Optim Lett (2014) 8:1225–1243
DOI 10.1007/s11590-013-0665-y

ORIGINAL PAPER

An edge-swap heuristic for generating spanning trees with minimum number of branch vertices

Ricardo M. A. Silva · Diego M. Silva · Mauricio G. C. Resende · Geraldo R. Mateus · José F. Gonçalves · Paola Festa

Received: 12 October 2012 / Accepted: 7 June 2013 / Published online: 19 June 2013

Abstract This paper presents a new edge-swap heuristic for generating spanning trees with a minimum number of branch vertices, i.e. vertices of degree greater than two. This problem was introduced in Gargano et al. (Lect Notes Comput Sci 2380:355–365, 2002) and has been called the minimum branch vertices problem by Cerulli et al. (Comput Optim Appl 42:353–370, 2009). The heuristic starts with a random spanning tree and iteratively reduces the number of branch vertices by swapping tree edges with edges not currently in the tree. It can be easily implemented as a multi-start heuristic. We report on extensive computational experiments comparing single-start

R. M. A. Silva
Center of Informatics, Federal University of Pernambuco,
Av. Jornalista Anibal Fernandes, s/n, Cidade Universitária,
CEP 50.740-560 Recife, PE, Brazil
e-mail: rmas@cin.ufpe.br

D. M. Silva
Department of Computer Science,
Federal University of Lavras,
CEP 37200-000 Lavras, MG, Brazil
e-mail: diego.silva@gmail.com

M. G. C. Resende (✉)
Algorithms and Optimization Research Department,
AT&T Labs Research, 180 Park Avenue, Room C241,
Florham Park, NJ 07932, USA
e-mail: mgcr@research.att.com

G. R. Mateus
Department of Computer Science,
Federal University of Minas Gerais,
CEP 31270-010 Belo Horizonte, MG, Brazil
e-mail: mateus@dcc.ufmg.br

and multi-start variants on our heuristic with other heuristics previously proposed in the literature.

Keywords Constrained spanning trees · Branch vertices · Minimum branch vertices problem · Heuristic · Multi-start heuristic · Edge swapping

1 Introduction

Given an undirected unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a vertex $v \in V$ is said to be a *branch* vertex if its degree $\delta(v)$ is greater than 2. In this paper, we consider the *minimum branch vertices* (MBV) problem whose goal is to find a spanning tree of G with minimum number of branch vertices. This problem finds applications in optical multicast network design. In these networks switches use light splitters to replicate the optical signal. Since switches need only be installed at branch vertices of the network, reducing the number of branch vertices will reduce the number of switches and consequently the cost to deploy the switches in the network.

For all $v \in V$, let y_v be a binary variable such that $y_v = 1$ if and only if vertex v is a branch vertex and for all $e \in E$, let x_e be a binary variable such that $x_e = 1$ if and only if edge e is in the spanning tree. Furthermore, let $E(S)$ be the set of edges having both endpoints in $S \subseteq V$ and let $A(v)$ be the set of edges incident to vertex $v \in V$. [2] formulated this problem as the following integer program:

$$\min \sum_{v \in V} y_v \tag{1}$$

s.t.

$$\sum_{e \in E} x_e = |V| - 1, \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall\, S \subseteq V, \tag{3}$$

$$\sum_{e \in A(v)} x_e - 2 \leq (|A(v)| - 2) y_v, \quad \forall\, v \in V, \tag{4}$$

J. F. Gonçalves
LIAAD, Faculdade de Economia do Porto,
Universidade do Porto, Rua Dr. Roberto Frias, s/n,
4200-464 Porto, Portugal
e-mail: jfgoncal@fep.up.pt

P. Festa
Department of Mathematics and Applications
"R. Caccioppoli", University of Napoli FEDERICO II,
Compl., MSA, Via Cintia, 80126 Naples, Italy
e-mail: paola.festa@unina.it

$$y_v \in \{0, 1\}, \quad \forall\, v \in V, \tag{5}$$

$$x_e \in \{0, 1\}, \quad \forall\, e \in E. \tag{6}$$

The objective function (1) minimizes the total number of branch vertices. Constraint (2) must be satisfied by any spanning tree of G and constraints (3) impose that the solution is a forest. Constraints (4) require that if vertex $v \in V$ has degree greater than two, then it must be a branch vertex. Finally, constraints (5)–(6) restrict the decision variables to be binary.

This problem was recently addressed in the literature by several authors. It was introduced by [7] who showed that the problem is NP-hard and presented some nonapproximability results. They also showed conditions which imply strong upper bounds. [4] developed a mixed integer linear formulation which, however, was only used to solve small instances. For large instances the authors proposed three heuristics:*Edge-Weighting Strategy* (EWS), *Node-Coloring Heuristic* (NCH), and *Combined Approach* (CA) (which combines EWS and NCH). [2] introduce four new formulations and their corresponding relaxations. With their algorithms, they computed lower and upper bounds for 80 instances introduced by them.

