An Iterative Refinement Algorithm for the Minimum Branch Vertices Problem

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

Dept. of Computer Science, Federal University of Lavras
C.P. 3037, CEP 37200-000, Lavras, MG, Brazil
diego.silva@gmail.com,rmas@dcc.ufla.br
Dept. of Computer Science, Federal University of Minas Gerais
C.P. 702, CEP 31270-010, Belo Horizonte, MG, Brazil
mateus@dcc.ufmg.br,diego.silva@gmail.com
Center of Informatics, Federal University of Pernambuco
Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária,
CEP 50.740-560, Recife, PE, Brazil
rmas@cin.ufpe.br
LIAAD, Faculdade de Economia do Porto,
Rua Dr. Roberto Frias, s/n. 4200-464 Porto, Portugal
jfgoncal@fep.up.pt
Internet and Network Systems Research, AT&T Labs Research
180 Park Avenue, Room C241, Florham Park, NJ 07932 USA
mgcr@research.att.com
Dept. of Mathematics and Applications "R. Caccioppoli",
University of Napoli FEDERICO II
Compl. MSA, Via Cintia - 80126, Napoli, Italy
paola.festa@unina.it

Abstract. This paper presents a new approach to solve the NP-complete *minimum branch vertices problem* (MBV) introduced by Gargano et. al[1]. In spite of being a recently proposed problem in the network optimization literature, there are some heuristics to solve it [3]. The main contribution of this paper consists in a new heuristic based on the iterative refinement approach proposed by Deo and Kumar [2]. The experimental results suggest that this approach is capable of finding solutions that are better than the best known in the literature. In this work, for instance, the proposed heuristic found better solutions for 78% of the instances tested. The heuristic looks very promising for the solution of problems related with constrained spanning trees.

Keywords: Constrained spanning trees, Branch vertices, Iterative refinement.

1 Introduction

Given a undirected unweighted graph $G = (V, E)$ the *minimum branch vertices problem* (MBV) consists in finding the spanning tree of G which has the mini-

mum number of *branch* vertices [1]. A vertex v of G is said to be a *branch* vertex if its degree δ is greater than 2, i.e., $\delta(v) > 2$.

This problem Has been recently proposed in the optimization literature. The main contributions were made by Cerulli et al. [3], who developed a mixed integer linear formulation which is able to find the optimal solution. However, for a reasonable computational running time, the model can only solve small instances. For large instances the authors proposed 3 heuristic methods capable of finding suboptimal solutions for the MBV: *Edge Weighting Strategy* (EWS), *Node Coloring Heuristic* (NCH), and a *combined strategy* (CS) between EWS and NCH. Details about these methods as well as their pseudo-codes can be found in [3].

The paper is organized as follows. In Section 2, we describe the iterative refinement algorithms introduced by Deo and Kumar [2]. In Section 3, we describe our iterative refinement algorithm for minimum branch vertices problem. Computational results are described in Section 4, and concluding remarks are made in Section 5.

2 Iterative refinement and constrained spanning trees

Among the approaches used in the literature to solve NP-complete constrained spanning tree problems there are the iterative refinement algorithms (IR) [2]. Consider the problem of constrained spanning tree defined by a weighted graph G and two constraints, $\mathcal{C}_1$ and $\mathcal{C}_2$, where $\mathcal{C}_1$ consists typically in the minimization of the sum of the weights in the spanning tree. The algorithm IR starts from a spanning tree partially constrained (which satisfies only $\mathcal{C}_1$) and moves at each iteration in the direction of a fully constrained tree (which satisfies $\mathcal{C}_2$), but sacrificing the optimality in relation to $\mathcal{C}_1$.

The general idea of the method is shown in the pseudo-code 1, extracted from [2]. First, a spanning tree T which satisfies only constraint $\mathcal{C}_1$ is constructed. Next, the edges which do not satisfy constraint $\mathcal{C}_2$ in T are identified and their weights in G are modified, originating G'. This is done in such a way that the new spanning tree constructed from G' violates less the constraint $\mathcal{C}_2$, in a step called *blacklisting*, whose aim is to discourage certain edges from reappearing in the next spanning trees. Usually the trick used in the *blacklisting* consists in increasing the weight of the edge associated with a violation of $\mathcal{C}_2$ in T. After each *blacklisting* step, a new spanning tree is constructed which satisfies only $\mathcal{C}_1$. This steps are repeated until a tree that satisfies $\mathcal{C}_2$ is found. Note that, the final spanning tree will satisfy $\mathcal{C}_2$, but will be sub-optimal in relation to $\mathcal{C}_1$.

The iterative refinement method is simple and easy to apply. The core of method consists in the design of a penalty function, or *blacklisting*, specific for the problem being studied. To be effective many important decisions must be taken regarding the number of edges to be penalized, what edges to penalize, and the value of the penalty for each edge to be penalized.

