue, budget and hop constraints (denoted by STPRBH, as formulated in Costa et al., 2009). In this problem, in addition to the costs $c_{ij} \geq 0$ associated with each edge $(i, j) \in E$, there is also a revenue $r_i \geq 0$ associated with each vertex $i \in V$. The problem consists of determining a rooted (without loss of generality, vertex 1 is fixed as the root) subtree of graph G, so as to maximize the collected revenues, while guaranteeing that the total cost of the solution does not exceed a given budget B (budget constraint), and the number of edges from the root to any vertex in the solution subtree does not exceed an upper bound equal to h (hop constraint). As a generalization of both the STPP (STP with profits, see Johnson et al., 2000; Costa et al., 2006; Haouari et al., 2013) and the STPH (STP with hop constraints, see Voβ, 1999; Akgün, 2011), the STPRBH is theoretically important and can be used to model many real-life problems, e.g., local access and telecommunication networks, heating or water supply systems, transportation planning, etc, in which the collected revenues should be maximized, while the available budget is limited and the reliability of the system should be guaranteed. For the STPRBH, researchers have developed various solution approaches. Respectively, Costa et al. (2008) proposed several fast heuristics, including a greedy algorithm, a destroy-and-repair algorithm and a tabu search (TS) algorithm. Computational results for instances with up to 500 vertices and 12500 edges were reported. In addition to the heuristics, several exact algorithms have also been proposed, including branch-and-cut (Costa et al., 2009), branch-and-price (Sinnl, 2011). Note that all the existing exact algorithms can only solve instances with up to 500 vertices and 625 edges to optimality, for larger instances, no result has been reported by any exact algorithm.

In this paper, we are interested in the STPRBH and propose a heuristic algorithm based on the Breakout Local Search (BLS) for this problem. BLS follows the general Iterated Local Search scheme (Lourenco et al., 2003) and alternates between a neighborhood search phase and a perturbation phase. BLS has recently shown its effectiveness for solving several combinatorial optimization problems, such as sum coloring (Benlic and Hao, 2012), maximum clique (Benlic and Hao, 2013a), quadratic assignment (Benlic and Hao, 2013b), and max-cut(Benlic and Hao, 2013c). For the STPRBH, the proposed BLS algorithm integrates a probabilistic constructive procedure to generate its initial solution, a Neighborhood Search (NS) procedure based on three specifically designed move operators to discover local optima, and an adaptive perturbation strategy to continually move from one local optimum to another one, by varying its perturbations depending on the search status. As a supplementary technique, a number of high-quality solutions are stored in a solutions pool, in order to provide useful information for local optimization and perturbations. Computational results based on a set of 240 STPRBH instances, including 56 the most challenging instances with unknown optimal solutions, demonstrate the effectiveness of the proposed BLS algorithm. In particular, it succeeds in improving 49 and matching one best known results out of these 56 unsolved instances.

The rest of this paper is organized as follows: After giving some preliminary definitions in Section 2, Section 3 describes the details of the proposed BLS approach. Computational results are provided in Section 4, and Section 5 concludes this paper.

2 Preliminary definitions

In this section, we provide some preliminary definitions which are useful for a precise description of the proposed 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.

Definition 2. Given a BHS-tree T, a *feasible candidate path* with respect to T is a path originating at a vertex $i \in v(T)$ ($v(T)$ denotes the set containing all the vertices belonging to solution T) and connecting to an uncollected profitable vertex j ($j \notin v(T), r_j > 0$), such that even after inserting this path to 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 can-

didate path exists. Otherwise, the BHS-tree is an *unsaturated (or partial) BHS-tree*. Contrary to a saturated BHS-tree, an unsaturated (or partial) BHS-tree can be further extended by adding some feasible candidate path without violating the budget and hop constraints.

Definition 4. The *constrained search space* Ω is composed of all possible BHS-trees (including saturated ones or unsaturated ones). The *saturated constrained search space* $\overline{\Omega}$ is composed of all possible saturated BHS-trees which is clearly a subspace of Ω.

As detailed below, our BLS algorithm restricts its search within the saturated constrained search space $\overline{\Omega}$. By doing so, the search process focuses always on the reduced zones composed of the most promising candidate solutions.

3 The proposed BLS algorithm

In this paper, we present for the first time a Breakout Local Search (BLS) approach for solving the STPRBH, just as outlined in Algorithm 1, whose key components are presented in the following subsections.

Our BLS algorithm operates within the saturated constrained search space $\overline{\Omega}$ (Section 2). The main idea of the approach for the STPRBH can be described as follows: starting from a saturated BHS-tree probabilistically constructed by the dedicated probabilistic constructive procedure (see Algorithm 1, line 3 and Section 3.2), BLS applies a Neighborhood Search (NS) procedure to reach a local optimum at first (line 4, see Section 3.3). After local optimization, BLS then attempts to continually move from one local optimum to another by employing varying perturbations, depending on the state of the search. For this purpose, an adaptive perturbation mechanism is developed, which is guided by some dedicated information of a number of recorded high quality solutions stored in the HSP (line 2 and line 10, see Section 3.3.2 and 3.4). Each time the incumbent solution is perturbed, the NS procedure is called again to improve it to a new local optimum (line 11). If the NS procedure reaches a local optimum not far enough from the original one, BLS then perturbs it more strongly, otherwise, BLS switches to weaker perturbations subsequently (lines 16-21). This process is repeated until (1) the upper bound of the collected revenues in Eq. (2) (see Section 3.2.2) is reached (meaning that an optimal solution is obtained), or (2) the best found solution cannot be further improved after visiting M new local optima (M is a parameter), or (3) the allowed computation time is consumed.

The performance of the BLS algorithm relies on several key factors. First, the initialization procedure should be able to generate different solutions of

reasonable quality, which serve as the restarting points of independent runs of BLS. Second, the neighborhood structure is also a key component because different neighborhoods lead to different search trajectories, thus solutions of different qualities. Third, we should control the jump magnitude, denoted by L, which determines the perturbation intensity applied to the current solution. In our case, this corresponds to decide how many paths to delete when perturbing the incumbent solution. Indeed, if L is too small, the search usually returns to the original local optimum, leading to search stagnation. Otherwise, if L is too large, the perturbation is reduced to random restarting. Finally, it is important to consider the perturbation type, e.g., directed perturbation or random perturbation. Unlike conventional pure random perturbations, we additionally employ a directed perturbation operator with the aid of selected high quality solutions, which provides useful information to guide the search towards good solutions. The components of the proposed BLS algorithm for the STPRBH are described below.

Algorithm 1 Breakout Local Search $BLS(G, B, h)$ for the STPRBH

Require: Graph $G(V, E)$, budget limit B, hops limit h, jump magnitude $L \in [L_{min}, L_{max}]$, high-quality (elite) solution pool HSP
Ensure: The best solution found meeting both the budget and hop constraints

/* Initialization phase */
$HSP \leftarrow InitHSP()$ /* Initialize HSP, see Sect. 3.3.2 */
$T \leftarrow InitSolution(G, B, h)$ /* Construct an initial solution, see Sect. 3.2 */
$T \leftarrow NS(T)$ /* Optimize T by neighborhood search, see Sect. 3.3 */
$T^{best} \leftarrow T$
$L \leftarrow L_{min}$
/* Main search procedure which is iterated until the stop condition is met */
while The stop condition is not met **do**
  /* Perturb T with L and HSP (Sect. 3.4) and then improve it (Sect. 3.3) */
  $T' \leftarrow Perturb(T, HSP, L)$
  $T^* \leftarrow NS(T')$
  /* Update the best solution T^{best} found so far if needed */
  if T^* is better than T^{best} (see Sect. 3.3.1) **then**
    $T^{best} \leftarrow T^*$
  end if
  /* Determine the jump magnitude L adaptively, detailed in Sect. 3.4 */
  if T^* is too close to T (defined in Sect. 3.4) **then**
    $L \leftarrow Min(L + 1, L_{max})$
  else
    $L \leftarrow Max(L - 1, L_{min})$
  end if
  /* Update T, which serves as the starting point of a new round of search */
  $T \leftarrow T^*$
end while
return T^{best}

3.1 Solution presentation

To represent the candidate solutions of the problem in a convenient way, we adopt a compact representation using an one-dimensional vector $T = \{t_i, i \in V\}$ which is explained as follows. Precisely, according to the constraints of the STPRBH, each feasible solution is a rooted tree with fixed root vertex 1. Therefore, for each vertex i belonging to a feasible solution T (except the root vertex), we can identify and record its parent vertex t_i. Specifically, the elements of $T = \{t_i, i \in V\}$ are defined such that t_i = the parent vertex of vertex i if $i \in v(T)\backslash 1$; Otherwise t_i = Null.

Consequently, each feasible solution is uniquely identified by a vector $T = \{t_i, i \in V\}$. Inversely, given a vector $T = \{t_i, i \in V\}$ corresponding to a feasible solution, it is easy to reconstruct the corresponding solution.

3.2 Probabilistic constructive procedure for initialization

Like any meta-heuristic based algorithm, BLS requires an initial solution to start its search. Moreover, given its stochastic nature, multiple runs of BLS from different initial solutions are typically applied to find the best possible solutions for a problem instance. To generate an initial solution, we use the criteria developed by Costa et al. (2008) for identifying, evaluating and selecting feasible candidate paths for insertion and devise a probabilistic constructive procedure in order to be able to obtain different initial solutions for multiple runs of the procedure. Starting from an empty solution containing only the root vertex, the constructive procedure identifies all the feasible candidate paths with respect to the incumbent solution at first and evaluates their priorities subsequently. Then, it probabilistically selects a candidate path to insert to the incumbent solution, according to its priority. This process is repeated until no feasible candidate path exists, meaning that a saturated BHS-tree satisfying both the budget and hop constraints is obtained, which would serve as the starting point of our BLS algorithm.

