Implementation of the Iterative Relaxation Algorithm for the Minimum Bounded-Degree Spanning Tree Problem [⋆]

Attila Bernáth, Krzysztof Ciebiera, Piotr Godlewski, and Piotr Sankowski

Institute of Informatics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
{athos,ciebie,pgodlewski,sank}@mimuw.edu.pl

Abstract. In the Minimum Bounded-Degree Spanning Tree Problem we want to find a minimum cost spanning tree that satisfies given degree bounds. For this problem a very good quality solution can be found using the iterative relaxation technique of Singh and Lau STOC'07: the cost will not be worse than the cost of the optimal solution, and the degree bounds will be violated by at most one. This paper reports on the experimental comparison of this state-of-art approximation algorithm with standard, although well-tuned meta-heuristics. We have implemented the Iterative Relaxation algorithm of Singh and Lau and speeded it up using several heuristics including row generation and combinatorial LP pivoting. On the other hand, as the heuristic point of reference we have chosen local search techniques in a Simulated Annealing framework, where we allow the violation of degree bounds by one. In such setting there are two natural objectives for comparison: the cost of the solution, and the number of violated degree bounds. If we keep the number of violated constraints fixed in both algorithms then Iterative Rounding usually outperforms Simulated Annealing by several percents.

1 Introduction

This paper is a report on the implementation and experimental verification of the Iterative Relaxation method for the Minimum Bounded-Degree Spanning Tree Problem (MBDSPT Problem). In this problem we are given an undirected graph $G = (V, E)$, costs for the edges $c : E \to \mathbb{R}_+$ and degree bounds $b : V \to \mathbb{Z}_+$. We are looking for the spanning tree T of $G = (V, E)$ that minimizes the total cost $c(T) = \sum_{t \in T} c(t)$ and satisfies degree constrains given by b: namely the number $d_T(v)$ of tree edges incident to a node v should not exceed $b(v)$ at any v. This problem has many applications in situations where we want to connect nodes, but we have physical constraints on the number of connections (wires) we can attach to a node. In particular, this problem arised for the first time in the context of backplane wiring among pins where no more than a fixed number of wire-ends could be wrapped around any pin on the wiring panel [4]. Similar problem is encountered in VLSI design, where the limit on the number of

⋆ Research was supported by the ERC StG project PAAl no. 259515.

transistors that can be driven by the output current of a transistor is the degree bound for VLSI routing trees [2]. Another type of applications is the design of a reliable communication network, since the maximum degree in a spanning tree is a measure of vulnerability to single-point failures [14].

Observe that setting all the degree bounds to 2 gives a reformulation of the Minimum Cost Hamiltonian Path Problem. The problem is hopeless to approximate, unless the cost function satisfies some additional property, like triangle inequality (the inapproximability can be generalized to degree bounds larger than two [14]). Hence, two questions naturally arise. First is theoretical: can we propose any theoretically good solution? The second is practical: would such solution be of practical importance, or would meta-heuristic approaches outperform it? There is vast amount of work that try to answer both questions.

From theoretical point of view, after taking into account the mentioned hardness results, the best possible solution has been found in [15] – there is no penalty in cost but some of the degree bounds can be violated. There was a long line of research of this type [14,6,7], ending with the ultimate paper by Singh and Lau [15] that shows that we can find a spanning tree T with cost no more than the cost of the optimal solution of the original problem, and the price we pay is that some of the degree bounds might be violated by one, that is $d_T(v)$ might be $b(v) + 1$ for some nodes. The technique used is the so-called Iterative Relaxation Technique.[1] On a high level the algorithm works as follows: we formulate a natural LP relaxation of the problem; we find a basic optimal solution of this relaxation; we remove edges with zero value in the LP solution; and we remove a degree bound at some node v where the number of remaining edges is not more than $b(v) + 1$. We iterate the above procedure till the solution found is integral. The main technical ingredient of the method is to prove that properties of basic solutions assure that there always exists a removable degree bound.