The edge-swap heuristic (ESH) proposed in this paper starts by constructing a spanning tree of G and iteratively attempts to reduce the number of branch vertices in the tree by exchanging tree edges with edges of G that are not in the tree. We propose a measure that quantifies the influence of removing/inserting edges from/to the current spanning tree and use this measure to carry out the swaps. If removed, edges that are incident to two branch vertices can potentially have more impact in reducing the number of branch vertices than edges that are incident to a single or no branch vertex. Likewise, if removed, an edge that is incident to a single branch vertex can potentially have more impact in reducing the number of branch vertices than an edge that is not incident to any branch vertex. Instead of using a strategy that seeks first to remove edges incident to vertices of degree three, our strategy prioritizes for removal edges incident to high-degree vertices. The removal of a tree edge disconnects the spanning tree with a cut. An edge in this cut, other than the one just removed, will need to be added to obtain a spanning tree. Instead of prioritizing edges incident to two branch vertices, as in the removal phase, we now prioritize edges incident to no branch vertex over edges incident to a single branch vertex and edges incident to a single branch vertex over edges incident to two branch vertices. Similarly, edges incident to low-degree vertices are preferable to edges incident to high-degree vertices.

The paper is organized as follows. In Sect. 2, we describe the new edge-swap heuristic ESH. Computational results are described in Sect. 3 and concluding remarks are made in Sect. 4.

2 Edge-swap heuristic for the minimum branch vertices problem

In this section, we describe the new edge-swap heuristic (ESH) for finding spanning trees with a small number of branch vertices. Pseudo-code for the heuristic is shown in Algorithm 1. The heuristic starts from a random spanning tree T of G. We choose to build T by first generating random edge weights and then solving a minimum spanning tree (MST) problem with Kruskal's algorithm [9]. T is computed in lines 1 and 2 of the pseudo-code. Then, a sequence of edge swaps is made until a stopping criterion is satisfied. Each swap consists of removing an edge from the current tree and replacing it with an edge not in the tree whose insertion results in a new, possibly better, spanning tree.

```
   Data    : G = (V, E).
   Result  : Solution T*.
 1 G' ← RandomWeights(G);
 2 T ← MST(G');
 3 T* ← T;
 4 repeat
 5 |  ExchangeDone ← false;
 6 |  L ← MakeRemovalEdges(T);
 7 |  while ExchangeDone is false and L ≠ ∅ do
 8 |  |  e* = (u*, v*) ← SelectRemovalEdge(L);
 9 |  |  L ← L \ (u*, v*);
10 |  |  T ← T \ (u*, v*);
11 |  |  R ← MakeInsertionEdges(T, G, (u*, v*));
12 |  |  e' = (u', v') ← SelectInsertionEdge(R, T, (u*, v*));
13 |  |  if (α_e' < α_e* ) or (α_e' = α_e* and σ_e' < σ_e*) then
14 |  |  |  T ← T ∪ (u', v');
15 |  |  |  ExchangeDone ← true;
16 |  |  |  if NumBV(T) < NumBV(T*) then
17 |  |  |  |  T* ← T;
18 |  |  |  end
19 |  |  else
20 |  |  |  T ← T ∪ (u*, v*);
21 |  |  end
22 |  end
23 until ExchangeDone is false;
24 return T*;
```

Algorithm 1: Pseudo-code for edge-swap heuristic (ESH) for minimum branch vertices

The swaps are carried out in lines 4 to 23 and are done until the current spanning tree is locally optimal with respect to single edge swaps. In line 5 the edge swap indicator ExchangeDone is set to be false. In line 6, a list L of candidate edges for swapping out is created. This list consists of all edges in the current spanning tree that are incident to at least one branch vertex. For each edge $e = (u, v) \in L$, MakeRemovalEdges computes two values, α_e and σ_e. For a given spanning tree, parameter α_e is 1 if only one endpoint (vertex u or vertex v) is a branch vertex or 2 if both u and v are branch vertices. Parameter σ_e is the sum of the degrees of the endpoints of edge e in the spanning tree. These parameters are used to prioritize spanning tree edges to be swapped out and non-spanning tree edges to be swapped in.

A swap is attempted in the loop in lines 7 to 22. The loop is run while there are edges in L or until an edge swap is done. In line 8 an edge (u^*, v^*) is selected from list L by procedure SelectRemovalEdge. Let $L' \subseteq L$ be the set of edges in L with maximum α_e value. If $|L'| = 1$, then edge $(u^*, v^*) \in L'$ is selected as the candidate for being swapped out. Otherwise, let $L'' \subseteq L'$ be the set of edges in L' with maximum σ_e value. If $|L''| = 1$, then edge $(u^*, v^*) \in L''$ is selected as the candidate for being swapped out. Otherwise, if $|L''| > 1$, then some edge $(u^*, v^*) \in L''$ is selected at random as the candidate for being swapped out.

In lines 9 and 10, edge (u^*, v^*) is removed from list L and from the current spanning tree creating two subtrees, T_1 and T_2. In line 11, the list R of candidate edges for swapping in is created by procedure `MakeInsertionEdges`. These edges are elements of $E \setminus (T_1 \cup T_2 \cup \{(u^*, v^*)\})$ with one endpoint in T_1 and the other in T_2. As before, parameters α_e and σ_e are computed for all edges $e \in R$. Whereas before the parameters were computed with respect to the current spanning tree T, here they are computed with respect to the spanning tree $T \setminus \{e^*\} \cup \{e\}$. Procedure `SelectInsertionEdge` in line 12 selects the candidate edge $(u', v') \in R$ to be swapped in. Let $R' \subseteq R$ be the set of edges in R with minimum α_e value. If $|R'| = 1$, then edge $(u', v') \in R'$ is selected as the candidate for being swapped in. Otherwise, let $R'' \subseteq R'$ be the set of edges in R' with minimum σ_e value. If $|R''| = 1$, then edge $(u', v') \in R''$ is selected as the candidate for being swapped in. Otherwise, if $|R''| > 1$, then some edge $(u', v') \in R''$ is selected at random as the candidate for being swapped in.