3.2.1 Hop constrained shortest path problem

Before presenting the criteria for identifying, evaluating and selecting a candidate path, we should solve the hop constrained shortest path problem at first, i.e., the problem of determining a shortest path between two vertices containing at most h edges. This problem can be solved efficiently by dynamic programming (Lawler, 1976): let $L(i, j, l)$ represent the cost of the shortest

path between vertex i and vertex j, containing at most l edges, then:

$$\begin{aligned} &L(i,i,0) = 0, i \in V, \\ &L(i,j,0) = \infty, i,j \in V, j \neq i, \\ &L(i,j,l) = \min\{L(i,j,l-1), \min_{k|(k,j)\in E}\{L(i,k,l-1) + c_{kj}\}, i,j \in V, l \geq 1\}. \end{aligned} \tag{1}$$

Note that in Costa et al. (2008), each time a new path is inserted, the costs of all the edges belonging to the incumbent solution are reset to 0 and the hop constrained shortest path algorithm is run again to re-calculate all the possible $L(1,j,l), l \leq h$. Contrary to this, with our BLS algorithm, we apply a preprocessing step to calculate and store all the possible $L(i,j,l), i,j \in V, r_j > 0, 1 \leq l \leq h$ which are then used directly during the search process, instead of re-calculating them repeatedly. This technique allows the algorithm to save computation time. In the following subsections, whenever reporting the computation time of BLS, this preprocessing time is always included.

3.2.2 Upper bound of the collected revenues

After calculating all the possible $L(1,j,h)$, we can recognize all the vertices reachable within h hops, i.e., $L(1,j,h) < \infty$. Then, we get an upper bound R^{ub} of the collected revenues as follows:

$$R^{ub} = \sum_{L(1,j,h)<\infty} r_j. \tag{2}$$

Clearly, if the collected revenues reaches the upper bound R^{ub}, then the incumbent solution corresponds to an optimal solution and the search process stops. This rule is indeed used as one of the termination criteria of the proposed BLS algorithm.

3.2.3 Identifying feasible candidate paths

At each step of constructing a saturated BHS-tree, we use the following criterion to dynamically identify all the feasible candidate paths. Specifically, for each profitable vertex j uncollected by the incumbent partial BHS-tree T ($r_j > 0$ and $j \notin v(T)$), let $L(i,j,h-h_i)$ denote the cost of the hop-constrained shortest path between vertex j and a vertex $i \in v(T)$, containing at most $h-h_i$ edges, h_i being the number of edges between vertex i and the root vertex. We use $\min\{L(i,j,h-h_i), i \in v(T)\}$ to denote the cost of the hop-constrained shortest path between vertex j and the partial BHS-tree T. Consequently, if $\min\{L(i,j,h-h_i), i \in v(T)\} \leq B - \sum_{(m,n)\in e(T)} c_{mn}$ ($e(T)$ is the set containing

all the edges of T, and $\sum_{(m,n)\in e(T)} c_{mn}$ represents the total cost consumed by T), it corresponds to a feasible candidate path connecting vertex j. Otherwise, no feasible candidate path connecting vertex j exists.

3.2.4 Evaluating feasible paths

At each step of initial solution construction, there may be more than one feasible candidate paths available. In Costa et al. (2008), the ratio $\frac{r_j^3}{L(1,j,h)}$ which considers both the objective and the constraint is used to evaluate the priority of each candidate path connecting an uncollected profitable vertex $j \notin v(T)$. In this work, we use instead the ratio $\frac{r_j^3}{min\{L(i,j,h-h_i),i\in v(T)\}}$ for this evaluation. Given two paths, the path with a higher ratio is considered to be of more priority than the other one. Additionally, due to the reasons described in Section 3.2.1, this criterion is essentially equivalent to the one developed in Costa et al., [10].

3.2.5 Selecting and inserting a candidate path

After identifying all feasible candidate paths, we should decide which path to insert to the incumbent solution. In order to be able to start the search from different starting points, we introduce the following randomized selection rule:

(1) If no feasible candidate path is available, stop the constructive procedure and return the incumbent solution (which is already a saturated BHS-tree) as the initial solution;
(2) If there is only one feasible candidate path available, select and insert it into the incumbent solution;
(3) If there are m ($m > 1$) feasible candidate paths available, select the path with the ith ($2 \leq i \leq m$) highest priority with probability $(1-\theta)^{i-1}\theta, \theta \in (0, 1]$, and select the path with the highest priority with probability $1 - \sum_{i=2}^{m}(1-\theta)^{i-1}\theta = \theta + (1-\theta)^m$, so as to guarantee that the accumulated probability is equal to 1.

The above probabilistic criterion has two interesting features. First, each feasible candidate path has the opportunity to be selected and inserted. Consequently the probabilistic constructive procedure is theoretically able to cover the whole solution space. Second, the probability that a feasible candidate path is selected is proportional to its priority. As such, the constructed initial solutions tend to be of good quality.

In addition, because of the hop-constraints, cycles may occur after inserting a new path. In this case, we destroy cycles by inspecting vertices with two incoming edges and eliminating the first inserted edge, as suggested in Costa

et al. (2008).

Finally, in the above-described probabilistic constructive procedure, if we let $\theta = 1.0$, it is reduced to the greedy algorithm (Costa et al., 2008). However, after a series of preliminary experiments (detailed in Section 4.2, Fig. 2), we find that the generalizations with small values of θ (i.e., θ <0.5) generally perform better than large ones (i.e., θ >0.5), especially than the choice of the greedy algorithm (i.e., θ=1.0), which unexceptionally selects the candidate path with the highest priority.

3.3 Neighborhood Search (NS) for local optimization

¿From an initial solution constructed above, our BLS algorithm improves it to a local optimum by using a specifically designed Neighborhood Search (NS) procedure, which iteratively replaces the incumbent solution with the best improving solution of its neighborhood, until such a solution does not exist. The proposed NS procedure is detailed in the following subsections.

3.3.1 Evaluation of solutions

To identify the best neighboring solution of the incumbent solution T within the neighborhood $N(T)$ (see next subsection), BLS uses the objective value (i.e., the collected revenues) as the main evaluation criterion. Thus given two solutions, the one collecting a higher revenue is better than the one collecting a lower revenue. If both solutions collect the same amount of revenue, the solution with a lower cost is naturally considered to be the better one. Given this evaluation rule, BLS identifies at each iteration the best improving solution among all the candidate neighboring solutions in $N(T)$ and moves to this selected solution.

3.3.2 Neighborhood structure

The neighborhood structure is the key component of any neighborhood-based search method. We design three dedicated move operators for the SPTRBH denoted by $Move\_k(i_1, \ldots, i_k), 1 \leq k \leq 3$ for generating neighboring solutions. As a preliminary, we first introduce two basic operators: $Delete(i)$, $Insert()$ as follows.

$Delete(i)$: delete the path connecting leaf vertex i from the incumbent solution. Note that for path deletion, we just delete the edges between vertex i and the first met branch vertex or profitable vertex, while maintaining the left edges

between vertex i and the root vertex (e.g., Fig. 1, $a \rightarrow b \rightarrow c$, see below for more comments).

$Insert()$: iteratively insert one path (excluding the recently deleted ones) connecting some profitable vertex to the incumbent solution, according to the criteria described in Section 3.2, until the incumbent solution becomes a saturated BHS-tree, i.e., no feasible candidate path can be further inserted. Note that zero or several paths may be inserted by $Insert()$ (as shown in Fig. 1, $c \rightarrow d$).

With these two basic operators, three move operators for generating neighboring solutions, i.e., $Move\_k(i_1, \ldots, i_k)$, $1 \leq k \leq 3$, are implemented as follows.

$$\begin{aligned} &\text{Move\_1}(i_1) = \text{Delete}(i_1) + \text{Insert}(), \\ &\text{Move\_2}(i_1, i_2) = \text{Delete}(i_1) + \text{Delete}(i_2) + \text{Insert}(), \\ &\text{Move\_3}(i_1, i_2, i_3) = \text{Delete}(i_1) + \text{Delete}(i_2) + \text{Delete}(i_3) + \text{Insert}(). \end{aligned} \tag{3}$$

where $i_1, i_2, i_3 \in lv(T)$, $lv(T)$ is the set containing all the leaf vertices of the incumbent solution T.

With these move operators, the generated neighboring solutions are denoted by $T \bigoplus Move\_k(i_1, \ldots, i_k)$. Note that different k corresponds to different neighboring solutions.

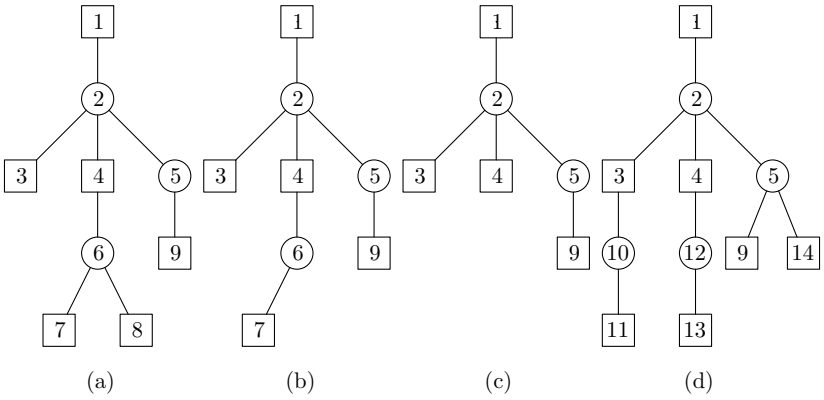

Fig. 1. Generate a neighboring solution by operator $Move\_2(8, 7)$

For example, Fig. 1 illustrates the process for generating a neighboring solution by $Move\_2(8, 7)$, where the profitable vertices with $r_i > 0$ are drawn in box (i.e., vertices 1, 3, 4, 7, 8, 9, 11, 13, 14), and the others are drawn in circle (i.e., vertices 2, 5, 6, 10, 12). As shown in Fig. 1, from the original solution a, the path connecting leaf vertex 8 is deleted at first to get solution b. Then the path connecting leaf vertex 7 is deleted to get solution c. Finally, three new paths connecting profitable vertices 11, 13, 14 are inserted into c to obtain a neighboring solution d of the original solution a, i.e., $d \leftarrow a \bigoplus Move\_2(8, 7)$.