On the other hand, the literature on experimental solution to the MBDSPT Problem is rather vast. The first implemented algorithm seems to be the branch-and-bound one given by Narula and Ho [12]. Since then almost every metaheuristic framework (e.g., ant-colony, simulated annealing, or genetic algorithms) has been implemented and tested for this problem. In this paper, we concentrate mostly on the implementation issues of the Iterative Relaxation (IR) Technique, and due to space limitation of this submission we refer the reader for the review on the existing work to the two recent papers on local search techniques [16] and Lagrangian relaxation [1].

We have implemented the IR Algorithm of Lau and Singh [15] and optimized it so that it can solve instances of reasonable size. There are several non-trivial issues here that need to be solved. The main issue is that the natural LP relaxation contains an exponential number of constraints (there exists a polynomial sized version of this LP, but it is still too large for LP solvers). We used a row generation technique to overcome this problem, and we tried several heuristics to speed it up. We also implemented a version that solves the LP relaxation only once, and for subsequent iterations it uses a "combinatorial pivoting" subroutine.

[1] This technique will be explained in more detail in Section 2.

The implemented IR algorithm is compared with some local search techniques in a simulated annealing (SA) framework. We make both algorithms work with the same assumptions, i.e., we allow the SA algorithm to violate every degree bound by one. We note that our SA algorithm is well tuned to the problem and is able to compete with state-of-art techniques [9,16,1]. Actually, both IR and SA perform much better than solutions implemented in [9,16] and in many cases deliver optimal solutions. You should however, keep in mind that a fair comparison here is hard, as there are two candidate objectives to consider when comparing existing approaches. First, the most natural one is the cost. Second, on the other extreme is the number of violated constraints (e.g., when we are looking for a Hamiltonian path in an unweighted graph, then we don't care about the cost, but we do care about the number of violated constraints). The problem in this comparison is that standard IR does not allow to control the number of violated constraints. Although it is allowed to violate all constraints by one, it actually produces solutions that violate many fewer constraints then SA. Hence, fair comparison of both method requires to develop a method that allows to control the number of violations in IR. We achieve this by rounding different number of variables in each round of the IR. On the other hand, in SA one can control the number of violated constraints by simply adding this requirement to the objective function. Equipped with this methods we observe experimentally that IR performs visibly better then SA when few violations are allowed. In such case the solution cost is lower by 10-20%. When we allow more violations the difference between the methods becomes smaller, but generally IR slightly outperforms SA.

We close the section by introducing some notation. For a graph $G = (V, E)$ and a subset $S \subseteq V$ we write $\delta_G(S)$ for the set of edges with exactly one endnode in S, and $I_G(S)$ for the set of edges with both endnodes in S (we note that this notation differs from the one used in [15]). Let furthermore $d_G(S) = |\delta_G(S)|$, and we will apply the notation $\delta_F(S)$ and $d_F(S)$ as well even if $F \subseteq E$ is only some subset of edges. We will omit the subscript and write $I(S)$, $\delta(S)$, etc. when it causes no confusion. We furthermore simplify $d_G(\{v\})$ to $d_G(v)$. For a vector $x : E \to \mathbb{R}$ and a set $F \subseteq E$ we let $x(F) = \sum_{f \in F} x(f)$. In a directed graph $D = (V, A)$ let $\delta^{in}(S)$ ($\delta^{out}(S)$) denote the set of arcs entering (leaving, resp.) a set of nodes S. An **arborescence** is a directed spanning tree in which every node is entered by at most 1 arc (that is, a spanning tree oriented out of some node r). For nodes $s, t \in V$, a subset S with $s \in S \subseteq V - t$ is called an **$s\bar{t}$-set**.

2 The Iterative Relaxation Algorithm

In this section we briefly recall the Iterative Relaxation Algorithm of Singh and Lau [15] for the Minimum Bounded-Degree Spanning Tree Problem. A nice account of this method (with many other similar iterative algorithms) can be found in the book of Lau, Ravi and Singh [10]. Let us start with some preliminaries.