The swap of edge e' for edge e^* is accepted in line 13 if $\alpha_{e'} < \alpha_{e^*}$, or if $\alpha_{e'} = \alpha_{e^*}$ and $\sigma_{e'} < \sigma_{e^*}$. If $\alpha_{e'} < \alpha_{e^*}$ then either $\alpha_{e'} = 0$ and $\alpha_{e^*} = 1$, or $\alpha_{e'} = 0$ and $\alpha_{e^*} = 2$, or $\alpha_{e'} = 1$ and $\alpha_{e^*} = 2$. If $\alpha_{e'} = 0$, then the insertion of edge e' will not increase the number of branch vertices and the deletion of edge e^* will either decrease the number of branch vertices by 1 or 2, or will decrease the degree of at least one of its endpoint vertices (one or both of which may be branch vertices). On the other hand, if $\alpha_{e'} = 1$ and $\alpha_{e^*} = 2$, then the insertion of edge e' either creates a new branch vertex or increases the degree of an existing branch vertex. To compensate for this, the removal of edge e^* either reduces the number of branch vertices by 1 or 2, or reduces the degrees of two branch vertices. If $\alpha_{e'} = \alpha_{e^*}$, then they must be both equal to 1 or to 2 (but not 0). If $\sigma_{e'} < \sigma_{e^*}$, then the removal of edge e^* and insertion of edge e' contributes to balancing the degree distribution in T of the branch vertices, whereas if $\sigma_{e'} > \sigma_{e^*}$ then the swap would contribute to unbalancing the degree distribution.

If accepted, the swap is completed in line 14. In line 15 the edge swap indicator `ExchangeDone` is set to be `true`, and if an improvement in the number of branch vertices results, the incumbent solution T^* is updated in line 17. If the swap is not acceptable, edge e^* is reinserted into the current spanning tree T in line 20.

Note that the complexity of Algorithm 1 is $O(|E| \log |V| + |V|^2)$ since computing the MST (using union-find) takes $O(|E| \log |V|)$ time and the loop from line 7 to 22 can run for at most $O(|V|)$ iterations because the initial cardinality of L in line 6 is at most $|V| - 1$. Since each run of the loop from line 7 to 22 either detects local optimality of the current tree or reduces the number of branch vertices by at least a unit, we can conclude that this loop can be run at most $O(|V|)$ times.

Figure 1 shows an example of the application of ESH on a 50-vertex, 188-edge graph. The figure shows intermediate spanning trees found during 12 iterations of the loop from line 4 to line 23 in the pseudo-code of Algorithm 1. The last spanning tree has no branch vertex and is, therefore, optimal.

Since ESH starts from a random spanning tree, it fits naturally within a multi-start scheme. In such a scheme, the heuristic is repeated a number of times, each time with a different seed for the random number generator, and the best spanning tree found