Given that vertex 6 is a branch vertex that has two branches, when deleting the path connecting leaf vertex 8 from a, we only delete edge (6,8) from a, instead of deleting all the edges between vertex 8 and the root. Similarly, because vertex 4 is a profitable vertex with $r_4 > 0$, when deleting the path connecting leaf vertex 7, we only delete edges (4,6) and (6,7) from b.

Clearly, if $|lv(T)| = m$, there are $C_m^k = \frac{m!}{k!(m-k)!}$ possible neighboring solutions with move operator $Move\_k(i_1, \ldots, i_k)$. Therefore, if we consider all the possible neighboring solutions, the neighborhood $N(T)$ contains $O(m^3)$ neighboring solutions. Although larger neighborhoods generally lead to better local optimal solution, more computation time is also needed. For the trade-off between solution quality and efficiency, in this work, we try to reduce the neighborhood $N(T)$ to contain $O(m)$ solutions selected from all the $O(m^3)$ possible candidate ones by the following steps.

(1) Let $N(T, k), 1 \leq k \leq 3$ denote the sub-neighborhood associated with operator $Move\_k(i_1, \ldots, i_k)$. Then, initialize $N(T, 1)$ to contain all the m possible neighboring solutions generated by $Move\_1(i_1)$, and initialize $N(T, 2), N(T, 3)$ to be $\emptyset$.

(2) As a preliminary, try to construct and fill a high-quality (or elite) solution pool HSP as follows. For each individual, call the probabilistic constructive procedure described in Section 3.2 to construct an initial solution, and then iteratively replace the incumbent solution with the best improving neighboring solution of its smallest sub-neighborhood $N(T, 1)$, until no improving solution exists in $N(T, 1)$. Independently repeat this process Q times to obtain Q local optima, i.e., $T^1, \ldots, T^Q$, then calculate their average collected revenues and retain the ones collecting no less revenues than the average value. These selected solutions are considered as high-quality solutions and stored into the HSP.

(3) For each vertex i, calculate the probability p_i that vertex i belongs to the solutions stored in the HSP, i.e.,

$$p_i = \frac{\sum_{T^a \in HSP} y_i^a}{|HSP|}. \tag{4}$$

where y_i^a indicates whether vertex i belongs to solutions $T^a \in HSP$ or not. If $i \in v(T^a)$, $y_i^a = 1$, otherwise, $y_i^a = 0$.

(4) Apply a probabilistic deletion operator, denoted by $ProbDel(k)$, to probabilistically delete k paths connecting k leaf vertices from the incumbent solution T, with the aid of the HSP (as detailed in Algorithm 2). Note that in Algorithm 2, each time one path connecting a leaf vertex is deleted, some branch vertex may become a new leaf vertex. The path connecting this new leaf vertex should also be considered when deciding the next path to delete.

(5) For each k, $2 \leq k \leq 3$, generate $|lv(T)| = m$ neighboring solutions and add them into sub-neighborhood $N(T, k)$ as follows: for each leaf vertex

$i \in lv(T)$, delete the path connecting vertex i at first, and then execute the probabilistic deletion operator $ProbDel(k-1)$ to probabilistically delete $k-1$ paths. After that, execute the basic operator $Insert()$ to insert as many feasible candidate paths (excluding the recently deleted ones) as possible into the incumbent solution, to obtain a saturated BHS-tree, i.e.,

$$N(T,k) = \{T + \text{Delete}(i) + \text{ProbDel}(k-1) + \text{Insert}(), i \in lv(T)\}, 2 \leq k \leq 3. \quad (5)$$

(6) Let $N(T)$ be the union set of all the above three sub-neighborhoods, i.e.,

$$N(T) \leftarrow N(T,1) \cup N(T,2) \cup N(T,3). \quad (6)$$

Algorithm 2 $ProbDel(k)$ for probabilistically deleting k paths from the incumbent solution T

Require: Solution T, elite solution pool HSP, number of paths to delete k
Ensure: Solution after deleting k paths

```
for each vertex i do
    Calculate the probability p_i that vertex i belongs to the solutions stored in the HSP, according to Eq. (4)
end for
l ← 0
while l < k and leaf vertex i ∈ lv(T), p_i < 1 exists do
    Deleted ← false
    while Deleted = false do
        Randomly select a path connecting some leaf vertex i ∈ lv(T), p_i < 1
        /* Delete the path connecting leaf vertex i with probability 1 − p_i */
        if GetRandomNum(100) > 100 × p_i then
            T ← Delete(i)
            Deleted ← true
        end if
    end while
    l ← l + 1
end while
return T
```

The neighborhood $N(T)$ defined in Eq. (6) will be used as the final neighborhood of the NS procedure. The following properties of this neighborhood are worth mentioning. First, each sub-neighborhood $N(T,k), 1 \leq k \leq 3$ contains m neighboring solutions, hence the final neighborhood $N(T)$ contains $3m$ neighboring solutions, instead of the original $O(m^3)$ possible ones. Second, $N(T,k_1) \bigcap N(T,k_2) = \emptyset, k_1 \neq k_2$, it means that any neighboring solution belongs to only one sub-neighborhood. Third, for each leaf vertex $i \in lv(T)$, at least one neighboring solution corresponding to deleting the path connecting vertex i belongs to each sub-neighborhood $N(T,k), 1 \leq k \leq 3$. This feature could be helpful to reinforce the diversity of the solutions of the neighborhood. Fourth, the higher the probability p_i of leaf vertex i, the larger its probability

to be retained while generating neighboring solutions. We utilize this mechanism to select $O(m)$ promising neighboring solutions among all the $O(m^3)$ candidates. Finally, every neighboring solution is unexceptionally a saturated BHS-tree, thus the search is restricted within the saturated constrained search space $\overline{\Omega}$.

In addition, to verify the impact of the neighborhood structure, we tested two other different neighborhoods, i.e., $N(T,1)$ and $N(T,1) \cup N(T,2)$ respectively, and compared their performances with the neighborhood defined by Eq. (6). Experiments showed that BLS with Eq. (6) yielded statistically much better results (in terms of solution quality) than these two compared variants, though some more (remaining reasonable) computational time is needed. To ensure that BLS finds high quality solutions, we adopt Eq. (6) as the final neighborhood.

Based on the above described neighborhood and the way to identify the best improving neighboring solution, the NS procedure starts from a given initial saturated BHS-tree T, and iteratively replaces T with the best improving neighboring solution of its neighborhood $N(T)$. This process continues until no such solution exists in the neighborhood. At this point, a local optimum is reached. To continue its search, our BLS procedure applies a dedicated perturbation mechanism for escaping from the incumbent local optimum, according to the procedure detailed below.

3.4 Adaptive perturbation mechanism

The purpose of the perturbation mechanism is to allow BLS to escape from the current local optimum in order to discover other local optima of better quality. For this, we employ as follows an adaptive perturbation mechanism which varies the perturbation intensity, depending on the search status.

Precisely, each time the search reaches a local optimum T, we perturb it to obtain a new solution. The perturbation consists in deleting L (initialized to L_{min}) paths from T and inserting subsequently as many feasible candidate paths as possible into the incumbent solution. From this perturbed solution, we call the NS procedure to reach another local optimal solution T^*. If the new solution T^* is too close to T (see below for the exact definition), we increase the jump magnitude L by 1, unless $L = L_{max}$; otherwise, we decrease L by 1, unless $L = L_{min}$. This process is repeated, until the stop condition is met.

To assess if two solutions T^a and T^b are close or not, we calculate their Hamming distance $D(T^a, T^b)$, based on which we further define the average distance between the solutions of the high-quality solutions pool HSP (Section

3.3.2, step 2), denoted by $AvgDis(HSP)$, as follows:

$$AvgDis(HSP) = \frac{\sum_{T^a,T^b \in HSP, a<b} D(T^a, T^b)}{\frac{1}{2} \times |HSP| \times (|HSP| - 1)}. \quad (7)$$

Based on this, T^* is considered to be too close to T if $D(T, T^*) < \alpha \times AvgDis(HSP)$. Parameter α would be further discussed in Section 4.2.

In addition to the jump magnitude L, BLS also considers the type of perturbations. In this work, we develop a directed perturbation operator, with the aid of the HSP. The basic idea is that the vertices which frequently occur in high-quality solutions are more likely to belong to the global optimal solution. Therefore, when we perturb the incumbent solution, it would be wise to retain these specific vertices with a larger probability, and retain the others with a smaller probability. Specifically, let T be the current local optimal solution, we perturb T by the following three steps, with a given jump magnitude L.

- For each vertex $i \in v(T)$, calculate the probability p_i that vertex i belongs to the solutions stored in the HSP, according to Eq. (4).
- Call Algorithm 2 to probabilistically delete L paths, unless no leaf vertex with $p_i < 1$ exists.
- Execute the basic operator $Insert()$ (Section 3.3.2) to insert as many feasible candidate paths (excluding the paths deleted in above step) as possible into the incumbent solution, until no path can be further inserted.