Theorem 1 (Edmonds [5]). *Given a graph $G = (V, E)$, the convex hull of incidence vectors of spanning trees is described by the following system of linear inequalities.*

$$x \in \mathbb{R}^E, x \geq 0, \tag{1}$$
$$x(I(S)) \leq |S| - 1 \textit{ for any non-empty } S \subsetneq V, \tag{2}$$
$$x(E) = |V| - 1. \tag{3}$$

Consider the following IP formulation of the MBDSPT Problem: we want to find an integer vector $x \in \mathbb{Z}^E$ satisfying (1)-(3) and $x(\delta(v)) \leq b(v)$ for every $v \in V$, and we want to minimize $\sum_{e \in E} c(e)x(e)$. If we drop the integrality constraints, we obtain an LP relaxation called **the Subtour LP**.

The Subtour LP: Assume that we are given an MBDSPT Problem with graph $G = (V, E)$, edge costs $c : E \to \mathbb{R}_+$, and degree bounds $b : V \to \mathbb{Z}_+$. Let us introduce the following polyhedron for some $W \subseteq V$ and $F \subseteq E$ (W is the set of nodes where the degree bound is not yet removed, while F is the subset of edges not yet fixed to zero).

$$\begin{aligned}
SubP(W,F) = \{x \in \mathbb{R}^E : x \geq 0, & \qquad (4)\\
x(I(S)) \leq |S| - 1 \text{ for any non-empty } S \subsetneq V, & \qquad (5)\\
x(E) = |V| - 1, & \qquad (6)\\
x(\delta(v)) \leq b(v) \text{ for every } v \in W, & \qquad (7)\\
x(e) = 0 \text{ for } e \notin F\}. & \qquad (8)
\end{aligned}$$

Note that $x(e) \leq 1$ is implied by the inequalities (5) above for any edge $e = uv \in E$ (take $S = \{u, v\}$). Note that we could get rid of the dependence on F by simply deleting the edges of $E - F$ as in [10]: we decided not to do so, because this is closer to our implementation (however, we will often ignore the reference to F and simply write $SubP(W)$ instead of $SubP(W, F)$). Given a vector $x \in SubP(W, F)$, a constraint in the system above that holds with equality for x will be called **x-tight** (or simply **tight**, if no confusion can arise). The constraints of form (5) and (6) will be called **set constraints**; those of form (7) will be called **degree constraints**, whereas the first ones (4) are called **non-negativity constraints**. Let $SubP = SubP(\emptyset, E)$ be the polyhedron without the degree bounds (that is, the convex hull of spanning trees by Theorem 1), and let $SubP_b = SubP(V, E)$ be the polyhedron with all the degree constraints.

The linear program below will be called the **Subtour LP** and will be denoted by $SubLP(W, F)$, where $W \subseteq V$ and $F \subseteq E$ (we will often ignore the dependence on F and write $SubLP(W)$ only).

$$\min\{\sum_{e \in E} c(e)x(e) : \ x \in SubP(W, F)\}$$

Clearly, the optimum value of $SubLP(V, E)$ is a lower bound on the optimum of the MBDSPT Problem. The following theorem gives a useful property of extreme-point solutions of this LP.

Theorem 2 (Singh and Lau, [15]). *Assume that x^* is an extreme-point solution of $SubLP(W, F)$ and let $E^* = \{e \in F : x^*(e) > 0\} \subseteq F$ be the support of x^*. Then there exists at least one node $v \in W$ such that $d_{E^*}(v) \leq b(v) + 1$.*

Such a degree bound is called **removable**. The IR Algorithm makes use of this fact by iteratively fixing more and more edges to zero, and relaxing the degree bounds that become removable.

2.1 The Algorithm

The Iterative Relaxation Algorithm of Singh and Lau [15] for the MBDSPT Problem is as follows.

```
Algorithm IR_MBDSPT($G, c, b$)
begin
     INPUT: A graph $G = (V, E)$, edge costs $c$, and degree bounds $b(v)$ for every $v \in V$.
     OUTPUT: A spanning tree $T$ of $G$ with cost at most the optimum value of $SubLP(V, E)$ and
     satisfying $d_T(v) \leq b(v) + 1$ at every node $v$.
1.1. Initialize $W$ to $V$ and $F$ to $E$.
1.2. SolveLP: Find an optimal extreme-point solution $x^*$ of $SubLP(W, F)$.
1.3. While $x^*$ is not an integer vector
1.4.      Let $E^* = \{e \in F : x^*(e) > 0\}$ be the support of $x^*$.
1.5.      RelaxLP: Let $W'$ be $W$ minus some of the nodes $v \in W$ with $d_{E^*}(v) \leq b(v) + 1$.
1.6.      ResolveLP: Find an extreme-point solution $x'$ of $SubLP(W', E^*)$ satisfying $c^T x' \leq$
          $c^T x^*$ (such solution exists, since $x^*$ is feasible to $SubLP(W', E^*)$).
1.7.      Set $W = W'$, $F = E^*$, and $x^* = x'$.
1.8. End while
1.9. Return $x^*$.
end
```