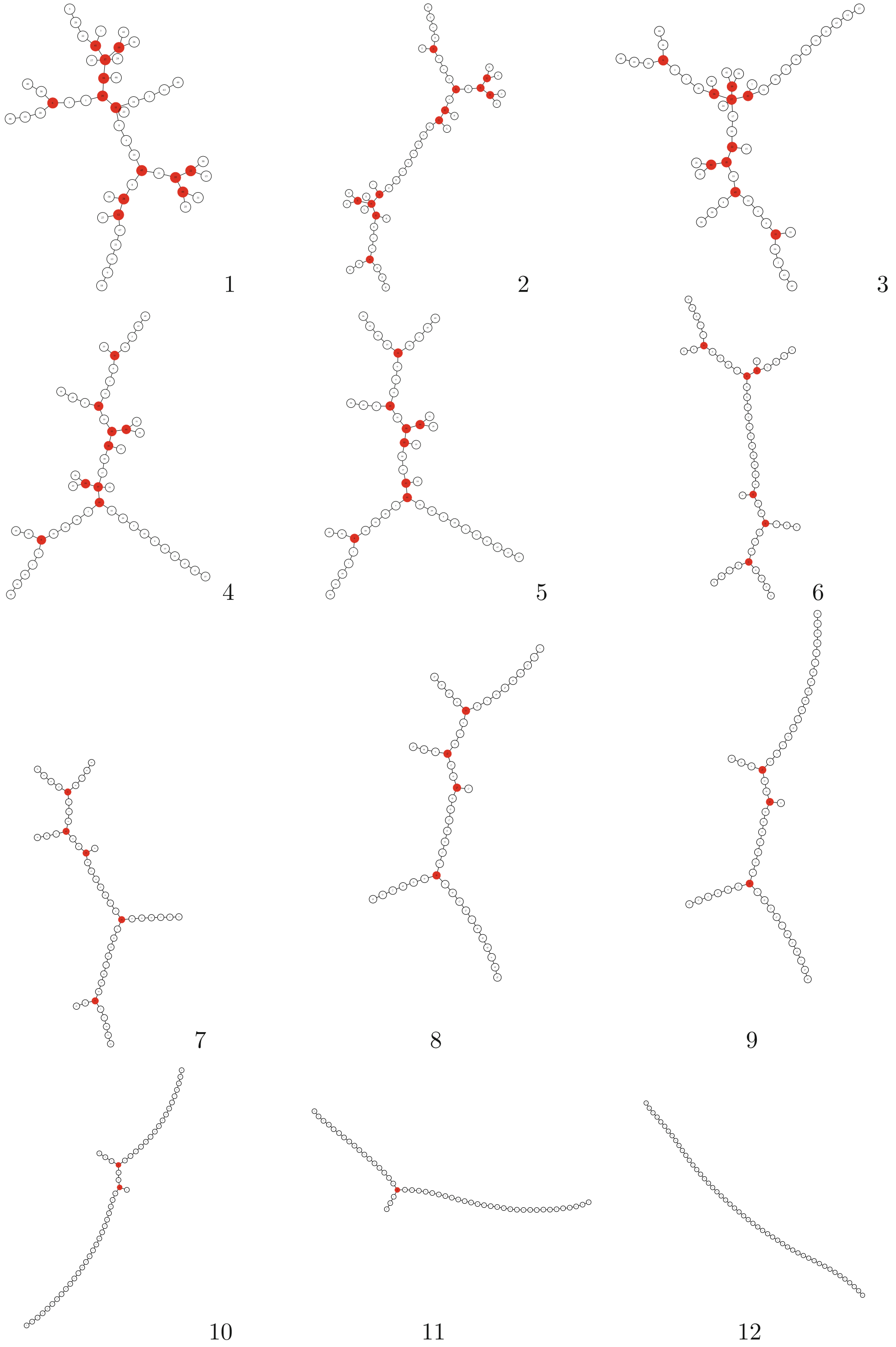

Fig. 1 No branch vertex solution of a 50-vertex, 188-edge instance found with ESH in 12 iterations of the algorithm

over all starts is returned as the solution. In the next section, we run experiments with both single-start and multi-start variants of ESH.

3 Experimental results

In this section, we report on computational experiments with ESH, the new edge-swap heuristic proposed in this paper as well as with our implementations of heuristics EWS and NCH proposed in [4]. We did not implement the combined approach of [4] since their paper does not offer sufficient detail on how this approach was implemented. A more detailed description of the experiments presented in this section can be found in [15].

The algorithms were implemented in C++ and compiled with `gcc` (Ubuntu version 4.3.2-1ubuntu11) and made use of `STL`, the C++ Standard Template Library [12]. We used the C++ implementation of the Mersenne Twister random number generator [11]. All experiments were done on a computer with a 1.66 GHz dual-core T5500 processor with 2048 Kb of cache and 1 Gb of RAM running Linux Ubuntu 11.4.

We implemented Union-Find [6] using `STL` for use in the implementations of the three heuristics. In EWS and NCH, Union-Find is used to determine if two vertices are in different connected components of a graph and in ESH to find the MST with Kruskal's algorithm and to build the list R of candidate edges for insertion.

The computational experiment utilized six classes of benchmark instances:

(1) *Klingman* The 10 instances in this class correspond to the first 10 of the 40 networks proposed by [8]. These instances are `p-1`, `p-2`, ..., and `p-10`. Their sizes vary in the range of 200–300 vertices and 1,300–6,300 edges. They are generated with Klingman's random network generator *Netgen*. Netgen is available at ftp://dimacs.rutgers.edu/pub/netflow/generators/network/netgen/.
(2) *Netgen* The 55 instances in this class are also generated with Netgen and vary in size in the range of 30–500 vertices and 67–18,037 edges.
(3) *TSPLIB* The four instances in this class are `alb1000`, `alb2000`, `alb3000a`, and `alb4000`, proposed in [14] and available through TSPLIB [13] at http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/hcp/. They vary in size in the range 1,000–4,000 vertices and 1,998–7,997 edges.
(4) *Goldberg* The nine instances in this class were generated with the random network generator `crand` which is distributed in the package SPC [5], available at http://www.avglab.com/andrew/soft.html. These instances vary in size in the range 500–1,000 vertices and 6,237–74,925 edges.
(5) *Beasley* Five instances are taken from OR-Library [1]. They are `steind11`, `steind12`, `steind13`, `steind14`, and `steind15`. Each instance has 1,000 vertices and 5,000 edges. They are available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html.
(6) *Leighton* These 12 instances were proposed in [10]. They are `le450_5a`, `le450_5b`, `le450_5c`, `le450_5d`, `le450_15a`, `le450_15b`, `le450_15c`, `le450_15d`, `le450_25a`, `le450_25b`, `le450_25c`, and `le450_25d`. They all have 450 vertices and edges in the range 5,714–17,425 and are available at ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.

Table 1 Heuristic solutions and running times (in s) for Klingman instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	d (%)	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
p-1	200	1,300	7	7	1.13	5	1.12	4	7.00	12	1.69	0.24	0.46	0.70
p-2	200	1,500	8	7	1.29	7	1.30	2	5.87	11	1.90	0.24	0.50	0.78
p-3	200	2,000	10	5	1.88	5	1.58	2	4.83	8	1.47	0.24	0.57	0.92
p-4	200	2,200	11	7	2.19	5	1.94	1	4.23	8	1.61	0.22	0.54	1.11
p-5	200	2,900	15	6	2.95	5	2.57	1	3.79	8	1.44	0.34	0.69	1.11
p-6	300	3,150	7	8	4.51	6	4.17	1	5.61	9	1.69	0.68	1.58	2.71
p-7	300	4,500	10	5	7.28	6	5.96	2	4.54	10	1.65	0.97	2.12	3.48
p-8	300	5,155	11	7	8.61	6	6.84	1	3.49	7	1.50	0.82	2.04	4.40
p-9	300	6,075	14	4	10.93	3	7.96	0	2.97	7	1.46	0.68	2.05	3.76
p-10	300	6,300	14	3	11.59	4	8.56	0	3.67	7	1.21	1.44	3.51	6.78

All of the instances used in the experiment are also available at http://www2.research.att.com/~mgcr/data/mbv.