In their paper, Deo and Kumar [2] applied the IR method for the *Degree Constrained Minimum Spanning Tree* problem. The implemented algorithm al-

Algorithm 1 `Iterative-Refinement-Algorithm(`G`, `$\mathcal{C}_1$`, `$\mathcal{C}_2$`)`

1: In graph G find a spanning tree that satisfies $\mathcal{C}_1$
2: **while** spanning tree violates $\mathcal{C}_2$ **do**
3: Using $\mathcal{C}_2$ alter weight of edges in G to obtain G' with new weights
4: In graph G' find a spanning tree that satisfies $\mathcal{C}_1$
5: Set $G \leftarrow G'$
6: **end while**

ternates the computation of the Minimum Spanning Tree (MST) with the increase of the weights on the edges whose degree exceeds a predetermined limit d.

In the *blacklisting*, the edges are penalized by a quantity proportional to:

(i) the number $f[e]$ of degree-violating vertices where the edge e is incident;
(ii) a constant k defined by the user;
(iii) the weight $w[e]$ and the range of weights in current spanning tree, given by $w_{min} \leq w[e] \leq w_{max}$.

All the edges e incident to a degree violating vertex, except for the edge with smallest weight amongst them, are penalized as follows:

$$w'[e] = w[e] + kf[e]\left(\frac{w[e] - w_{min}}{w_{max} - w_{min}}\right) w_{max}. \quad (1)$$

In another paper, Boldon et. al [4] applied the dual-simplex approach to the *Degree Constrained Minimum Spanning Tree Problem* involving iterations in two stages until the convergence criteria are reached. The first stage consists in computing a MST using Prim's algorithm, which in the first iterations will violate the degree constraints of several vertices. The second stage consists in adjusting the weights of the violating edges using a *blacklisting* function which will increase the weight of an edge e as follows:

$$w'[e] = w[e] + fault \times w_{max} \times \left(\frac{w[e] - w_{min}}{w_{max} - w_{min}}\right). \quad (2)$$

In this function, $fault$ is a variable which takes the values 0, 1 or 2 depending on the number of vertices incident to the edge which is currently violating the degree constraint. Note that, this approach is very similar to the one used by Deo and Kumar [2]. The refinement idea is also referred in [5].

Other authors have applied the iterative refinement approach for other tree problems, such as *Diameter-Constrained Minimum Spanning Tree* [6]. In that paper, the authors presented two algorithms using the iterative refinement, IR1 and IR2. IR1 consists in iteratively computing a MST as solution to the tree diameter problem, and applying penalties to a subset of edges of the graph, such that they will be discouraged from appearing in the next iteration. The selection of the edges to modify is associated with presence of these edges or not in long paths of the tree, since its elimination aims at reducing the diameter of the tree.

Let l be a set of edges to be penalized; $w(l)$ the current weight of l; w_{max} and w_{min} the smallest and largest weight in the current spanning tree, respectively; $dist_c(l)$ the distance of the edge l to the central vertex of the path, increased by 1 unit. When the center is the edge l_c, we have $dist_c(l_c) = 1$, and as well there the only edge l incident to one of the extremes of the central edge l_c will have $dist_c(l) = 2$. The penalty imposed to each edge l in the current spanning tree will be:

$$\max\left\{ \left(\frac{w(l) - w_{min}}{dist_c(l)(w_{max} - w_{min})} \right) w_{max}, \epsilon \right\}, \tag{3}$$

where $\epsilon > 0$ is the minimum penalty which guarantees that the iterative refinement will not stay in the same spanning tree when the sum of the edges has penalty zero. The penalty decreases as the edges penalized are more distant from the center of current spanning tree, in such way that a path is broken in two sub-paths significantly shorter instead of a short sub-path and a long sub-path. The algorithm IR2 works almost same way as IR1, except for the fact that it does not recompute a new spanning tree at each iteration. A new spanning tree is created by modifying the current spanning tree, by removing one edge at a time.

3 An iterative refinement algorithm for the MBV problem

Section 2 presented several cases where the iterative refinement approach has been used to solve constrained spanning tree problems. Although these cases deal with weighted graphs, we propose an adaptation of the IR approach to solve the *Minimum Branch Vertices Problem.*

Let $G = (V, E)$ be a unweighted undirected graph representing a network in which we would like to find a spanning with the minimum number of vertices *branch.* By assigning random weights in the interval $[0, \dots, 1]$ to each edge $e \in E$, the graph G becomes a new weighted graph $G' = (V, E)$. A minimum spanning tree T constructed on G' using the Kruskal's algorithm would be the starting solution for the iterative refinement algorithm. However, this initial solution may not satisfy the constraint $\delta(v) \leq 2, \forall\, v \in V$. Therefore, the topology of the initial solution will depend only on the weight that were assigned to the edges of G'.