Note that in the Relaxation Step we can remove an arbitrary subset of removable degree constraints, but we have to relax at least one of them in order to make progress (i.e., $x' = x^*$ otherwise). We used this observation for finding different solutions for single instances: see details in Section 4.2. Note also that the Resolve Step can simply resolve the LP, but the algorithm also works if x' is not an optimal solution of $SubLP(W', E^*)$. This can be used to speed up the algorithm.However, choosing a suboptimal x' can degrade the quality ($\sim$ cost) of the final solution returned by the algorithm.

The main technical difficulty is Step 1.2: finding an optimal extreme point solution of an LP with exponentially many constraints. We tried many approaches to handle this issue. One possibility is using a polynomial sized LP instead, e.g., the Extended Bidirected LP. However, this LP is still too large to allow for efficient solutions. Instead we used row generation method together with exponential size Subtour LP. Algorithm obtained this way is not guaranteed to run in polynomial time, but performs very well in practice. We have tested the following implementation variants of Algorithm IR_MBDSPT

1. Find_Most_Violated: variant with row generation for the Subtour LP done by choosing the most violated constraint,
2. Find_First_Violated: variant with row generation for the Subtour LP done by choosing the first violated constraint from a random source,

3. Combinatorial_Pivoting: variant with row generation for the Subtour LP done by choosing the first violated constraint from a random source and solving the LP only once using the combinatorial pivoting.

Let t_1, t_2 and t_3 be the average running times (on our benchmark instances) of the respective variants. Row generation (finding violated constraints) for the Subtour LP was the bottleneck of Find_Most_Violated. Changing the row generation method in Find_First_Violated improved the average running time considerably: $t_2 \approx 0.1 \cdot t_1$.

Solving the LP for the first time (Step 1.2 of Algorithm IR_MBDSPT) was the bottleneck of Find_First_Violated. 85% of running time t_2 was spent on that step, and only the the remaining 15% was needed for the loop in Steps 1.3 - 1.8. The third variant: Combinatorial_Pivoting improved the running time of this loop by 60%. It did not, however, have a significant effect on the total running time of the algorithm, as this loop was not the bottleneck of the implementation (that is, the average running time of Combinatorial_Pivoting became $t_3 \approx 0.85 \cdot t_2 + 0.4 \cdot 0.15 \cdot t_2 \approx 0.9 \cdot t_2$).

Combinatorial_Pivoting did however decrease the number of violated degree bounds by a factor of up to 10, while only increasing the solution cost by 2%. This fastest version of the Iterative Relaxation Algorithm solved the largest problems in our benchmarks (cliques of 700 nodes) in approximately 15 minutes.

3 Local Search

We have tried a few approaches to local search and we come to the conclusions that Simulated Annealing [8] gives the best results in our case. We limit the running time of our Local Search to 10s: we have observed that there is no substantial improvement in the cost of the solution after this time limit. Our local search algorithm works as follows. An initial spanning tree T of G is selected. We used a BFS tree rooted at a random node, or a Minimum Cost Spanning Tree. However, this initial choice has no impact on the obtained solution. Then we try to improve T by finding two edges $e_1 \in T$ and $e_2 \in G - T$ such, that $T_{new} = T - e_1 + e_2$ is a spanning tree of G. The penalty of the tree T is defined as follows.

$$P(T) = \sum_{e \in T} c(e) + c_{max} \cdot \sum_{v \in V} \max(d_T(v) - b(v) - 1, 0), \tag{9}$$

where c_{max} is the maximum cost of edge in G. In other words the penalty is equal to the sum of costs of edges in T plus the penalty for each unsatisfied degree bounds. For every unsatisfied degree bound we add penalty as big as the biggest edge weight times the height of the violation.