The experiment consisted in running the new (randomized) edge-swap heuristic (ESH) 100 times, each using a different seed for the random number generator on each of the 95 instances. For each instance, we record the minimum, mean, and maximum number of branch vertices of the solutions produced by the heuristic, as well as its standard deviation. We also record minimum, maximum, and average running times. We ran our implementations of the (deterministic) heuristics EWS and NCH on each instance, recording the number of branch vertices in the solutions produced by each heuristic and the corresponding running times.

Tables 1, 2, 3, 4, 5, and 6 summarize the experimental results. We make the following observations regarding the experiments:

We validated our implementations of the heuristics EWS and NCH of [4] by running them on the 600 instances shared with us for this purpose by [3]. [3] also shared with us average solution values obtained by their implementations of EWS and NCH on 120 blocks of five instances each. [2] reported results for 80 of these 120 blocks. Our implementations of both heuristics were run on each instance and average solution values were computed for each block so we could compare them with the values shared with us by [3].

On the one hand, of the 120 blocks, the average values of the solutions found by our implementation of NCH matched those of [3] in 119 blocks and found a slightly better average value (of 0.1) for one block. On the other hand, the average values of the solutions found by our implementation of EWS matched those of [3] in only 4 of the 120 blocks. Our implementation had better average values in 13 blocks and worse in 103. On the blocks where our implementation found better solutions, the average value difference was 1.23 (with a maximum difference of 4.40). On those where the values reported by [3] were better, the average value

Table 2 Heuristic solutions and running times (in s) for Netgen instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	seed	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
n-01	30	67	1,596	2	0.008	2	0.008	0	0.85	3	0.70	0.000	0.002	0.012
n-02	30	67	2,429	2	0.008	2	0.012	0	0.68	3	0.71	0.000	0.002	0.008
n-03	30	66	7,081	2	0.012	2	0.008	0	1.11	3	0.90	0.000	0.003	0.008
n-04	30	66	7,236	1	0.008	1	0.012	0	1.37	3	0.82	0.000	0.003	0.008
n-05	30	66	7,880	1	0.008	1	0.012	0	1.37	3	0.77	0.000	0.002	0.008
n-06	30	124	1,172	1	0.008	1	0.016	0	0.84	2	0.65	0.000	0.004	0.016
n-07	30	122	2,488	0	0.016	0	0.020	0	0.40	2	0.55	0.000	0.004	0.012
n-08	30	122	4,970	1	0.016	1	0.016	0	0.45	2	0.54	0.000	0.004	0.012
n-09	30	128	5,081	0	0.016	0	0.016	0	0.24	2	0.47	0.000	0.003	0.012
n-10	30	125	8,788	1	0.016	1	0.016	0	0.28	1	0.45	0.000	0.004	0.016
n-11	50	182	1,054	2	0.040	2	0.048	0	1.55	5	1.08	0.000	0.008	0.020
n-12	50	179	3,335	2	0.040	2	0.040	0	1.16	4	0.73	0.000	0.009	0.024
n-13	50	180	4,663	2	0.036	3	0.036	0	1.12	4	0.79	0.000	0.008	0.024
n-14	50	182	4,985	2	0.040	2	0.040	0	1.50	4	0.92	0.000	0.008	0.020
n-15	50	186	7,085	4	0.040	4	0.044	0	1.39	3	0.84	0.000	0.008	0.016
n-16	50	341	1,720	0	0.080	0	0.072	0	0.56	2	0.69	0.004	0.012	0.024
n-17	50	345	6,752	2	0.084	2	0.048	0	0.36	3	0.58	0.004	0.013	0.024
n-18	50	349	7,009	2	0.052	2	0.076	0	0.42	2	0.59	0.004	0.012	0.020
n-19	50	343	7,030	1	0.072	1	0.076	0	0.32	2	0.51	0.000	0.012	0.020
n-20	50	344	9,979	0	0.076	0	0.072	0	0.40	2	0.62	0.004	0.012	0.020
n-21	100	723	2,312	3	0.276	3	0.284	0	1.28	3	0.94	0.024	0.046	0.080