Usually, the method starts with an initial solution with many vertices *branch.* The spanning tree T will then be modified iteratively, being recreated by changes to the topology of the previous spanning tree, and moves towards better solutions, i.e., with fewer vertices *branch.*

The difference between the iterative process proposed in this paper for the *minimum branch vertices* problem and the other iterative processes published by [2], [5], and [6] is the way the penalties are applied to the violating incident edges. In previous works the idea has been to penalize edges by increasing their weights to discourage them from reappearing in the trees in the next iterations. In this paper, we choose to penalize each violating edge by removing it explicitly from the tree and replacing it with an edge with less violations. A violating

edge of T (denoted by ‘cutting edge’) is selected for removal and is replaced by another edge of G' which is not yet in T (denoted by ‘replacement edge’). Such replacement is defined by the exchange of the weights of the cutting edge and the replacement edge in G'. This replacement of edges continues until there are no replacement can reduce the number of vertices *branch* in the current tree T. The pseudo-code 2 describes the steps of the algorithm, and will be detailed next.

The strategy used to replace the edges is based on two measures of the violation of the edges, expresses as 1) the number of actual extreme vertices violating the edge (α); and 2) the sum of the degree of the extreme edges of the edge minus 2 (σ), which tells the sum of the degree of extremes of edge if we removed it from the tree. Good cutting edges are those edges that have many violating extreme vertices (i. e., with high values of α, followed by high values of σ). In a similar way, a good replacement edge is an edge that contributes the most to the reduction of the number of vertices *branch* in T (i.e., with low values of α, followed by low values of σ).

At each iteration of the refinement, one identifies a cutting edge to be removed from the tree T. Any edge incident to a vertex *branch* can be chosen as a cutting edge; edges of this type are used to construct a list of candidates L_{cut}. Next, we select one of the edges in the candidate list and remove it from L_{cut} and from T. The choice of the cutting edges at each iteration takes into account the degree of violation of the edge in T, quantified by the values α and σ. The edge selected will be the edge that has the largest α followed by the largest σ.

The removal of the cutting edge will divide the tree T into two connected components, and the set V of vertices of T into two sub-sets, S and S'. To avoid cycles, each edge removed from T is inserted into a special set, denoted by B_{list}, which indicates that the edges is tagged and cannot be reinserted in T.

To reconnect the two connected components we need to find an advantageous replacement edge capable of connecting both components without creating cycles. The candidate list L_{rep} includes all the edges in G which are not in T and that are capable of connecting both components and are not in B_{list}. A good replacement edge is an edge that does create violations when it is inserted in T, i.e., an edge that has lower values of α, followed by lower values of σ. Once the replacement edge selected is inserted in T, the current iteration of the algorithm ends.

If there is no advantageous replacement edges, then the replacement does not occur. The cutting edge returns to T. A new cutting edge is selected from L_{cut} and a new list L_{rep} is created to select the replacement edge. The algorithm continues until no more cutting edges can be replaced in T, i.e., $L_{cut} = \emptyset$.

We will illustrate the method using a simple instance of the problem MBV. The steps are shown in Figure 1. In (a) we have an initial spanning tree T, created by applying Kruskal's algorithm on graph G'. The tree shown in (a) is the result of assigning weights to the edges of G', which initial topology indicates the occurrence of two vertices *branch*: vertices 5 and 8.

Algorithm 2 `Mbv-Iterative-Refinement-Algorithm`$(G = (V, E))$

1: $G' \leftarrow$ `AssignRandomWeights`(G)
2: $T \leftarrow$ `CalculateMinimumSpanningTree`(G')
3: $B_{list} \leftarrow \emptyset$
4: **repeat**
5: `ThereWasExchange` $\leftarrow$ `false`
6: $L_{cut} \leftarrow$ `CreateCutList`(T, B_{list})
7: **while** $(($`ThereWasExchange` $\neq$ `true`$) \wedge (|L_{cut}| \neq 0))$ **do**
8: $(u^*, v^*) \leftarrow$ `SelectArcFromCutList`(L_{cut})
9: $L_{cut} \leftarrow L_{cut} \setminus \{ (u^*, v^*) \}$
10: $L_{rep} \leftarrow$ `CreateReplacementListToCutArc`$(T, G', (u^*, v^*))$
11: $(u, v) \leftarrow$ `SelectArcFromReplacementList`(L_{rep})
12: **if** $(\exists\ (u, v))$ **then**
13: $B_{list} \leftarrow B_{list} \bigcup \{(u, v)\}$
14: `SwapWeightsIntoGraph`$((u^*, v^*), (u, v))$
15: $T \leftarrow T \setminus \{(u^*, v^*)\}$
16: $T \leftarrow T \bigcup \{(u, v)\}$
17: `ThereWasExchange` $\leftarrow$ `true`
18: **end if**
19: **end while**
20: **until** $($`ThereWasExchange` $\neq$ `false`$)$
21: **return** T

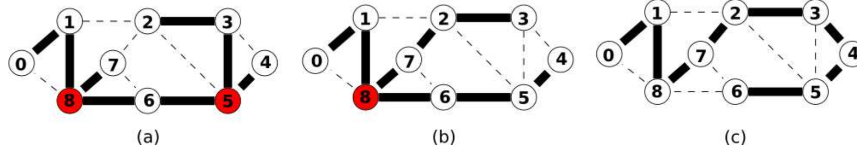

Fig. 1. An example of the iterative refinement algorithm solving an instance of MBV.