This perturbation procedure has the following features. First, the incumbent solution is perturbed in an adaptive way, controlled by the jump magnitude L, i.e., the larger the magnitude L, the stronger the perturbation. Second, the perturbation attempts to reconstruct a new incumbent solution in a biased mode, guided by some dedicated information from the high-quality solutions in the HSP. Finally, the new solution never violates the budget and hop constraints and is always a saturated BHS-tree, hence no repair is required and the search always operates within the saturated constrained search space $\overline{\Omega}$.

3.5 Discussions

As shown previously, the probabilistic constructive procedure used by our BLS approach is inspired by the greedy heuristic from Costa et al. (2008) and provides a natural generalization. In addition, BLS distinguishes itself from the existing heuristics by several significant features. First, unlike previously heuristics, BLS follows the iterated local search framework which includes an adaptive breakout perturbation strategy to escape from local optima. Second, with its dedicated neighborhood structure, we ensure that BLS operates

within a largely reduced and more focused search space, i.e., the saturated constrained search space $\overline{\Omega}$. This is in shape contrast with respect to the existing heuristics which explore usually much larger spaces including unfeasible or unsaturated solutions. As we show in the next section, the proposed BLS algorithm equipped with these particular features is able to reach very competitive results with respect to the existing methods in terms of both solution quality and computational time.

4 Computation results and comparisons

We evaluate our BLS algorithm on a large number of benchmark instances of the literature and compare our results with the best known published results. Given that it is difficult to make a really fair comparison of the computational efforts based on different platforms, we consider the collected revenues (objective function values) as our main evaluation criterion, and include the runtime just for indicative purposes. For information, the BLS algorithm is implemented in C++ [1] and executed on an Intel Xeon E5440 2.83GHz processor (with a peak value of 25.5, according to the Standard Performance Evaluation Corporation via www.spec.org) 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 (Costa, 2006; Costa et al., 2008; 2009), and a computer with an Intel Xeon E5540 2.53GHz processor (with a peak value of 29.4) and 3GB RAM was used in Sinnl (2011). In order to make the comparisons as fair as possible, whenever reporting the results corresponding to the reference algorithms, their CPU times are harmonized with respect to our processor according to the peak values evaluated by SPEC. Respectively, the CPU times reported in (Costa, 2006; Costa et al., 2008; 2009) are multiplied by 0.73 ($\frac{18.7}{25.5}$), while the CPU times reported in Sinnl (2011) are multiplied by 1.15 ($\frac{29.4}{25.5}$).

4.1 Benchmark instances

We use the 40 challenging benchmark graphs from Costa (2006) and Costa et al. (2008), which are adapted Steiner graphs from the series C of the OR-Library [2] (Beasley, 1990). These instances are also used in Costa et al. (2009)

[1] The source code of the BLS algorithm is available at `http://www.info.univ-angers.fr/pub/hao/stprbh.html`

[2] These instances and the solution certificates of our BLS algorithm are available at `http://www.info.univ-angers.fr/pub/hao/stprbh.html`. The initial Steiner graphs are available at `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html`

and Sinnl (2011). Note that for each graph, 6 different scenarios are considered, leading to 40×6=240 cases. These 240 cases are further classified into 3 groups as follows.

Group 1: This group contains 60 cases from the first 10 graphs (steinc1_10, ..., steinc5_10, steinc1_100, ..., steinc5_100), all with 500 vertices and 625 edges. For each graph, 6 different cases associated with different $B = \frac{\sum_{(i,j)\in E} c_{ij}}{b}$ and h, i.e., $b = 10, 30$ and $h = 5, 15, 25$, are created. These 60 cases have all been solved to optimality by exact algorithms, with a time limit of 5256s in Costa et al. (2009) or 11500s in Sinnl (2011) (after harmonizing by SPEC).

Group 2: This group contains 72 cases from 12 different graphs (steinc8_10, steinc8_100, steinc9_10, steinc9_100, steinc10_10, steinc10_100, steinc13_10, steinc13_100, steinc14_10, steinc14_100, steinc15_100, steinc15_100), with 500 vertices and up to 2500 edges. These cases are rather large and cannot be solved by the above exact algorithms. However, for 16 special cases, the allowed budget is abundant enough that it is not difficult for heuristics to reach the upper bound R^{ub} of the collected revenues (see Eq. (2)). Meanwhile, for the remaining 56 cases, the optimal results still remain unknown. These 56 cases can thus be considered as the most challenging problems.

Group 3: The last group contains the remaining 108 cases corresponding to the remaining 18 graphs, with 500 vertices and up to 12500 edges. These cases are really large scale and no result is reported by any exact algorithm. However, like the 16 special instances of the second group, for each of these instances, the allowed budget is abundant enough that heuristics can easily reach the upper bound of the collected revenues. Hence, these cases have all been solved to optimality (with collected revenues reaching the upper bound) by previous heuristics.

It should be mentioned that the four heuristics proposed in Costa et al. (2008), i.e., greedy, D&R, TS(2000), TS(10000), were evaluated in two different modes. For each test case, the two deterministic heuristics, i.e., greedy and D&R, were executed only once, while the two randomized heuristics, i.e., TS(2000) and TS(10000), were independently run 10 times. Since only the mean computing time of the 10 runs for TS(2000) and TS(10000) was reported, the total used time for 10 runs should be multiplied by 10 (we have confirmed this point with Dr. AM Costa), just as shown in the following Tables. For fair comparisons, we also evaluate the performance of our BLS algorithm in two different modes, named Single-BLS and Multiple-BLS respectively. With Single-BLS, we run our BLS algorithm only once for each test case, while for Multiple-BLS, we independently run BLS 10 times from different initial solutions generated by the probabilistic constructive procedure (Section 3.2). The cutoff time for each independent run is set to 12 minutes (for Single-BLS), thus up to two hours (12×10 minutes) is allowed for Multiple-BLS.

Table 1
Parameter settings

Parameters	Section	Description	Values
L_{min}	3.4	lower bound of the jump magnitude L	1
L_{max}	3.4	upper bound of the jump magnitude L	$\|pv(T)\| - 1$
Q	3.3.2	number of local optima for constructing the HSP	100
θ	3.2.5	parameter for determining the probability for selecting and inserting a candidate path	0.3
α	3.4	parameter for setting the threshold of limited distance	0.3
M	3	maximum number of visiting non-improved local optima	100

4.2 Parameters

The six parameters used in our BLS algorithm are given in Table 1. L_{min} and L_{max} are respectively the lower and upper bound of the jump magnitude L. Since L is tuned in an adaptive way (see Section 3.4), we just need to make sure that L_{min} (L_{max}) is small (large) enough within a reasonable range. In this paper, we set L_{min} to equal 1, and set L_{max} to equal $|pv(T)| - 1$, where $pv(T)$ is the set containing all the profitable vertices of the incumbent solution T. Apparently, L_{min} (L_{max}) is the possible minimum (maximum) number of paths that could be deleted while perturbing the incumbent solution (the root vertex is not permitted to be deleted). Parameter Q is used to control the size of the elite solution pool HSP (Section 3.3.2). Preliminary tests show that when $Q \geq 100$, no statistically significant difference (in terms of solution quality) is observed with different values of Q. Hence we choose 100 as its default value.

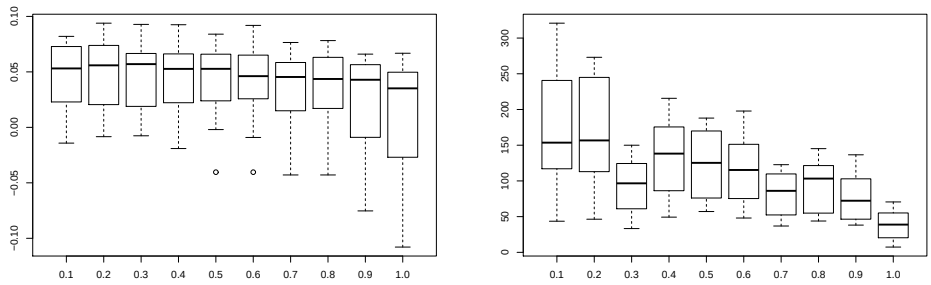

Fig. 2. Box and whisker plots corresponding to different values of $\theta \in (0, 1]$ in terms of solution quality (left sub-figure) and computational time (in seconds, right sub-figure). X-axis indicates the tested θ values and Y-axis shows the performance.

In addition to these three robust parameters, BLS has three other parameters, i.e., θ, α, M. Parameter θ determines the probability for selecting a candidate path to insert to the solution (Section 3.2.5). α is used in the adaptive perturbation mechanism to control the distance between the incumbent solution and the perturbed solution (Section 3.4). M serves as one of the terminal criteria of BLS (second paragraph of Section 3). Preliminary experiments using the Friedman test demonstrate that these three parameters are sensitive, thus deserve a careful tuning. In what follows, we briefly describe how these

parameters are tuned.

In order to identify an appropriate value for a given parameter, we vary its values within a reasonable range and compare their performances, while keeping the other parameters with their default values (as those shown in Table 1). To compare the results in terms of both solution quality and computation time, we use the popular box and whisker plots based on a sample of 12 instances taken (without bias) from the 56 most challenging cases.