Table 2 continued

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	seed	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
n-22	100	730	299	3	0.268	3	0.256	0	1.09	4	0.95	0.028	0.046	0.072
n-23	100	722	4,414	2	0.236	2	0.316	0	1.41	4	0.98	0.024	0.044	0.068
n-24	100	724	5,885	1	0.212	1	0.292	0	1.50	4	0.99	0.024	0.046	0.084
n-25	100	719	6,570	3	0.296	3	0.228	0	1.69	5	1.12	0.028	0.046	0.084
n-26	100	1,399	5,309	1	0.792	1	0.384	0	0.55	2	0.66	0.040	0.082	0.128
n-27	100	1,383	6,105	1	0.764	1	0.416	0	0.43	2	0.59	0.040	0.076	0.128
n-28	100	1,386	6,259	1	0.772	1	0.464	0	0.40	2	0.57	0.040	0.077	0.112
n-29	100	1,389	7,695	1	0.628	1	0.480	0	0.34	2	0.54	0.036	0.074	0.112
n-30	100	1,391	9,414	0	0.656	0	0.612	0	0.66	3	0.71	0.056	0.083	0.132
n-31	150	1,624	199	3	1.200	2	0.996	0	2.06	6	1.25	0.092	0.146	0.208
n-32	150	1,619	3,738	1	1.112	1	1.060	0	1.69	4	1.05	0.096	0.140	0.264
n-33	150	1,624	5,011	4	1.196	3	1.028	0	1.52	4	1.03	0.072	0.135	0.200
n-34	150	1,627	7,390	2	1.084	2	1.032	0	1.62	5	1.10	0.068	0.146	0.244
n-35	150	1,624	878	3	0.988	2	1.048	0	1.82	5	1.11	0.076	0.147	0.272
n-36	150	3,120	2,051	1	2.808	1	1.800	0	0.46	2	0.58	0.200	0.300	0.424
n-37	150	3,120	2,833	1	2.756	1	1.704	0	0.50	2	0.58	0.204	0.303	0.432
n-38	150	3,141	3,064	1	3.196	1	1.984	0	0.58	3	0.68	0.208	0.330	0.436
n-39	150	3,116	5,357	1	2.648	1	1.564	0	0.29	2	0.48	0.192	0.292	0.416
n-40	150	3,117	5,687	2	2.900	2	1.816	0	0.34	2	0.54	0.164	0.292	0.428
n-41	300	6,502	1,545	1	13.377	1	8.769	0	1.62	4	1.07	0.724	1.042	1.332
n-42	300	6,471	365	3	13.429	3	8.869	0	1.81	5	1.01	0.884	1.054	1.444

Table 2 continued

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	seed	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
n-43	300	6,481	4,071	5	13.377	3	8.545	0	1.61	5	1.13	0.852	1.071	1.432
n-44	300	6,513	4,889	1	13.277	1	8.761	0	1.27	4	0.86	0.852	1.034	1.324
n-45	300	6,505	681	4	13.249	4	8.837	0	1.88	5	0.99	0.868	1.056	1.444
n-46	300	12,539	1,358	2	37.506	2	16.661	0	0.54	3	0.64	2.232	3.004	3.668
n-47	300	12,508	2,067	3	37.478	2	17.257	0	0.34	3	0.55	2.460	3.074	3.748
n-48	300	12,447	4,372	1	36.126	1	17.201	0	0.40	2	0.60	2.464	3.250	3.808
n-49	300	12,480	960	1	37.630	1	17.073	0	0.65	3	0.67	2.372	2.996	3.716
n-50	300	12,474	9,886	1	36.994	1	16.401	0	0.49	3	0.69	1.740	2.939	3.772
n-51	500	18,034	1,456	2	82.825	2	42.139	0	1.85	4	1.12	4.924	5.665	8.073
n-52	500	18,055	1,653	3	82.913	3	42.415	0	1.40	4	1.03	4.860	6.188	8.129
n-53	500	18,009	4,444	2	82.533	2	41.947	0	1.74	5	1.05	4.832	6.678	8.161
n-54	500	18,048	6,849	2	82.925	2	42.275	0	1.81	5	1.06	4.912	6.833	8.181
n-55	500	18,037	8,824	4	82.985	3	42.379	0	1.59	4	0.99	4.776	6.596	7.945

Table 3 Heuristic solutions and running times (in s) for *TSPLIB* instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	d (%)	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
alb1000	1k	1,998	0.4	73	6.6	73	7.9	54	69.1	80	5.2	12.9	19.6	27.9
alb2000	2k	3,996	0.2	129	30.5	141	33.2	121	135.7	155	7.5	127.2	183.5	244.2
alb3000a	3k	5,999	0.1	226	69.5	244	77.0	191	208.6	233	8.1	536.5	713.3	927.9
alb4000	4k	7,997	0.1	277	126.1	308	136.5	247	271.9	298	10.0	1,433.6	1,783.4	2,276.1

Table 4 Heuristic solutions and running times (in s) for *Goldberg* instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	d (%)	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
g−1	500	6,237	5	2	17.3	2	13.5	1	4.7	10	1.8	2.7	4.8	6.5
g−2	500	12,475	10	0	45.4	0	26.6	0	1.8	5	1.1	5.7	8.6	11.9
g−3	500	18,712	15	0	83.6	0	40.1	0	0.9	3	0.8	12.5	15.1	17.6
g−4	800	15,980	5	2	88.1	2	57.4	2	4.5	8	1.5	15.1	22.4	30.6
g−5	800	31,960	10	0	259.1	0	114.5	0	1.8	4	1.0	33.5	47.2	59.6
g−6	800	47,940	15	1	523.0	1	172.5	0	0.9	2	0.7	73.0	89.4	101.8
g−7	1,000	24,975	5	0	191.8	0	107.2	2	4.5	9	1.5	33.9	51.7	64.7
g−8	1,000	49,950	10	1	671.0	1	234.1	0	1.8	4	0.9	92.6	116.0	136.4
g−9	1,000	74,925	15	0	1,569.1	0	366.6	0	0.8	3	0.8	201.2	217.7	236.2

Table 5 Heuristic solutions and running times (in s) for *Beasley* instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	d (%)	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
steind11	1k	5k	1	34	22.4	35	22.3	33	41.4	50	3.5	27.5	46.3	65.5
steind12	1k	5k	1	40	22.3	36	22.4	26	35.5	46	4.5	25.0	40.4	63.5
steind13	1k	5k	1	40	22.2	35	22.5	28	39.8	54	4.2	30.1	44.2	64.7
steind14	1k	5k	1	34	22.3	33	22.4	28	38.2	50	3.9	20.4	44.5	71.5
steind15	1k	5k	1	45	22.4	40	22.5	27	38.9	48	3.6	27.8	45.6	70.3