The standard hill climbing method tries to find T_{new} with lower penalty than T, whereas in the SA method the penalty can increase with the following probability

$$p(T, T_{new}) = \exp\left(-\frac{P(T_{new}) - P(T)}{t}\right),$$

where t is "temperature" of the process that continually decreases during the execution of the algorithm. We use exponential cooling scheme where the temperature of the next step is equal to αt for a constant $\alpha < 1$.

We have tried two methods of finding T_{new}. They both start by selecting a random non-tree edge $e_1 \in G - E(T)$ and finding a fundamental cycle C in $T + e_1$. The first one chooses e_2 to be a random edge in $C - e_1$ whereas the second one sets e_2 to be the edge on the cycle C that decreases the penalty to the largest extent. In the worst case, this selection takes $O(d)$ time, where d is the diameter of the tree T which is upper bounded by $|V|$. In our benchmarks this implementation ran fast enough, so that the SA process always finished in 10 seconds, and longer cooling schemes did not give better quality solutions.

We have run experiments on both hill climbing and simulated annealing. The latter works visibly better when the degree bounds are small. Moreover, we have tried different penalties for violations of degree bounds and we have found out that setting penalties as in (9) gave the best results. In Figure 1 the results obtained using three implementations of the local search technique are compared using TSPLIB data

– the hill climbing (HC),
– the simulated annealing with random choice of edges e_1 and e_2 (SA 1),
– the simulated annealing with random choice of edge e_1 and respective best choice of edge e_2 (SA 2). Later comparisons will compare SA 2 with IR., since SA 2 gave the best results of three tested algorithms.

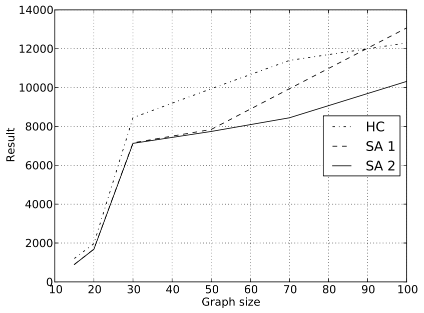

Fig. 1. Comparison of different implementations of local search techniques. Graph size is the number of vertices. The y axis shows the cost of a DBST found by hill-climbing (HC) and two variants of Simulated Annealing (SA 1, SA 2).

4 Experimental Results

We implemented the algorithms described above in C++ (gcc-4.6). As a part of this project a generic framework for implementing IR methods has been developed and became a part of the PAAl library [13]. We also used the LEMON library [11] that proved especially useful for its interface towards LP solvers – we used the CPLEX LP solver [3]. All of our programs were compiled with `-O3` optimization option and run on Intel Xeon CPU E5649@2.53GHz machine.

4.1 Test Cases

We tested the algorithms on the following data sets.

TSPLIB Instances: We have taken symmetric TSP instances from the TSPLIB library. We interpreted these as MBDSPT Problem instances by setting the degree bounds identically to 2.

Instances From [9]: We compared the results provided by the algorithms with those found by the heuristics of [9] in terms of the cost function. Using SA we obtained better quality results, e.g., in 16 out of 31 cases we improved heuristically obtained solutions of [9]. On the other hand, in all tests with degree bounds 3, 4 or 5 (except tests *str1009* and *sym704* with degree bound = 3), IR found an optimal integral solution for the initial LP – without violating any degree bounds. For 39 out of 118 tests with degree bounds = 2 the IR also found an optimal (integral) solution for the initial LP.

Instances From [16]: We also compared the results of our algorithms on instances generated using the algorithm described in [16]. For all of the instances generated that way the IR found an optimal (integral) solution for the initial LP. Our SA implementation with limiting of the allowed number of constraint violations to 0 also found results with the same costs as the IR.