For the purpose of conciseness, we take parameter θ as an example and show in Fig. 2 the box and whisker plots obtained with ten different values $\theta \in (0, 1]$. The left sub-figure corresponds to solution quality expressed as the percentage deviation of the obtained results from the best-known results reported in the literature, while the right sub-figure concerns computational time. X-axis indicates the tested θ values and Y-axis shows the performance (solution quality and computational time in seconds). It is clearly observed that small values of θ (i.e., $\theta \leq 0.5$) yield better results than large ones (i.e., $\theta > 0.5$). Specifically, the variant with a deterministic insertion (θ=1.0) which always selects the path with the highest priority for insertion (Costa et al., 2008) performs the worst. In addition, we observe that among the variants with $\theta \leq 0.5$, BLS with θ=0.3 performs the best in terms of computation time. Therefore, $\theta = 0.3$ is used as the default value by BLS.

To tune parameters α and M, we use the same procedure and choose $\alpha = 0.3$, $M = 100$ as their default values.

4.3 Results of the first group of 60 cases

We first consider the first group of 60 cases, corresponding to 10 different graphs of series C. For these cases which have already been solved to optimality by previous exact algorithms, we summarize in Table 2 (the detailed results are provided as an appendix in Table 5) the results obtained by our BLS, with respect to the existing exact or heuristic approaches. In Table 2, column 'Method' lists, in addition to our Single-BLS and Multiple-BLS algorithms (last two rows), the reference approaches. The first six approaches, i.e., S1, S3, S5 (Costa et al., 2009), BP1, BP2, BP3 (Sinnl, 2011) are exact algorithms, while the following four approaches, i.e., greedy, D&R, TS(2000) and TS(10000) (Costa et al., 2008) are heuristics. Column 'Success (Opt)' indicates the number of cases (out of 60) for which the optimal value is reached by each method. Column 'Fail (Sub-Opt)' indicates the number of cases where the corresponding exact approach cannot terminate within the allowed time (5256 seconds in Costa et al. (2009) and 11500 seconds in Sinnl (2011)) or the corresponding heuristic approach misses the optimal solutions. Columns 'Mean

Table 2
Summarized results of the first group of 60 cases

	Method	Success (Opt)	Fail (Sub-Opt)	Mean Gap	Max Gap	Mean Time(s)	Max Time(s)
Exact	S1	59	1	-	-	190.38	5256
	S3	59	1	-	-	250.26	5256
	S5	36	24	-	-	2164.78	5256
	BP1	58	2	-	-	766.33	11500
	BP2	59	1	-	-	443.01	11500
	BP3	59	1	-	-	376.65	11500
Heuristics	Greedy	28	32	9.68%	21.86%	0.02	0.12
	D&R	34	26	8.09%	13.77%	2.26	17.51
	TS(2000)	29	31	5.36%	11.42%	30.05	53.22
	TS(10000)	30	30	4.26%	9.75%	145.55	252.51
	Single-BLS	**35**	25	**2.78%**	7.82%	17.70	81.22
	Multiple-BLS	**43**	17	**1.48%**	3.94%	79.51	511.39

Gap' and 'Max Gap' respectively list the mean gap (only for the cases missing the optimal solutions) and maximal gap between the optimal solutions and the best solutions obtained by each heuristic approach (exact approaches can always solve the problem to optimality, unless they cannot terminate within the limited time). Columns 'Mean Time' and 'Max Time' present the mean CPU time and maximal CPU time (in seconds, after harmonizing by SPEC, so as the follows) used by each method to reach its solutions. For our Single-BLS and Multiple-BLS algorithms, the preprocessing time (see Section 3.2.1) is also included, as well as in the following tables.

As shown in Table 2, the previous exact approaches, i.e., S1, S3, S5, BP1, BP2, BP3 can respectively solve 59, 59, 36, 58, 59, 59 cases to optimality within different time limits (5256s for S1, S3, S5 and 11500s for BP1, BP2, BP3), with a mean time of 190.38s, 250.26s, 2164.78s, 766.33s, 443.01s, 376.65s, respectively. On the other hand, the previous heuristics: greedy, D&R, TS(2000), TS(10000) respectively reach 28, 34, 29, 30 optimal solutions, with a mean time of 0.02s, 2.26s, 30.05s, 145.55s, and with mean gaps 9.68%, 8.09%, 5.36%, 4.26% respectively. For comparison, BLS with a single run (Single-BLS) and multiple runs (Multiple-BLS) respectively reaches 35 and 43 optimal results out of these 60 cases, with a small mean gap of 2.78% and 1.48%, while consuming a mean time of 17.70s and 79.51s. Multiple-BLS clearly dominates TS(10000), which is the best heuristic in Costa (2006) and Costa et al. (2008). It is remarkable that even Single-BLS performs better than TS(10000) in terms of both solution quality and computation time, indicating the effectiveness of BLS against TS. As to the deterministic heuristic D&R, Single-BLS yields much better results than D&R, but requires more computing time. Finally, as a special case of our probabilistic constructive procedure, the greedy algorithm (corresponding to θ=1.0) is extremely fast, but for challenging instances, it generally leads to solutions of inferior quality.

4.4 Results of the second group of 72 cases

We now show the results of BLS on the second group of 72 cases, corresponding to 12 different graphs from series C, with 500 vertices and up to 2500 edges. This group of cases are the most challenging ones of all the three groups of 240 cases, due to two reasons. On one hand, due to their large size, no optimal result has been reported by any exact algorithm. On the other hand, due to their strict budget limitation, except 16 special cases with best known results already reaching the upper bound of the collected revenues, the optimal results of the remaining 56 cases still remain unknown. Therefore, for these 56 cases, there is room for improvement with respect to the best known results reported in the literature.

The results of Single-BLS and Multiple-BLS and those of the reference heuristics (Costa, 2006; Costa et al., 2008) are provided in Table 3. The first three columns respectively denote the graph name, the budget limit $B = \frac{\sum_{(i,j)\in E} c_{ij}}{b}$ and the hop limitation h. The following eight columns list the results reported by the reference heuristics, i.e., columns 4-7 indicate the collected revenues R and consumed CPU time $t(s)$ of the two deterministic heuristics: greedy and D&R, columns 8-11 indicate the best collected revenues R^{best} among 10 runs and the consumed CPU time $t(s)$ of TS(2000) and TS(10000) (after harmonizing by SPEC). Columns 12-13 list the collected revenues R and consumed CPU time $t(s)$ of Single-BLS. The last 5 columns show the results obtained by Multiple-BLS, including the best (R^{best}) and the average (R^{avg}) of the collected revenues among 10 runs, the times that Multiple-BLS improves (column $\surd$) or matches (column =) the best known result among the 10 runs, and its CPU time $t(s)$. The results in bold indicate the best results for each test case, obtained by all the listed algorithms, while the results in italic indicate that the results reach the upper bound.

As shown in Table 3, for the 16 cases with the best known results already reaching the upper bound, Single-BLS and Multiple-BLS can unexceptionally reach the upper bound within very short time. More importantly, for the 56 unsolved cases with unknown optimal solutions, Single-BLS (Multiple-BLS, respectively) succeeds in improving 32 (49) and matching one current best known results. Statistically, the mean improvement gained by Single-BLS (Multiple-BLS, respectively) over the best known results on these 72 cases is 1.34% (3.12%), indicating that BLS produces competitive results for this group of challenging benchmarks. On the other hand, the mean CPU time on these 72 cases corresponding to Single-BLS (Multiple-BLS, respectively) is 36.95s (294.62s), while the mean CPU times corresponding to greedy, D&R, TS(2000), TS(10000) are 0.19s, 29.80s, 55.14s, 258.97s respectively.

Table 3: Detailed results of the second group of 72 cases.