In the sequence, the algorithm determines a cutting edge and a replacement edge and tries to transform T into a tree with less violations. L_{cut} is constructed with the incident edges to vertices *branch*, i.e.: $L_{cut} = \{(5,3), (5,4), (5,6), (8,1), (8,6), (8,7)\}$. The edges $(5,3)$, $(5,6)$, $(8,1)$ and $(8,6)$ have the highest values of α, followed by σ. We select edge $(5,3)$, which is removed from T and from L_{cut} and creates a cut in T which separates V in $S = \{2,3\}$ e $S' = \{0,1,4,5,6,7,8\}$.

The edges capable of connecting S and S' are $L_{rep} = \{(1,2), (2,5), (2,7), (3,4)\}$. Amongst these, the edges $(2,7)$ and $(3,4)$ are the only ones that can replace $(5,3)$ without causing any violations in the tree. We choose to replace $(5,3)$ by $(2,7)$, inserting $(2,7)$ in T to obtain (b). This concludes the first iteration of the algorithm.

The next iteration continues from the tree T showed in (b). $L_{cut} = \{(8,1), (8,6), (8,7)\}$, where $(8,1)$ is the most interesting edge to cut since it has the highest values of α and σ. We remove $(8,1)$ from T and L_{cut}, creating the cut $S = \{0,1\}$ and $S' = \{2,3,4,5,6,7,8\}$. The edges capable of connecting S and S' are

$L_{rep} = \{(0,8),(1,2)\}$, which have the same value of α and σ. Edge $(0,8)$ is selected to replace $(8,1)$ in T.

Replacing edge $(8,1)$ by $(0,8)$ does not bring any advantages in T since it does not reduce the number of vertices *branch* in the tree, which will continue having 1 vertex *branch*. The replacement is canceled, edge $(8,1)$ returns to the tree T, and we select a new cutting edge from $L_{cut} = \{(8,6),(8,7)\}$. The edge selected is $(8,6)$, given the value of α and σ. Removing $(8,6)$ from the tree will divide V into $S = \{0,1,8,7,2,3\}$ and $S' = \{6,5,4\}$, with $L_{rep} = \{(2,5),(3,4),(6,7)\}$. The edge $(3,4)$ is the best replacement for edge $(8,6)$, and is inserted into T. The replacement ends the second iteration of the algorithm, resulting in tree (c).

The third iteration begins with $L_{cut} = \emptyset$. There are no more vertices *branch* in the tree and therefore there is no cutting edge available. The tree T presented in (c) is then a solution to be returned by the algorithm.

4 Experimental results

In this section, we present results on computational experiments with the iterative refinement method applied to a set of instances with the purpose of comparing the quality of the results obtained by our IR algorithm with the results obtained by the heuristics EWS and NCH proposed by [3]. All the algorithms cited were implemented in `ANSI C++`, compiled with `gcc version 4.3.2`, using the libraries `STL` and run in the operating system `Ubuntu 4.3.2-1`. The algorithm used to find an initial solution was the Kruskal's algorithm, using efficient data structures to represent the disjoint sets (*union-find structures*). This data structures where used by all methods to determine if two vertices were in different connected components of a graph, as well as to determine the replacement edges candidates capable of connecting S e S'. Details about the efficient implementation of the data structures *union-find* can be found in [7] and [8].

The instances were created by the network flow problem generator NetGen [9], available in `public ftp` from DIMACS [1]. The instances were divided into different classes, each containing different number of vertices and edges. NetGen constructs network flow problems using as input a file that specifies the input parameters. Since the *minimim branch vertices* problem consists of an unweighted and uncapacitated graph, we used only the topology of the graphs input. Repeated edges were ignored.

The input files used by NetGen to generate the instances follow the format given in Table 1. According to the table, the only parameters that can vary are the seed for the random number generator and the number of vertices and edges of the output graph. In table 2 the values presented for each instance in columns d, n and s, correspond to the number of edges, number of vertices, and seed for each instance, respectively. The column m represents the 'real' number of edges of the graph, removing the repeated edges.

[1] `ftp://dimacs.rutgers.edu/pub/netflow/`

Table 1. NetGen parameters for input files.