Table 6 Heuristic solutions and running times (in s) for *Leighton* instances

Benchmark				Cerulli et al. (2009)				Edge-Swap Heuristic						
				EWS		NCH		Branch vertices				Time		
Prob	n	m	d (%)	Value	Time	Value	Time	Min	Mean	Max	Dev	Min	Mean	Max
le450_5a	450	5,714	6	3	13.6	3	11.5	1	4.2	8	1.5	2	3.1	4.8
le450_5b	450	5,734	6	4	14.1	5	11.7	1	4.2	7	1.3	1.8	3.0	4.5
le450_5c	450	9,803	10	3	28.9	3	20.5	0	1.9	4	1.1	3.0	3.8	5.1
le450_5d	450	9,757	10	2	29.0	3	20.2	0	1.9	5	1.0	2.9	3.8	5.9
le450_15a	450	8,186	8	7	22.1	6	16.4	4	6.6	10	1.5	2.6	4.7	8.3
le450_15b	450	8,169	8	10	22.1	9	16.4	3	7.6	12	1.8	2.9	5.5	8.4
le450_15c	450	16,680	17	3	64.7	3	35.0	0	1.1	3	0.9	3.4	5.5	7.8
le450_15d	450	16,750	17	3	65.6	3	35.1	0	1.1	3	0.8	3.6	5.4	7.3
le450_25a	450	8,160	8	12	23.0	11	17.0	8	13.7	19	2.4	3.8	8	15.1
le450_25b	450	8,263	8	10	23.0	7	17.2	4	8.9	13	2.1	3.6	6.1	10.1
le450_25c	450	17,343	17	4	69.8	3	36.4	0	2.0	5	1.2	5.4	7.2	10.4
le450_25d	450	17,425	17	1	69.6	1	36.4	0	1.5	4	1.0	5.3	6.8	9.1

difference was 3.68 (with a maximum difference of 16.2). The average value of the solutions reported by [3] was about 93.1% of that found by our implementation of EWS. A possible explanation for this difference is line 8 of Algorithm 1 of [4] where arc (u^*, v^*) is selected from list L. The pseudo-code makes reference to a tie-breaking rule but, even though the paper states that the tie-breaking rule is very important, it does not elaborate on this rule. We break ties by selecting the arc with the smallest index. Perhaps this is a different tie-breaking criterion than the one implemented in the code used by [3].

Since both [3] and [2] reported similar average solution values for NCH and EWS and our implementation of NCH matches the solutions of [3], then our solution values for NCH should be a good estimate for those of EWS.

For each instance, the tables list the name of the instance, its dimension, density (with the exception of *Netgen*), the solution values and running times (in s) of the heuristics EWS and NCH, as well as statistics for the 100 runs of ESH (minimum, mean, maximum solutions, as well as the standard deviation), and running times (in s) for ESH (minimum, mean, and maximum).

The minimum value for the solution obtained by ESH corresponds to the solution found by the 100-iteration multi-start variant of ESH. On only a single instance in the experiment, the 100-iteration multi-start variant of ESH failed to find a solution that was better than or equal to the best solutions found by either EWS or NCH. Running times for the 100-iteration multi-start variant of ESH are about 100 times the mean running time shown in the tables for ESH (some time should be deducted to account for the multiple inputs of the problem data). The running times for the 100-iteration multi-start variant were always greater than those of both EWS and NCH.

On the *Klingman* instances, the average solution values found by EWS and NCH were 5.9 and 5.2, respectively. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 80 % of the instances. The solutions found by the 100-iteration multi-start variant of ESH, however, were strictly better than the best solutions found by either EWS or NCH on all instances. The maximum running times for the single-start variant of ESH were smaller than those of both EWS and NCH.

On the *Netgen* instances, the average solution values found by EWS and NCH were 1.78 and 1.67, respectively. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 83 % of the instances. The solution found by the 100-iteration multi-start variant of ESH was strictly better than the best solution found by either EWS or NCH on 91% of the instances. Furthermore, the 100-iteration multi-start variant of ESH was never worse than either EWS or NCH. For all instances in this class, the 100-iteration multi-start variant of ESH found solutions with no branch vertex. The maximum running times for the single-start variant of ESH were smaller or equal than those of both EWS and NCH for all instances but one.

On the *TSPLIB* instances, the average solution values found by EWS and NCH were 176.25 and 191.50, respectively. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 75 % of the instances. The solution found by the 100-iteration multi-start variant of ESH was strictly better than the best solution found by either EWS or NCH on all instances. However, the minimum running times for the single-start variant of ESH were greater than those of both EWS and NCH for all instances.

On the *Goldberg* instances, the average solution values found by EWS and NCH were both equal to 0.67. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 11 % of the instances. However, the 100-iteration multi-start variant of ESH was better than or equal to the best solution found by either EWS or NCH in 8 of the 9 instances in this class. It was strictly better on 3 of the 9 instances. The maximum running times for the single-start variant of ESH were smaller than those of both EWS and NCH for all instances.

On the *Beasley* instances, the average solution values found by EWS and NCH were 38.6 and 35.8, respectively. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 33 % of the instances. The solution found by the 100-iteration multi-start variant of ESH was strictly better than the best solution found by either EWS or NCH on all instances. However, the minimum running times for the single-start variant of ESH were greater than those of both EWS and NCH for all but one instance.