Instance			Greedy		D&R		TS(2000)		TS(10000)		Single-BLS		Multiple-BLS				
Graph	b	h	R	$t(s)$	R	$t(s)$	R^{best}	$t(s)$	R^{best}	$t(s)$	R	$t(s)$	R^{best}	R^{avg}	$\surd$	$=$	$t(s)$
*steinc*8_10	20	5	201	0.01	208	1.32	≤228	14.2	≤**229**	65.4	228	9.1	228	224.0	0	0	81.69
*steinc*8_10	50	5	104	<0.01	104	0.12	≤**116**	13.6	≤**116**	65.2	113	4.7	116	113.7	0	2	30.95
*steinc*8_10	20	15	293	0.07	303	9.51	≤308	42.0	≤319	206.2	316	17.1	**326**	316.4	**5**	0	172.04
*steinc*8_10	50	15	142	0.04	152	1.91	≤161	41.6	≤166	199.1	164	6.8	**167**	164.1	**2**	2	34.63
*steinc*8_10	20	25	294	0.12	311	16.23	≤310	65.6	≤319	324.0	319	16.4	**328**	320.2	**6**	1	150.98
*steinc*8_10	50	25	138	0.05	151	3.19	≤160	63.7	≤166	311.6	164	10.4	**171**	165.8	**5**	1	42.21
*steinc*8_100	20	5	2102	0.01	2261	1.28	≤2365	13.5	≤**2368**	70.3	2327	17.2	2341	2321.7	0	0	100.73
*steinc*8_100	50	5	1067	<0.01	1151	0.23	≤1204	13.5	≤**1220**	66.4	1170	5.7	1201	1177.5	0	0	31.26
*steinc*8_100	20	15	2988	0.07	3269	9.24	≤3237	43.1	≤3306	202.8	3255	20.0	**3378**	3302.7	**5**	0	204.17
*steinc*8_100	50	15	1483	0.03	1696	1.88	≤1675	41.8	≤1705	199.9	1721	7.4	**1763**	1717.2	**8**	0	41.04
*steinc*8_100	20	25	3302	0.14	3310	16.87	≤3337	66.8	≤3340	315.3	3297	24.2	**3392**	3343.0	**6**	0	187.76
*steinc*8_100	50	25	1735	0.06	1735	1.79	≤1742	64.3	≤1755	311.0	1620	8.0	**1780**	1707.1	**1**	0	42.24
*steinc*9_10	20	5	270	0.02	274	2.26	≤279	16.0	≤283	78.5	280	27.4	**284**	277.9	**1**	0	229.10
*steinc*9_10	50	5	143	0.01	143	0.26	≤145	16.1	≤146	76.9	144	5.3	**147**	144.4	**2**	1	32.67
*steinc*9_10	20	15	342	0.08	354	11.35	≤349	42.4	≤354	195.7	365	20.4	**376**	368.6	**10**	0	162.46
*steinc*9_10	50	15	160	0.04	172	2.36	≤165	39.3	≤169	192.9	176	9.6	**181**	178.0	**9**	1	49.24
*steinc*9_10	20	25	353	0.14	353	9.45	353	63.0	353	309.3	375	29.8	**379**	375.0	**10**	0	164.72
*steinc*9_10	50	25	160	0.05	168	3.70	≤167	59.1	≤172	301.2	181	14.1	**183**	182.1	**10**	0	47.65
*steinc*9_100	20	5	2825	0.02	2864	2.24	≤2933	16.4	≤**2954**	77.8	2891	34.4	2921	2862.0	0	0	225.13
*steinc*9_100	50	5	1492	<0.01	1514	0.50	≤1509	15.8	≤1533	83.9	1536	6.5	**1536**	1506.8	**2**	0	42.98
*steinc*9_100	20	15	3568	0.08	3642	11.15	≤3588	41.5	≤3631	194.3	3744	31.1	**3904**	3834.3	**10**	0	188.66
*steinc*9_100	50	15	1626	0.04	1675	2.13	≤1742	38.6	≤1732	192.7	1852	10.0	**1883**	1856.9	**10**	0	54.23
*steinc*9_100	20	25	3556	0.12	3602	17.26	≤3644	62.1	≤3673	309.2	3907	25.8	**3926**	3891.0	**10**	0	212.18
*steinc*9_100	50	25	1626	0.05	1674	3.35	≤1745	59.4	≤1753	296.7	1882	16.7	**1892**	1874.7	**10**	0	51.18
*steinc*10_10	20	5	331	0.02	341	2.07	≤371	14.4	≤372	68.6	365	31.9	**385**	381.0	**9**	0	458.71
*steinc*10_10	50	5	151	0.01	156	0.39	≤173	13.9	≤175	67.7	184	11.6	**184**	179.5	**7**	0	62.49
*steinc*10_10	20	15	488	0.10	505	13.07	≤505	41.4	≤509	203.0	535	19.6	**545**	532.2	**10**	0	134.52
*steinc*10_10	50	15	209	0.04	211	2.54	≤221	40.4	≤226	198.8	242	17.9	**245**	241.4	**10**	0	71.89
*steinc*10_10	20	25	476	0.16	482	19.86	≤501	67.2	≤513	313.8	533	30.0	**547**	535.9	**10**	0	102.18
*steinc*10_10	50	25	210	0.07	219	3.83	≤218	62.9	≤229	311.2	247	27.6	**254**	245.5	**10**	0	80.70
*steinc*10_100	20	5	3216	0.02	3530	1.88	≤3811	15.5	≤3863	68.3	3896	63.7	**4027**	3962.8	**8**	0	463.66
*steinc*10_100	50	5	1483	<0.01	1513	0.32	≤1836	14.0	≤1858	67.9	1918	10.6	**1937**	1916.5	**9**	0	84.66
*steinc*10_100	20	15	5095	0.10	5163	13.30	≤5227	45.5	≤5253	202.0	5491	23.4	**5687**	5572.7	**10**	0	138.55
*steinc*10_100	50	15	2356	0.05	2415	2.88	≤2365	39.7	2356	198.4	2555	16.9	**2555**	2457.4	**7**	0	67.17
*steinc*10_100	20	25	5187	0.18	5287	22.97	≤5286	64.9	≤5331	321.3	5542	26.7	**5642**	5536.5	**10**	0	131.01
*steinc*10_100	50	25	2422	0.07	2472	4.66	≤2432	63.8	2422	311.9	2510	26.6	**2511**	2483.3	**9**	0	86.82
*steinc*13_10	20	5	*439*	0.10	*439*	9.80	*439*	26.9	*439*	128.6	439	2.9	*439*	439.0	0	*10*	15.46
*steinc*13_10	100	5	236	0.04	242	2.67	≤243	26.6	≤246	124.7	240	12.9	**248**	242.1	**1**	0	135.87
*steinc*13_10	20	15	*439*	0.28	*439*	29.72	*439*	79.6	*439*	330.0	439	4.3	*439*	439.0	0	*10*	11.00
*steinc*13_10	100	15	277	0.13	303	12.57	≤296	72.8	≤302	327.4	296	7.3	**308**	298.7	**3**	2	56.55
*steinc*13_10	20	25	*439*	0.45	*439*	50.79	*439*	115.1	*439*	524.5	439	6.9	*439*	439.0	0	*10*	12.74
*steinc*13_10	100	25	291	0.21	302	22.19	≤298	107.3	≤302	520.3	300	9.1	**307**	303.1	**6**	0	46.47
*steinc*13_100	20	5	*4463*	0.10	*4463*	9.82	*4463*	26.9	*4463*	126.7	4463	3.0	*4463*	4463.0	0	*10*	15.96
*steinc*13_100	100	5	2421	0.04	2507	2.48	≤2526	25.9	≤2544	134.4	2558	16.5	**2558**	2522.9	**3**	0	168.50
*steinc*13_100	20	15	*4463*	0.28	*4463*	29.51	*4463*	73.5	*4463*	335.3	4463	4.3	*4463*	4463.0	0	*10*	10.72
*steinc*13_100	100	15	2860	0.12	3064	11.64	≤3029	70.4	≤3111	326.4	3220	8.7	**3236**	3152.2	**8**	0	59.53
*steinc*13_100	20	25	*4463*	0.45	*4463*	51.06	*4463*	118.1	*4463*	541.4	4463	6.3	*4463*	4463.0	0	*10*	11.95
*steinc*13_100	100	25	2991	0.20	3064	22.72	≤3057	109.8	≤3136	521.3	3092	9.6	**3248**	3168.8	**7**	0	64.68
*steinc*14_10	20	5	*648*	0.15	*648*	20.01	*648*	27.7	*648*	128.2	648	4.7	*648*	648.0	0	*10*	25.87
*steinc*14_10	100	5	339	0.05	344	5.33	339	26.8	≤341	130.3	366	27.0	**367**	338.4	**4**	0	144.91
*steinc*14_10	20	15	*648*	0.39	*648*	55.12	*648*	75.0	*648*	319.7	648	6.9	*648*	648.0	0	*10*	18.54
*steinc*14_10	100	15	369	0.15	377	17.72	≤370	71.4	≤371	312.7	396	10.4	**399**	392.5	**10**	0	47.99
*steinc*14_10	20	25	*648*	0.61	*648*	89.17	*648*	111.0	*648*	506.8	648	10.0	*648*	648.0	0	*10*	22.02
*steinc*14_10	100	25	369	0.23	377	28.02	369	101.8	≤373	493.2	394	14.4	**397**	390.7	**10**	0	58.78
*steinc*14_100	20	5	*6566*	0.15	*6566*	20.59	*6566*	28.1	*6566*	127.5	6566	4.9	*6566*	6566.0	0	*10*	26.40
*steinc*14_100	100	5	3364	0.05	3485	5.02	≤3416	26.3	≤3503	124.2	3333	26.1	**3824**	3699.8	**8**	0	158.45
*steinc*14_100	20	15	*6566*	0.39	*6566*	54.62	*6566*	74.0	*6566*	322.1	6566	6.8	*6566*	6566.0	0	*10*	18.98
*steinc*14_100	100	15	3823	0.16	3846	17.32	≤3872	71.5	≤3929	312.9	4048	15.2	**4122**	4063.3	**10**	0	68.00
*steinc*14_100	20	25	*6566*	0.61	*6566*	90.75	*6566*	106.7	*6566*	507.4	6566	10.7	*6566*	6566.0	0	*10*	22.17
*steinc*14_100	100	25	3823	0.24	3846	27.43	≤3856	102.2	≤3897	495.3	4120	15.7	**4120**	4073.0	**10**	0	72.65
*steinc*15_10	20	5	1162	0.23	1174	79.48	≤1192	28.9	≤1211	130.0	1206	758.0	**1213**	1204.5	**2**	0	7295.82
*steinc*15_10	100	5	397	0.07	401	4.94	≤427	26.2	≤435	126.4	414	20.3	**451**	416.8	**1**	0	196.81
*steinc*15_10	20	15	*1248*	0.78	*1248*	179.23	*1248*	78.6	*1248*	336.0	1248	17.7	*1248*	1248.0	0	*10*	68.00
*steinc*15_10	100	15	493	0.19	515	23.48	≤501	71.0	≤502	324.1	545	21.4	**548**	540.7	**10**	0	93.39
*steinc*15_10	20	25	*1248*	1.22	*1248*	281.76	*1248*	118.6	*1248*	524.4	1248	21.7	*1248*	1248.0	0	*10*	65.99
*steinc*15_10	100	25	510	0.31	520	38.60	510	107.7	510	506.6	539	26.4	**546**	540.4	**10**	0	77.24
*steinc*15_100	20	5	11963	0.23	12078	83.31	≤12198	28.5	≤**12306**	131.5	12121	752.6	12213	12106.9	0	0	7023.82
*steinc*15_100	100	5	4108	0.06	4180	5.17	≤4358	27.1	≤**4379**	125.9	4378	23.6	4378	4268.3	0	0	271.20
*steinc*15_100	20	15	*12533*	0.81	*12533*	183.24	*12533*	74.5	*12533*	333.5	12533	16.8	*12533*	12533.0	0	*10*	71.29
*steinc*15_100	100	15	5153	0.20	5355	24.04	≤5177	70.8	≤5159	326.6	5642	21.2	**5770**	5642.7	**10**	0	106.43
*steinc*15_100	20	25	*12533*	1.24	*12533*	298.06	*12533*	118.8	*12533*	555.2	12533	24.3	*12533*	12533.0	0	*10*	76.11
*steinc*15_100	100	25	5243	0.30	5393	41.07	5243	106.7	≤5274	524.9	5663	27.1	**5703**	5655.3	**10**	0	110.16

Specifically, statistical results show that Multiple-BLS gains a mean improvement of 3.44% over TS(10000), while consuming nearly the same mean CPU time. Furthermore, Single-BLS leads to a mean improvement of 1.66%, 2.66%, 4.01% over TS(10000), TS(2000), and D&R, respectively, while consuming much less CPU time than TS(10000), TS(2000), and about 23% more CPU time than D&R. Finally, we do not directly compare BLS with the greedy heuristic, given that it is a special case of our probabilistic constructive procedure for initialization.