#	*Parameters*	*Input*	*Parameter Description*
1	SEED	*Variable*	Random numbers seed
2	PROBLEM	1	Problem number
3	NODES	*Variable*	Number of nodes
4	SOURCES	1	Number of sources (including transshipment)
5	SINKS	1	Number of sinks (including transshipment)
6	DENSITY	*Variable*	Number of (requested) edges
7	MINCOST	0	Minimum cost of edges
8	MAXCOST	1000	Maximum cost of edges
9	SUPPLY	1	Total supply
10	TSOURCES	0	Transshipment sources
11	TSINKS	0	Transshipment sinks
12	HICOST	1	Percent of skeleton edges given maximum cost
13	CAPACITED	1	Percent of edges to be capacitated
14	MINCAP	0	Minimum capacity for capacitated edges
15	MAXCAP	3	Maximum capacity for capacitated edges

We have generated instances with 30, 50, 100, 150, 300, and 500 vertices, with edges densities of 15% and 30% in 5 graphs capable of representing each of these classes.

The methods EWS and NCH were run only once, since the runs result in the same deterministic values. The iterative refinement method has been statistically evaluated, since it depends on the weights of the graph G' assigned randomly at the beginning of the algorithm. The methodology used consisted in 100 runs for each instance, each one with a different seed. Table 2 presents the minimum, maximum, average, median, standard deviation, and variance found for the execution time, and the solution value of each instance, respectively in columns 'Min', 'Max', 'Mean', 'Med', 'Dev' and 'Var' of the column 'Value' corresponding to the results of algorithm IR.

The rows of column 'C' are tagged with the character 'y' when the IR methods found solutions with an average number of vertices *branch* (column 'Mean') lower than the values found by the algorithms EWS and NCH. This condition occurred happens in 43 out 55 instances (78% of the instances).

The median values suggests that the IR method performed very well for these instances. For the 55 instances tested, the IR method obtained median values better than the ones obtained by EWS and NCH in 37 instances, and equal values in 15 instances.

Even when the IR method did not obtain the best values, we can see that it obtains value very close to the ones obtained by EWS and NCH.

The histograms 2, 3, 4, 5, 6, and 7 report frequencies computed for the 100 runs of some of the instances in which IR did not obtain better values than EWS and NCH. Note that, the most frequent strip corresponds to values close or equal to the EWS and NCH heuristics.

The reported running times are in seconds. The running times of the EWS and NCH heuristics correspond to only one run. The running time for the IR algorithm varies since the computational effort to find a solution depends on the number of *branch* vertices existing in the initial solution, and therefore should be analyzed statistically by the measures 'Min', 'Max', 'Mean', 'Dev' and 'Var' of the block 'Time'. These measure correspond to the minimum, the maximum, the average, the median, the standard deviation, and variance of the 100 runs of the IR algorithm, respectively.

It is worth mentioning that for each instance from *benchmark* evaluated, there was at least one run of the 100 runs were the IR method obtained a solution with zero vertices *branch*. Figures 8, 9, 10, and 11 depict the difference in topology of some of the best solutions found by IR and the one found by NCH. The vertices highlighted correspond to vertices *branch*, i.e., $\delta(v) > 2$.

5 Concluding remarks

According to the results presented in Section 4, for the *benchmark* used in the paper, the iterative method presented has better performance than the methods proposed by [3]: edge weighting and node coloring strategies. In 78% of the instances the IR algorithm obtained average results better than the ones found by the methods EWS and NCH. The small standard deviation as well as the better median values in 37 of the 55 instances classes further support quality of the IR algorithm compared to EWS and NCH.

The experimental results show that the iterative refined method is an effective approach to solve the MBV directly or as a sub-problem of large problems. Since the test *benchmark* is made of artificial instances generated by NetGen, further research should be conducted using real instances. Comparisons with exact methods such as the algorithms proposed in [3] should also be carried in future experiments.

6 Acknowledgment

Ricardo M.A Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Fundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brazil (CAPES), and Fundação de Apoio ao Desenvolvimento da UFPE, Brazil (FADE). José F. Gonçalves was partially supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006. Diego M. Silva was partially supported by CAPES-MINTER Program between the Federal Universities of Minas Gerais and Lavras, Brazil.