Random Generators: We used four random generators. In all generators we generate connected graphs with a given number of nodes and edges, we assign edge costs and degree bounds so that the problem is feasible. Let $U(N)$ denote a random integer variable uniformly distributed in the range $[1, N]$. The generators produce the following types of graphs:

- Generator 1: we generate a random spanning tree T and set the edge costs independently from $U(200)$ (the random spanning tree is generated as follows: add the nodes one by one and connect the new node with a randomly chosen previous one). Then we set the degree bounds to be $d_T(v)$, so that T becomes a feasible solution and $c(T)$ is an upper bound on the cost of an optimal feasible solution. Then we add more edges between nodes not connected so far, with costs from $100 + U(200)$ (thus we don't destroy the solution T very much). Our observations show that the randomly generated tree T rarely has nodes with high degrees.
- Generator 2 uses the ideas from [2]: we generate a graph that has a unique Minimum Cost Spanning Tree (MST), but this tree has high vertex degrees. We set the degree bounds so that the MST is not a feasible solution, but we make sure that there exists a feasible solution. This is achived in the following way. First we generate a random spanning tree $T = (V, E)$ (as in the previous generator), with costs from $100 + U(100)$, and we set the degree bounds to be $d_T(v)$ for every v (again $c(T)$ is an upper bound on the cost of an optimal feasible solution). Then we form the MST by choosing node disjoint stars (each of size roughly $\sqrt{|V|}$), and connecting them in a random way to get a spanning tree. The costs of the edges in this spanning tree are

chosen from $U(100)$, therefore indeed this tree will be the unique MST (if we happen to use an edge created previously, we decrease its cost, so the cost of T might decrease). Finally we add more edges between nodes not connected so far, with costs from $100 + U(100)$.

- Generator 3: first we generate a Hamiltonian path with edge costs from $100 + U(100)$, and set the degree bounds to be equal 2 for every node. Next we add a unique (non-feasible) MST with high degrees, as detailed in the previous generator (edge costs from $U(100)$). Finally we add more edges between nodes not connected so far, and assign the edge costs from $100 + U(100)$.
- Generator 4: we generate a clique of k nodes. Edge costs are chosen from $100 + U(100)$ for all edges except for 3 arbitrary edges incident with an arbitrary node (we call this node the center of the clique), which have costs form $U(100)$. We repeat the described procedure k times to generate k independent graphs. Next we treat those graphs as nodes of a new clique and repeat the above procedure recursively. We add the new edges between the cliques so that each node that was not the center of a clique in the previous recursive step gets exactly one new edge to some other clique. We set the degree bounds to be equal 2 for every node (the generated instance has got a Hamiltonian path, but it also has a non-feasible MST using all 3 low-cost edges in some cliques).

4.2 Comparison of IR and SA

First we performed the vanilla test where both IR and SA are allowed to violate all constraints by one. The comparison of the two methods on TSPLIB instances

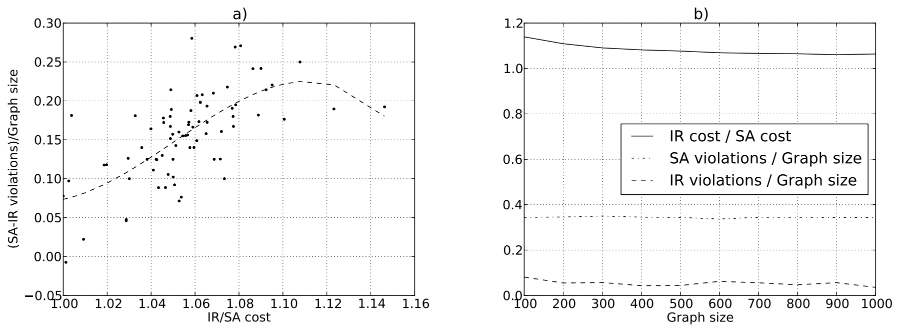

Fig. 2. Comparison of costs and numbers of violated constraints found by IR and SA: (a) for TSPLIB instances, each point represents a single instance, x axis shows cost ratio (SA always outperforms IR), y axis shows difference in the number of violations ratio (IR ouperforms SA in all but one cases) divided by graph size, dashed line is spline interpolation; (b) for random graphs, x axis shows the number of vertices; SA outperforms IR in terms of costs but it violates more constraints.