In addition, from Table 3, we see that Multiple-BLS performs very well for the cases with large h, but fails to match the best known results of six cases with a very small hop limitation, i.e., $h = 5$. One explanation is that when h is very small, cycles frequently occur after inserting a new path to the incumbent solution. In this case, the proposed algorithm should detect and eliminate cycles efficiently. Our current version of BLS perhaps does not deal well with this special situation, which merits certainly further research.

4.5 Results of the third group of 108 cases

The last group of 108 large cases from 18 graphs of series C have 500 vertices and up to 12500 edges. They are too large to be solved by the existing exact algorithms. However, as described in Section 4.1, all the cases of this group have large budgets so that several heuristics can reach the upper bound of the collected revenues. Indeed, these cases have all been solved to optimality by previous heuristics. Like for the first group, we summarize the existing results and our results in Table 4 (the detailed results are provided as an appendix in Table 6). The meaning of each column in Table 4 is similar to that of Table 2. From Table 4, one observes that the previous heuristics: greedy, D&R, TS(2000), TS(10000) respectively miss four, zero, two, and two optimal results out of the 108 cases, while Single-BLS and Multiple-BLS respectively miss one and zero optimal result with much less computing times than the main reference heuristics TS(10000), TS(2000), and D&R.

Table 4
Summarized results of the third group of 108 cases

Method	Opt	Sub-Opt	Mean Time(s)	Max Time(s)
Greedy	104	4	3.01	23.20
D&R	108	0	485.82	4740.33
TS(2000)	106	2	676.26	2099.77
TS(10000)	106	2	3599.84	12180.78
Single-BLS	107	1	20.47	744.71
Multiple-BLS	108	0	125.78	5757.81

5 Conclusion

We have proposed a Breakout Local Search (BLS) algorithm for the Steiner tree problem with revenue, budget and hop constraints (STPRBH), which can model a number of network designing problems. The proposed BLS approach relies on a probabilistic constructive procedure for initialization, a neighborhood search procedure based on several specifically designed move operators for local optimization, and an adaptive breakout perturbation mechanism for escaping from local optima. Experiments based on three groups of 240 representative benchmarks from the literature demonstrate that BLS is a competitive algorithm for the STPRBH, compared to previous proposed approaches. For the 184 cases with known optimal solutions, BLS can attain an optimal result for 167 cases (90.8%) within short computing time. More importantly, for the 56 most challenging cases with unknown optimal solutions, BLS finds 49 improved solutions and matches one more best known results within reasonable time. Nevertheless, BLS misses the best known solutions for six cases, in particular with very small hop limitations.

The work described in this paper can be further extended by following several directions: (1) design specific techniques to deal with the cases with very small hop limitations, (2) investigate other move operators and perturbation strategies, (3) combine BLS with a population-based method like the memetic framework, and (4) adapt the ideas of this research to other STP variants.

Acknowledgements

We are grateful to the reviewers for their helpful comments and suggestions. The work is partially supported by the Conseil General 49 and the Pays de la Loire Region within the RaDaPop (2009-2013) and LigeRO (2010-2014) projects. Our sincere thanks to Dr. AM Costa for kindly making the benchmark instances available to us and answering our questions.

References

[1] Akgün, I., 2011. New formulations for the hop-constrained minimum spanning tree problem via Sherali and Driscoll's tightened Miller-Tucker-Zemlin constraints. Computers & Operations Research 38(1), 277-286.

[2] Avella, P., Villacci, D., & Sforza, A., 2005. A Steiner arborescence model for the feeder reconfiguration in electric distribution networks. European Journal of Operational Research 164(2), 505-509.

[3] Beasley, J.E., 1990. OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society 41(11), 1069-1072.

[4] Benlic, U., & Hao, J.K., 2012. A study of breakout local search for the minimum coloring problem. In L. T. Bui et al. (Eds.): Lecture Notes in Computer Science 7673, 128-137.

[5] Benlic, U., & Hao, J.K., 2013a. Breakout local search for maximum clique problems. Computers & Operations Research 40(1), 192-206.

[6] Benlic, U., & Hao, J.K., 2013b. Breakout local search for the quadratic assignment problem. Applied Mathematics and Computation 219(9): 4800-4815.

[7] Benlic, U., & Hao, J.K., 2013c. Breakout local search for the max-cut problem. Engineering Applications of Artificial Intelligence 26(3), 1162-1173.

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

[9] Costa, A.M., Cordeau, J.-F., & Laporte, G., 2006. Steiner tree problems with profits. INFOR 44, 99-115.

[10] Costa, A.M., Cordeau, J.-F., & Laporte, G., 2008. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. European Journal of Operational Research 190(1), 68-78.

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

[12] Garey, M.R., Graham, R.L., & Johnson, D.S., 1977. The complexity of computing Steiner minimal trees. SIAM Journal on Applied Mathematics 32(4), 835-859.

[13] Haouari, M., Layeb, S.B., & Sherali, H.D., 2013. Tight compact models and comparative analysis for the prize collecting Steiner tree problem. Discrete Applied Mathematics 161(4-5), 618-632.

[14] Johnson, D.S., Minkoff, M., & Phillips, S., 2000. The prize-collecting Steiner tree problem: Theory and practice. In: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, p. 760-769.

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

[16] Lourenco, H.R., Martin, O., & Stützle, T., 2003. Iterated local search. Handbook of Metaheuristics. Berlin Heidelberg:Springer-Verlag.

[17] Prim, R.C., 1957. Shortest connection networks and some generalizations. Bell System Technical Journal 36, 1389-1401.

[18] Sinnl, M., 2011. Branch-and-Price for the Steiner tree problem with revenues, budget and hop constraints. Diplom-Ingenieur, Fakultät für Informatik der Technischen Universität Wien.

[19] Voβ, S., 1999. The Steiner tree problem with hop constraints. Annals of Operations Research 86, 321-345.

[20] Voβ, S., 2006. Steiner tree problems in telecomunications. 2nd ed. In: Pardalos P, Resende MGC, editors. Handbooks of Optimization in Telecommunications, New York: Springer; p. 459-492.

Appendix

Herein, we provide as appendices in Table 5 and Table 6 the results obtained by our BLS (Single-BLS, Multiple-BLS respectively) for the first and third group of test cases (each row lists two cases, corresponding to the same graph and the same hop constraint h, but different budget constraint $B = \frac{\sum_{(i,j)\in E} c_{ij}}{b}$). In Table 5, column 'Opt' indicates the optimal results obtained by previous exact algorithms, while in Table 6, column 'UB' indicates the upper bound of the collected revenues calculated by Eq.(2). The meanings of the other columns are similar to the ones in Table 3.