Table 2. Comparison results of 'value' and 'running time' for EWS, NCH and IR

Instances				Cerulli et al				Iterative Refinement Approach												
				EWS		NCH		Value							Time					
d	*n*	*m*	*s*	*Val*	*Time*	*Val*	*Time*	*Min*	*Mean*	*Med*	*Max*	*Dev*	*Var*	*C.*	*Min*	*Mean*	*Max*	*Dev*	*Var*
68	30	67	1596	2	0,004	2	0,004	0	0,85	1	3	0,7	0,49	y	0,000	0,002	0,012	0,003	0,000
68	30	67	2429	2	0,008	2	0,008	0	0,68	1	3	0,71	0,5	y	0,000	0,002	0,008	0,002	0,000
68	30	66	7081	2	0,008	2	0,004	0	1,11	1	3	0,9	0,81	y	0,000	0,003	0,008	0,003	0,000
68	30	66	7236	1	0,012	1	0,008	0	1,37	1	3	0,82	0,68	n	0,000	0,003	0,008	0,003	0,000
68	30	66	7880	1	0,012	1	0,012	0	1,37	1	3	0,77	0,6	n	0,000	0,002	0,008	0,002	0,000
135	30	124	1172	1	0,020	1	0,012	0	0,84	1	2	0,65	0,42	y	0,000	0,004	0,016	0,003	0,000
135	30	122	2488	0	0,028	0	0,004	0	0,4	0	2	0,55	0,3	n	0,000	0,004	0,012	0,003	0,000
135	30	122	4970	1	0,016	1	0,020	0	0,45	0	2	0,54	0,29	y	0,000	0,004	0,012	0,003	0,000
135	30	128	5081	0	0,036	0	0,008	0	0,24	0	2	0,47	0,22	n	0,000	0,003	0,012	0,003	0,000
135	30	125	8788	1	0,032	1	0,016	0	0,28	0	1	0,45	0,2	y	0,000	0,004	0,016	0,003	0,000
188	50	182	1054	2	0,064	2	0,028	0	1,55	1	5	1,08	1,16	y	0,000	0,008	0,020	0,004	0,000
188	50	179	3335	2	0,056	2	0,028	0	1,16	1	4	0,73	0,54	y	0,000	0,009	0,024	0,004	0,000
188	50	180	4663	2	0,052	3	0,024	0	1,12	1	4	0,79	0,63	y	0,000	0,008	0,024	0,005	0,000
188	50	182	4985	2	0,060	2	0,024	0	1,5	1	4	0,92	0,84	y	0,000	0,008	0,020	0,004	0,000
188	50	186	7085	4	0,056	4	0,028	0	1,39	1	3	0,84	0,7	y	0,000	0,008	0,016	0,004	0,000
375	50	341	1720	0	0,132	0	0,048	0	0,56	0	2	0,69	0,47	n	0,004	0,012	0,024	0,005	0,000
375	50	345	6752	2	0,164	2	0,048	0	0,36	0	3	0,58	0,33	y	0,004	0,013	0,024	0,005	0,000
375	50	349	7009	2	0,148	2	0,040	0	0,42	0	2	0,59	0,35	y	0,004	0,012	0,020	0,004	0,000
375	50	343	7030	1	0,160	1	0,040	0	0,32	0	2	0,51	0,26	y	0,000	0,012	0,020	0,004	0,000
375	50	344	9979	0	0,144	0	0,056	0	0,4	0	2	0,62	0,38	n	0,004	0,012	0,020	0,004	0,000
750	100	723	2312	3	0,588	3	0,292	0	1,28	1	3	0,94	0,89	y	0,024	0,046	0,080	0,011	0,000
750	100	730	299	3	0,708	3	0,248	0	1,09	1	4	0,95	0,91	y	0,028	0,046	0,072	0,010	0,000
750	100	722	4414	2	0,544	2	0,244	0	1,41	1	4	0,98	0,95	y	0,024	0,044	0,068	0,010	0,000
750	100	724	5885	1	0,640	1	0,184	0	1,5	1	4	0,99	0,98	n	0,024	0,046	0,084	0,010	0,000
750	100	719	6570	3	0,644	3	0,244	0	1,69	2	5	1,12	1,25	y	0,028	0,046	0,084	0,011	0,000
1500	100	1399	5309	1	2,276	1	0,500	0	0,55	0	2	0,66	0,43	y	0,040	0,082	0,128	0,017	0,000
1500	100	1383	6105	1	1,820	1	0,372	0	0,43	0	2	0,59	0,35	y	0,040	0,076	0,128	0,015	0,000
1500	100	1386	6259	1	2,112	1	0,316	0	0,4	0	2	0,57	0,32	y	0,040	0,077	0,112	0,015	0,000
1500	100	1389	7695	1	2,096	1	0,380	0	0,34	0	2	0,54	0,29	y	0,036	0,074	0,112	0,015	0,000
1500	100	1391	9414	0	2,348	0	0,500	0	0,66	1	3	0,71	0,51	n	0,056	0,083	0,132	0,017	0,000
1688	150	1624	199	3	2,684	2	0,620	0	2,06	2	6	1,25	1,55	n	0,092	0,146	0,208	0,024	0,001
1688	150	1619	3738	1	3,060	1	0,872	0	1,69	2	4	1,05	1,1	n	0,096	0,140	0,264	0,024	0,001
1688	150	1624	5011	4	3,696	3	0,932	0	1,52	1,5	4	1,03	1,06	y	0,072	0,135	0,200	0,024	0,001
1688	150	1627	7390	2	2,804	2	0,700	0	1,62	1,5	5	1,1	1,21	y	0,068	0,146	0,244	0,035	0,001
1688	150	1624	878	3	3,288	2	0,884	0	1,82	2	5	1,11	1,24	y	0,076	0,147	0,272	0,040	0,002
3375	150	3120	2051	1	9,405	1	1,572	0	0,46	0	2	0,58	0,33	y	0,200	0,300	0,424	0,062	0,004
3375	150	3120	2833	1	9,189	1	1,288	0	0,5	0	2	0,58	0,33	y	0,204	0,303	0,432	0,062	0,004
3375	150	3141	3064	1	11,089	1	1,620	0	0,58	0	3	0,68	0,47	y	0,208	0,330	0,436	0,062	0,004
3375	150	3116	5357	1	10,025	1	1,688	0	0,29	0	2	0,48	0,23	y	0,192	0,292	0,416	0,054	0,003
3375	150	3117	5687	2	10,933	2	1,588	0	0,34	0	2	0,54	0,29	y	0,164	0,292	0,428	0,058	0,003
6750	300	6502	1545	1	45,391	1	5,720	0	1,62	2	4	1,07	1,15	n	0,724	1,042	1,332	0,116	0,013
6750	300	6471	365	3	43,331	3	6,144	0	1,81	2	5	1,01	1,02	y	0,884	1,054	1,444	0,108	0,012
6750	300	6481	4071	5	45,731	3	5,888	0	1,61	1,5	5	1,13	1,27	y	0,852	1,071	1,432	0,116	0,013
6750	300	6513	4889	1	45,303	1	5,832	0	1,27	1	4	0,86	0,74	n	0,852	1,034	1,324	0,114	0,013
6750	300	6505	681	4	45,239	4	5,696	0	1,88	2	5	0,99	0,98	y	0,868	1,056	1,444	0,116	0,013
13500	300	12539	1358	2	151,349	2	11,273	0	0,54	0	3	0,64	0,41	y	2,232	3,004	3,668	0,349	0,122
13500	300	12508	2067	3	160,454	2	13,337	0	0,34	0	3	0,55	0,31	y	2,460	3,074	3,748	0,263	0,069
13500	300	12447	4372	1	178,551	1	13,541	0	0,4	0	2	0,6	0,36	y	2,464	3,250	3,808	0,304	0,092
13500	300	12480	960	1	179,371	1	13,513	0	0,65	1	3	0,67	0,45	y	2,372	2,996	3,716	0,333	0,111
13500	300	12474	9886	1	181,843	1	13,369	0	0,49	0	3	0,69	0,47	y	1,740	2,939	3,772	0,453	0,206
18750	500	18034	1456	2	290,798	2	24,350	0	1,85	2	4	1,12	1,26	y	4,924	5,665	8,073	0,716	0,513
18750	500	18055	1653	3	279,593	3	23,586	0	1,4	1	4	1,03	1,07	y	4,860	6,188	8,129	0,853	0,727
18750	500	18009	4444	2	299,959	2	29,026	0	1,74	2	5	1,05	1,1	y	4,832	6,678	8,161	0,924	0,853
18750	500	18048	6849	2	313,808	2	29,798	0	1,81	2	5	1,06	1,13	y	4,912	6,833	8,181	0,913	0,833
18750	500	18037	8824	4	320,636	3	29,142	0	1,59	2	4	0,99	0,97	y	4,776	6,596	7,945	0,893	0,797