is shown in Figure 2 (a). Similarly, we have tested the algorithms on random graphs obtained from our 4 generators for sizes from 25 to 1100 nodes. The results for all four graph types are qualitatively the same – Figure 2 (b) gives results for the first generator. We found out that SA almost always found a lower cost spanning tree than IR on those four sets of graphs. On the other hand, IR violates many fewer degree constraints. These results do not give clear answer which method is better. Hence, in order to obtain more decisive results we decided to compare the methods when both of them are allowed the same number of violations. For this comparison we used the Find_First_Violated version of Algorithm IR_MBDSPT (as described in Section 2.1). While it was 10% slower than the fastest version, it had the advantage that in each iteration we could relax more than one constraint. Setting different upper limits on the number of constraints removed in each Relaxation Step of Algorithm IR_MBDSPT gave different trees as the result of the algorithm. Figures 3 and 4 illustrate the comparison of numbers of violated constraints and spanning tree costs in selected cases of random graphs and graphs from TSPLIB. This freedom allows as to compare both algorithms in case of when the number of violated constrains is fixed to be the same. We observed that in most of the cases IR finds solutions

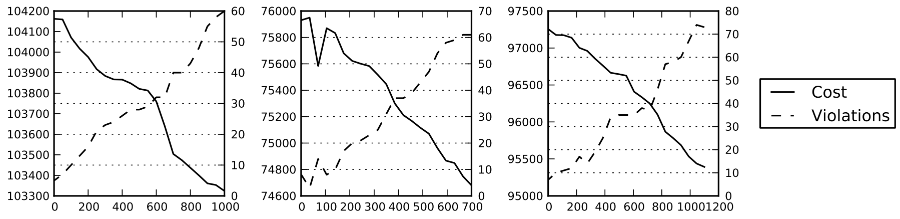

Fig. 3. Comparison of costs and number of violated constraints of three random graphs using IR setting different limits on number of removed constraints (x axis); the left y axis shows the solution cost and the right y axis the number of violations

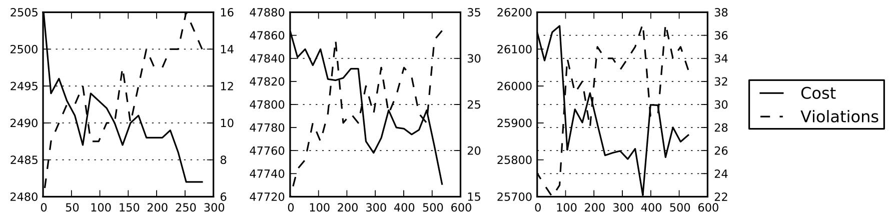

Fig. 4. Comparison of costs and number of violated constraints of three TSPLIB graphs using IR setting different limits on number of removed constraints (x axis); the left y axis shows the solution cost and the right y axis the number of violations

having lower cost than SA with the same number of violations. Figures 5 and 6 show results of such comparison using the same set of graphs as in Figures 3

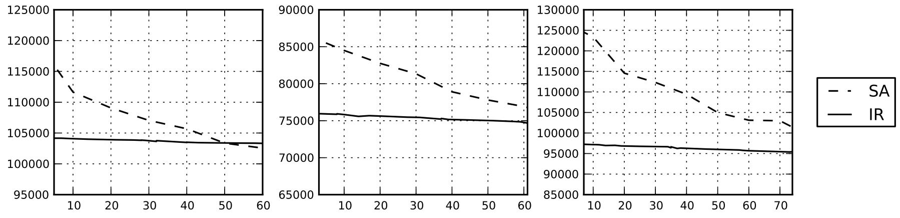

Fig. 5. Comparison of IR and SA algorithm on three random graphs. The x axis shows the number of violated constraints. The y axis shows the solution cost.

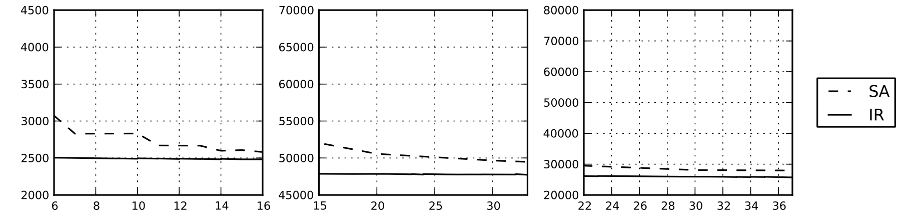

Fig. 6. Comparison of IR and SA algorithm on three TSPLIB graphs. The x axis shows the number of violated constraints. The y axis shows the solution cost.