On the *Leighton* instances, the average solution values found by EWS and NCH were 5.17 and 4.75, respectively. The average solution values found by ESH were better than the best solutions found by either EWS or NCH in 50 % of the instances. The solution found by the 100-iteration multi-start variant of ESH was strictly better than the best solution found by either EWS or NCH on all instances. The maximum running times for the single-start variant of ESH were smaller than those of both EWS and NCH for all instances.

4 Concluding remarks

In this paper we introduced a new edge-swap heuristic (ESH) for finding a spanning tree with a small number of branch vertices, i.e. vertices with degree greater than two. This problem was called the *minimum branch vertices* (MBV) problem by [4]. It finds applications in optical multicast network design. ESH starts from a random spanning tree and by way of simple edge swaps generates a sequence of spanning trees with the objective of ending up with a spanning tree with no or few branch vertices. ESH comes in two flavors, a single-start variant which is applied a single time, starting from a single spanning tree, and a multi-start variant which repeatedly applies the single-start variant, each time starting from a different random spanning tree. Since iterations of the multi-start algorithm are independent of each other, this heuristic can be easily implemented in parallel. We implemented both variants of ESH in C++ and tested them on a set of benchmark instances that we introduce in this paper for this purpose.

We also present C++ implementations of the heuristics EWS and NCH of [4] and use them to gauge the effectiveness and efficiency of the implementations of ESH. We conducted an experiment with 600 instances provided to us by [3] with EWS and NCH to see if we had reproduced the heuristics described in [4]. Whereas our implementation of NCH matched closely the solutions provided to us by [3], our implementation of EWS did not do as well. Nevertheless, in one of six testbed classes our implementation of EWS did better than our implementation of NCH while in another they tied.

The six classes of testbed instances we introduce in this paper come from a variety of sources and have diverse characteristics. They consist of 95 instances of sizes varying from 30 to 4,000 vertices and 67 to 74,925 edges. For each instance, we ran EWS, NCH, and the single-start and 100-iteration multi-start variants of ESH. For EWS and NCH, we measure solution values and running times. For the single-start variant of ESH, we compute minimum, average, and maximum solution values for each instance. Likewise, we computed minimum, average, and maximum running times. For the 100-iteration multi-start variant of ESH, we measure the values of solutions found as well as their corresponding running times.

Of the 95 instances, the average solution value of the single-start variant of ESH was strictly less than both values of EWS and NCH in 63 instances (66.3 %). The solution of the multi-start variant of ESH was strictly less than both values of EWS and NCH in 84 instances (88.4 %). Finally, the solution of the multi-start variant of ESH was less than or equal to both values of EWS and NCH in 94 instances (98.9 %). On only one instance (`g-7` of *Goldberg*) was the solution found by either EWS or NCH strictly better than the one found by the multi-start variant of ESH.

The average running time for the single-start variant of ESH was smaller than those of EWS and NCH for four of the six problem classes. On the two where ESH was slower, in one (*Beasley*) it was about a factor of two slower, while in the other (*TSPLIB*) it was up to about a factor of 14 slower.

Acknowledgments The research of R.M.A Silva was partially done while he was a post-doc scholar at AT&T Labs Research in Florham Park, New Jersey, and was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG), Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES), and Foundation for the Support of Development of the Federal University of Pernambuco, Brazil (FADE). José F. Gonçalves was supported by funds granted by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT – Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010. Diego M. Silva was partially supported by CAPES-MINTER Program between the Federal Universities of Minas Gerais and Lavras, Brazil.

References

1. Beasley, J.E.: An SST-based algorithm for the Steiner problem in graphs. Networks **19**, 1–16 (1989)
2. Carrabs, F., Cerulli, R., Gaudioso, M., Gentili, M.: Lower and upper bounds for the spanning tree with minimum branch vertices. Technical Report 3, Department of Mathematics and Computer Science, University of Salerno. Salerno, Italy (2009)
3. Cerulli, R.: Personal, communication. January (2010)
4. Cerulli, R., Gentili, M., Iossa, A.: Bounded-degree spanning tree problems: models and new algorithms. Comput. Optim. Appl. **42**, 353–370 (2009)
5. Cherkassky, B.V., Goldberg, A.V.: Negative-cycle detection algorithms. Technical Report 96–029NEC Research Institute, Inc., Princeton, NJ (1996)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press, Cambridge (2001)

7. Gargano, L., Hell, P., Stacho, L., Vaccaro, U.: Spanning trees with bounded number of branch vertices. In 29th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 2380, pp. 355–365. Springer, Berlin (2002)
8. Klingman, D., Napier, A., Stutz, J.: NETGEN—a program for generating large scale (un)capacitated assignment, transportation, and minimum cost flow network problems. Manage. Sci. **20**, 814–821 (1974)
9. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)
10. Leighton, F.T.: A graph colouring algorithm for large scheduling problems. J. Res. Natl. Bureau Standards. **84**(6), 489–503 (1979)
11. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998)
12. Plauger, P.J., Lee, M., Musser, D., Stepanov, A.A.: C++ Standard Template Library. Prentice Hall PTR, Englewood Cliffs (2000)
13. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
14. Reinelt, G.: TSPLIB 95 documentation. University of Heidelberg, Technical report (1995)
15. Silva, D.M.: Abordagem de refinamento iterativo para o problema da árvore geradora com número mínimo de vértices branch. Master's thesis, U. Federal de Minas Gerais, Belo Horizonte (MG). Brazil (2011)