References

1. Gargano, L., Hell P., Stacho L., Vaccaro, U.: Spanning Trees with Bounded Number of Branch Vertices: 29th International Colloquium on Automata, Languages and Programming (ICALP). pp. 355–365, 2002
2. Deo, N., Kumar, N.: Computation of Constrained Spanning Trees: A Unified Approach. Network Optimization: Lecture Notes in Economics and Mathematical Systems. 450, pp. 194–220, Springer-Verlag, New York (1997)
3. Cerulli, R., Gentili, M., Iossa, A.: Bounded-Degree Spanning Tree Problems: Models and New Algorithms. Comput. Optim. Appl. Vol. 42, pp. 353–370 (2009)
4. Boldon B., Deo N., Kumar N.: Minimum-Weight Degree-Constrained Spanning Tree Problem: Heuristics and Implementation on an SIMD Parallel Machine. Technical Report CS-TR-95-02, Department of Computer Science, University of Central Florida, Orlando, FL (1995).
5. Mao, L.J., Deo, N., Lang, S.D.: A Comparison of Two Parallel Approximate Algorithms for the Degree-Constrained Minimum Spanning Tree Problem. Congressus Numerantium, 123, pp. 15-32, (1997)
6. Abdalla, A., Deo, N., Gupta, P.: Random-Tree Diameter and the Diameter Constrained MST. Congressus Numerantium, 144, pp. 161–182 (2000)
7. Cormen, T.H., Leiserson, C.E., Rivest R.L., Stein, C.: Introduction to Algorithms. MIT Press, Second Edition, pp. 498–509 (2001)
8. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, pp. 521–522 (1993)
9. Klingman, D., Napier, A., STUTZ, J.: NETGEN – A program for generating large scale (un)capacitated assignment, transportation, and minimum cost flow network problems. Managent Science 20, pp. 814–821 (1974)

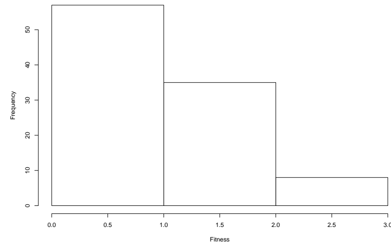

Fig. 2. Histogram for the instance $n = 30$, $m = 68$, $s = 7236$: $NCH_{val} = 1$; $IR_{mean} = 1,37$; $freq[0 \dots 1] \sim 55$.