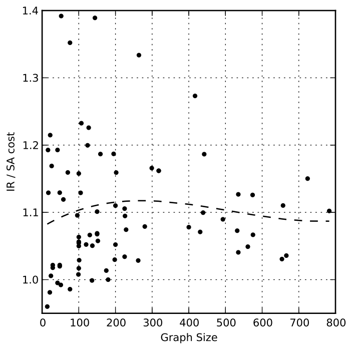

Fig. 7. Comparison of IR and SA algorithm on all TSPLIB graphs when the number of violated constrains in SA is fixed to be the same as violated by IR. The axis x shows number of vertices. The y axis shows ratio of costs. SA finds better solution only in case of small graphs. Dashed line is spline interpolation.

and 4. These figures demonstrate that when the number of violations is small the IR outperforms SA usually by a good 10%. On the other hand, when more violations are allowed this difference decreases, but is still visible in most of the cases. These results on TSPLIB instances are aggregated on Figure 7. It shows that SA was able to find only a few solutions that were better then the ones found by IR. This happened only when graph sizes were small – number of vertices less or equal to 136. The above results give clear evidence that IR delivers betters solutions than SA. We note that, although there is a method to control the number of violated constraints in IR, this method is far from perfect as it is visible on Figure 4. The results are not monotone and so in order to get the right number one needs to go through all the possibilities.

Acknowledgements. The authors would like to thank Tamás Király, Marcin Mucha, Mohit Singh, Łukasz Sznuk, Piotr Wygocki and the developers of the LEMON Library for their help.

References

1. Andrade, R., Lucena, A., Maculan, N.: Using Lagrangian dual information to generate degree constrained spanning trees. Discrete Appl. Math. 154(5), 703–717 (2006), `http://www.sciencedirect.com/science/article/pii/S0166218X0500301X`
2. Boldon, B., Deo, N., Kumar, N.: Minimum-weight degree-constrained spanning tree problem: Heuristics and implementation on an SIMD parallel machine. Parallel Comput. 22(3), 369–382 (1996)
3. CPLEX, I.I.: High performance mathematical programming engine, `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`
4. Deo, N., Hakimi, S.: The shortest generalized hamiltonian tree. In: Proceedings of the 6th Annual Allerton Conference, pp. 879–888. University of Illinois, Illinois (1968)
5. Edmonds, J.: Matroids and the greedy algorithm. Math. Program. 1(1), 127–136 (1971)
6. Furer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithm 17(3), 409–423 (1994)
7. Goemans, M.X.: Minimum bounded degree spanning trees. In: FOCS 2006, pp. 273–282. IEEE Computer Society, Los Alamitos (2006)
8. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
9. Krishnamoorthy, M., Ernst, A.T., Sharaiha, Y.M.: Comparison of algorithms for the degree constrained minimum spanning tree. J. Heuristics 7(6), 587–611 (2001)
10. Lau, L.C., Ravi, R., Singh, M.: Iterative methods in combinatorial optimization. Cambridge University Press, Cambridge (2011)
11. Library for Efficient Modeling and Optimization in Networks (LEMON), `http://lemon.cs.elte.hu`
12. Narula, S., Ho, C.: Degree-constrained minimum spanning tree. Comput. Oper. Res. 7, 239–249 (1980)

13. Practical Approximation Algorithms Library (PAAL), `http://paal.mimuw.edu.pl`
14. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt III, H.B.: Many birds with one stone: Multi-objective approximation algorithms. In: STOC 1993, pp. 438–447. ACM, New York (1993), `http://doi.acm.org/10.1145/167088.167209`
15. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: STOC 2007, pp. 661–670. ACM, New York (2007)
16. Zahrani, M.S., Loomes, M.J., Malcolm, J.A., Albrecht, A.A.: A local search heuristic for bounded-degree minimum spanning trees. Eng. Optimiz. 40(12), 1115–1135 (2008), `http://www.tandfonline.com/doi/abs/10.1080/03052150802317440`