Table 5
Detailed results obtained by BLS for the first group of 60 cases

	Information			Single-BLS		Multiple-BLS		Information			Single-BLS		Multiple-BLS	
Graph	b	h	*Opt*	R	$t(s)$	R^{best}	$t(s)$	b	h	*Opt*	R	$t(s)$	R^{best}	$t(s)$
*steinc*1_10	10	5	8	**8**	0.48	**8**	2.84	30	5	8	**8**	0.48	**8**	2.84
*steinc*1_10	10	15	27	**27**	0.84	**27**	3.36	30	15	27	**27**	0.84	**27**	3.36
*steinc*1_10	10	25	27	**27**	1.18	**27**	3.70	30	25	27	**27**	1.19	**27**	3.71
*steinc*1_100	10	5	71	**71**	0.48	**71**	2.83	30	5	71	**71**	0.48	**71**	2.83
*steinc*1_100	10	15	274	**274**	0.84	**274**	3.36	30	15	274	**274**	0.84	**274**	3.36
*steinc*1_100	10	25	274	**274**	1.18	**274**	3.71	30	25	274	**274**	1.18	**274**	3.70
*steinc*2_10	10	5	32	**32**	0.70	**32**	3.18	30	5	32	**32**	0.70	**32**	3.18
*steinc*2_10	10	15	59	**59**	1.41	**59**	4.11	30	15	53	51	1.71	**53**	6.81
*steinc*2_10	10	25	59	**59**	2.09	**59**	4.78	30	25	53	51	2.34	**53**	7.84
*steinc*2_100	10	5	328	**328**	0.71	**328**	3.20	30	5	328	**328**	0.71	**328**	3.20
*steinc*2_100	10	15	604	**604**	1.41	**604**	4.12	30	15	546	533	1.73	**546**	7.29
*steinc*2_100	10	25	604	**604**	2.09	**604**	4.78	30	25	546	533	2.31	**546**	8.70
*steinc*3_10	10	5	151	**151**	4.14	**151**	9.19	30	5	95	**95**	5.56	**95**	22.79
*steinc*3_10	10	15	289	273	26.68	288	142.18	30	15	129	120	13.66	**129**	46.00
*steinc*3_10	10	25	289	288	29.29	288	135.84	30	25	129	**129**	17.93	**129**	50.43
*steinc*3_100	10	5	1519	**1519**	4.16	**1519**	9.27	30	5	968	**968**	5.95	**968**	23.14
*steinc*3_100	10	15	2971	2970	32.45	2970	173.42	30	15	1343	1238	12.47	**1343**	54.65
*steinc*3_100	10	25	2979	2936	28.31	2969	163.84	30	25	1343	1255	18.81	1296	59.89
*steinc*4_10	10	5	115	**115**	5.85	**115**	10.01	30	5	84	**84**	6.42	**84**	17.46
*steinc*4_10	10	15	336	333	81.22	335	405.63	30	15	134	133	23.93	**134**	79.60
*steinc*4_10	10	25	341	325	50.51	335	391.64	30	25	136	134	28.92	**136**	86.94
*steinc*4_100	10	5	1148	**1148**	5.87	**1148**	10.00	30	5	854	**854**	7.32	**854**	17.19
*steinc*4_100	10	15	3458	3390	64.33	3456	511.40	30	15	1380	1345	20.34	1376	86.58
*steinc*4_100	10	25	3504	3420	63.17	3432	402.54	30	25	1396	1336	27.35	**1396**	102.45
*steinc*5_10	10	5	258	**258**	11.94	**258**	21.96	30	5	154	**154**	12.64	**154**	34.88
*steinc*5_10	10	15	494	467	40.26	478	184.86	30	15	182	181	44.60	**182**	174.50
*steinc*5_10	10	25	495	479	55.83	482	226.52	30	25	183	178	53.60	181	205.71
*steinc*5_100	10	5	2600	**2600**	11.97	**2600**	22.05	30	5	1584	**1584**	14.59	**1584**	36.68
*steinc*5_100	10	15	5032	4814	47.45	4834	218.37	30	15	1857	1830	41.08	1837	146.83
*steinc*5_100	10	25	5044	4808	56.92	4875	211.56	30	25	1860	1845	61.34	1845	173.53

Table 6
Detailed results obtained by BLS for the third group of 108 cases

	Information			Single-BLS		Multiple-BLS		Information			Single-BLS		Multiple-BLS	
Graph	b	h	UB	R	$t(s)$	R^{best}	$t(s)$	b	h	UB	R	$t(s)$	R^{best}	$t(s)$
*steinc*6_10	20	5	27	**27**	0.50	**27**	3.13	50	5	27	**27**	0.50	**27**	3.13
*steinc*6_10	20	15	27	**27**	0.84	**27**	3.43	50	15	27	**27**	0.85	**27**	3.44
*steinc*6_10	20	25	27	**27**	1.17	**27**	3.76	50	25	27	**27**	1.19	**27**	3.78
*steinc*6_100	20	5	274	**274**	0.50	**274**	3.13	50	5	274	**274**	0.50	**274**	3.13
*steinc*6_100	20	15	274	**274**	0.84	**274**	3.43	50	15	274	**274**	0.84	**274**	3.43
*steinc*6_100	20	25	274	**274**	1.18	**274**	3.77	50	25	274	**274**	1.18	**274**	3.77
*steinc*7_10	20	5	49	**49**	0.75	**49**	3.65	50	5	49	**49**	0.90	**49**	5.22
*steinc*7_10	20	15	59	**59**	1.43	**59**	4.27	50	15	59	**59**	1.49	**59**	5.18
*steinc*7_10	20	25	59	**59**	2.10	**59**	4.94	50	25	59	**59**	2.18	**59**	5.81
*steinc*7_100	20	5	503	**503**	0.75	**503**	3.66	50	5	503	**503**	0.95	**503**	5.55
*steinc*7_100	20	15	604	**604**	1.42	**604**	4.27	50	15	604	**604**	1.50	**604**	5.21
*steinc*7_100	20	25	604	**604**	2.11	**604**	4.95	50	25	604	**604**	2.22	**604**	5.82
*steinc*11_10	20	5	27	**27**	0.50	**27**	3.11	100	5	27	**27**	0.50	**27**	3.11
*steinc*11_10	20	15	27	**27**	0.85	**27**	3.42	100	15	27	**27**	0.85	**27**	3.43
*steinc*11_10	20	25	27	**27**	1.18	**27**	3.76	100	25	27	**27**	1.17	**27**	3.75
*steinc*11_100	20	5	274	**274**	0.50	**274**	3.11	100	5	274	**274**	0.50	**274**	3.11
*steinc*11_100	20	15	274	**274**	0.84	**274**	3.41	100	15	274	**274**	0.85	**274**	3.43
*steinc*11_100	20	25	274	**274**	1.17	**274**	3.75	100	25	274	**274**	1.18	**274**	3.76
*steinc*12_10	20	5	59	**59**	0.76	**59**	3.78	100	5	59	**59**	0.77	**59**	3.79
*steinc*12_10	20	15	59	**59**	1.43	**59**	4.30	100	15	59	**59**	1.43	**59**	4.30
*steinc*12_10	20	25	59	**59**	2.11	**59**	4.99	100	25	59	**59**	2.11	**59**	4.98
*steinc*12_100	20	5	604	**604**	0.76	**604**	3.77	100	5	604	**604**	0.76	**604**	3.78
*steinc*12_100	20	15	604	**604**	1.42	**604**	4.30	100	15	604	**604**	1.43	**604**	4.30
*steinc*12_100	20	25	604	**604**	2.10	**604**	4.97	100	25	604	**604**	2.11	**604**	4.98
*steinc*16_10	100	5	27	**27**	0.51	**27**	3.11	200	5	27	**27**	0.51	**27**	3.12
*steinc*16_10	100	15	27	**27**	0.85	**27**	3.43	200	15	27	**27**	0.84	**27**	3.42
*steinc*16_10	100	25	27	**27**	1.19	**27**	3.77	200	25	27	**27**	1.18	**27**	3.77
*steinc*16_100	100	5	274	**274**	0.51	**274**	3.10	200	5	274	**274**	0.51	**274**	3.11
*steinc*16_100	100	15	274	**274**	0.85	**274**	3.43	200	15	274	**274**	0.84	**274**	3.42
*steinc*16_100	100	25	274	**274**	1.18	**274**	3.77	200	25	274	**274**	1.19	**274**	3.77
*steinc*17_10	100	5	59	**59**	0.77	**59**	3.72	200	5	59	**59**	0.77	**59**	3.71
*steinc*17_10	100	15	59	**59**	1.44	**59**	4.25	200	15	59	**59**	1.44	**59**	4.26
*steinc*17_10	100	25	59	**59**	2.12	**59**	4.95	200	25	59	**59**	2.12	**59**	4.96
*steinc*17_100	100	5	604	**604**	0.77	**604**	3.72	200	5	604	**604**	0.77	**604**	3.76
*steinc*17_100	100	15	604	**604**	1.45	**604**	4.29	200	15	604	**604**	1.44	**604**	4.28
*steinc*17_100	100	25	604	**604**	2.12	**604**	4.94	200	25	604	**604**	2.12	**604**	4.94
*steinc*18_10	100	5	439	**439**	5.98	**439**	27.06	200	5	439	**439**	5.96	**439**	27.03
*steinc*18_10	100	15	439	**439**	10.49	**439**	20.77	200	15	439	**439**	10.46	**439**	20.77
*steinc*18_10	100	25	439	**439**	16.11	**439**	26.35	200	25	439	**439**	16.09	**439**	26.33
*steinc*18_100	100	5	4463	**4463**	5.97	**4463**	26.93	200	5	4463	**4463**	6.01	**4463**	26.96
*steinc*18_100	100	15	4463	**4463**	10.48	**4463**	20.90	200	15	4463	**4463**	10.49	**4463**	20.93
*steinc*18_100	100	25	4463	**4463**	16.09	**4463**	26.45	200	25	4463	**4463**	16.13	**4463**	26.50
*steinc*19_10	100	5	648	**648**	9.70	**648**	46.79	200	5	648	**648**	9.60	**648**	46.77
*steinc*19_10	100	15	648	**648**	16.00	**648**	33.62	200	15	648	**648**	16.03	**648**	33.75
*steinc*19_10	100	25	648	**648**	24.51	**648**	41.91	200	25	648	**648**	24.37	**648**	41.68
*steinc*19_100	100	5	6566	**6566**	9.69	**6566**	48.38	200	5	6566	**6566**	9.74	**6566**	48.23
*steinc*19_100	100	15	6566	**6566**	16.07	**6566**	34.81	200	15	6566	**6566**	16.01	**6566**	34.79
*steinc*19_100	100	25	6566	**6566**	24.41	**6566**	42.70	200	25	6566	**6566**	24.65	**6566**	43.03
*steinc*20_10	100	5	1248	**1248**	31.45	**1248**	216.07	200	5	1248	**1248**	744.71	**1248**	5757.81
*steinc*20_10	100	15	1248	**1248**	36.35	**1248**	110.67	200	15	1248	**1248**	36.16	**1248**	110.39
*steinc*20_10	100	25	1248	**1248**	52.90	**1248**	123.37	200	25	1248	**1248**	53.09	**1248**	123.74
*steinc*20_100	100	5	12533	**12533**	32.40	**12533**	225.40	200	5	12533	12532	628.25	**12533**	5331.57
*steinc*20_100	100	15	12533	**12533**	37.02	**12533**	120.13	200	15	12533	**12533**	37.38	**12533**	120.95
*steinc*20_100	100	25	12533	**12533**	53.77	**12533**	132.44	200	25	12533	**12533**	53.40	**12533**	131.88