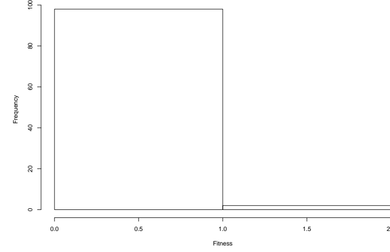

Fig. 3. Histogram for the instance $n = 30$, $m = 135$, $s = 5081$: $NCH_{val} = 0$; $IR_{mean} = 0,24$; $freq[0 \dots 1] \sim 98$.

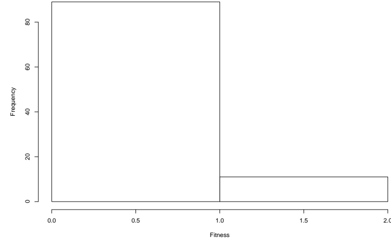

Fig. 4. Histogram for the instance $n = 50$, $m = 375$, $s = 1720$: $NCH_{val} = 0$; $IR_{mean} = 0,56$; $freq[0 \dots 1] \sim 90$.

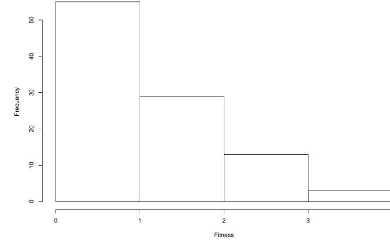

Fig. 5. Histogram for the instance $n = 100$, $m = 750$, $s = 5885$: $NCH_{val} = 1$; $IR_{mean} = 1,5$; $freq[0 \dots 1] \sim 55$.

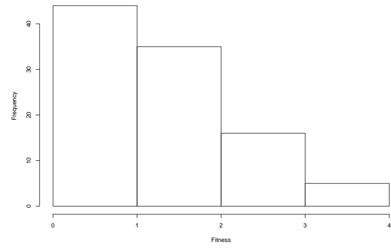

Fig. 6. Histogram for the instance $n = 150$, $m = 1688$, $s = 3738$: $NCH_{val} = 1$; $IR_{mean} = 1,69$; $freq[0 \dots 1] \sim 45$.

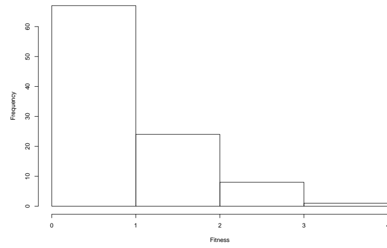

Fig. 7. Histogram for the instance $n = 300$, $m = 6750$, $s = 4889$: $NCH_{val} = 1$; $IR_{mean} = 1,27$; $freq[0 \dots 1] \sim 65$.

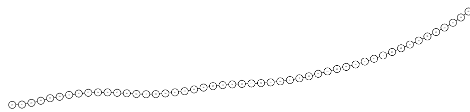

Fig. 8. IR solution for the inst. $n = 50$, $m = 186$, $s = 7085$, $branch = 0$.

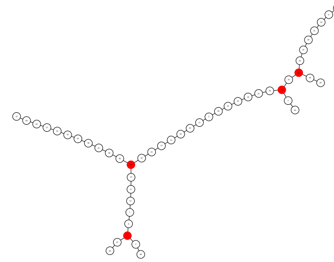

Fig. 9. NCH solution for the inst. $n = 50$, $m = 186$, $s = 7085$, $branch = 4$.

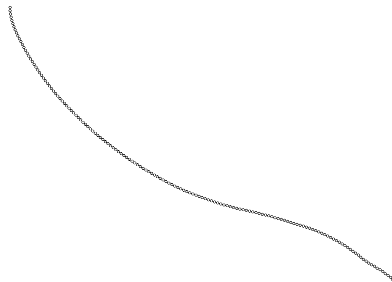

Fig. 10. IR solution for the inst. $n = 150$, $m = 1688$, $s = 5011$, $branch = 0$.

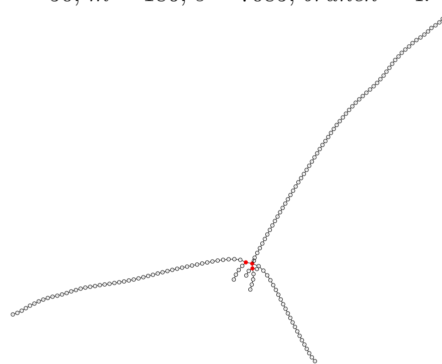

Fig. 11. NCH solution for the inst. $n = 150$, $m = 1688$, $s = 5011$, $branch = 3$.