A practical greedy approximation for the Directed Steiner Tree problem

Dimitri Watel[1,2] and Marc-Antoine Weisser[1]

[1] SUPELEC System Sciences, Computer Science Dpt., 91192 Gif sur Yvette, France
dimitri.watel@supelec.fr, marc-antoine.weisser@supelec.fr
[2] University of Versailles, 45 avenue des Etats-Unis, 78035, France

Abstract. The Directed Steiner Tree (DST) NP-hard problem asks, considering a directed weighted graph with n nodes and m arcs, a node r called *root* and a set of k nodes X called *terminals*, for a minimum cost directed tree rooted at r spanning X. The best known polynomial approximation ratio for DST is a $O(k^\varepsilon)$-approximation greedy algorithm. However, a much faster k-approximation, returning the shortest paths from r to X, is generally used in practice. We give in this paper a new $O(\sqrt{k})$-approximation greedy algorithm called Greedy$_{\text{FLAC}}{}^{\triangleright}$, derived from a new fast k-approximation algorithm called Greedy$_{\text{FLAC}}$ running in time at most $O(nmk^2)$.
We provide computational results to show that, Greedy$_{\text{FLAC}}$ rivals the running time of the fast k-approximation and returns solution with smaller cost in practice.
Keywords: Directed Steiner Tree, Approximation algorithm, Greedy algorithm

1 Introduction

The *Undirected Steiner Tree* problem asks, in an undirected weighted graph, for a minimum cost tree spanning a set of specific nodes called *terminals*. It is a classical NP-hard problem [1] and can be polynomially approximated with constant ratio [2–4]. We focus here on its directed extension, the *Directed Steiner Tree* problem (DST).

Problem 1. Given a *directed* graph $G = (V, A)$ with n nodes and m arcs, a node r called *root*, a set X of k nodes called *terminals* and non negative weights ω on the arcs, the Directed Steiner Tree problem (DST) consists in finding a minimum cost directed tree rooted at r spanning every terminal.

These two problems are known to have applications essentially in multicast routing where one wants to minimize bandwidth consumption [5–7]. DST is used when the model of a symmetric network is not sufficient.

Contrary to the undirected version, there is no polynomial approximation algorithm achieving a constant ratio for DST. Indeed, as a generalization of the NP-hard Set Cover problem [1], it is known to be inapproximable within a $O(\log(k))$ ratio unless $NP \subseteq DTIME[n^{O(\log\log n)}]$, where k is the number of

terminals [8]. It was later proved under the same assumption that for $\varepsilon > 0$ there is no $O(\log^{2-\varepsilon}(k))$ approximation algorithm [9].

Charikar *et al.* introduced a greedy algorithm [10, 11] which extends a $\log(k)$-approximation algorithm for Set Cover [12, 13]. At each step, it searches for a tree T of cost $\omega(T)$ covering only a part $X(T)$ of the terminals and minimizing the *density* $d(T) = \frac{\omega(T)}{|X(T)|}$. A small density is characteristic of a tree which makes a compromise between covering many terminals and reducing its cost. However, finding a minimum density Directed Steiner Tree is an NP-hard problem and has to be polynomially approximated too. Nonetheless, an efficient approximation for this problem implies a greedy algorithm with efficient ratio for DST [10]. This method gives the current best known approximation, CH_l, with ratio $2l^2(l-1)\sqrt[l]{k/2} = O(k^{\frac{1}{l}})$ for any $l > 1$ [10, 11].

In order to achieve the best approximation ratio with its algorithm, Charikar *et al.* do not work with the instance itself but with the shortest paths instance.

Definition 1. *Let $\mathcal{I} = (G, r, X, \omega)$ be a directed Steiner tree instance. We define as $\omega^{\triangleright}(u, v)$ the cost of a shortest path linking u to v, or $+\infty$ if such a path does not exist. The* shortest paths instance *$\mathcal{I}^{\triangleright} = (G^{\triangleright} = (V, A^{\triangleright}), r, X, \omega^{\triangleright})$ is a complete directed graph where each arc (u, v) is weighted by $\omega^{\triangleright}(u, v)$.*

From a feasible solution $T^{\triangleright}$ of $\mathcal{I}^{\triangleright}$, one can build a feasible solution T of $\mathcal{I}$ by selecting for each arc (u, v) in $T^{\triangleright}$ any shortest path linking u and v. The cost of T is smaller than the cost of $T^{\triangleright}$.

Considering an approximation algorithm $\mathcal{B}$ for DST, one can build an approximation algorithm $\mathcal{B}^{\triangleright}$ for DST using algorithm 1.

Algorithm 1 An approximation algorithm $\mathcal{B}^{\triangleright}$ for DST from an approximation algorithm $\mathcal{B}$ for DST

Require: A DST instance $\mathcal{I} = (G, r, X, \omega)$, an algorithm $\mathcal{B}$ for DST.
Ensure: A feasible solution for $\mathcal{I}$.
1: Build $\mathcal{I}^{\triangleright}$
2: $T^{\triangleright} \leftarrow \mathcal{B}(\mathcal{I}^{\triangleright})$
3: $T \leftarrow \bigcup_{(u,v)\in T^{\triangleright}}$ a shortest path from u to v in $\mathcal{I}$
4: **return** T

Generally, $\mathcal{B}^{\triangleright}$ achieves a smaller approximation ratio than $\mathcal{B}$. However, $\mathcal{B}^{\triangleright}$ is necessarily longer than $\mathcal{B}$, especially when the original graph is sparse.

If the shortest path instance is precomputed, the running time of CH_l is $O(n^l k^{2l})$ with n as the number of nodes. As l increases, this grows exponentially. Consequently, a much faster k-approximation is frequently used, specifically when the application needs an answer fast and is satisfied by any correct solution [14–16]. It returns the shortest paths from r to all terminals. In a network, it would be equivalent to replacing the multicast tree by k unicast paths.

In order to compare algorithms, a test set named *SteinLib* aggregates most of the generated Undirected Steiner Problem instances that can be found in literature [17], and is used to compare exact or approximation algorithms for this problem [18–20]. However, the DST instances are still under construction. As a consequence, comparing DST algorithms requires transforming undirected instances into directed ones. This is usually done by replacing each edge by two opposites arcs [21–23]. Nonetheless, this method seems too restrictive as this kind of graph does not reflect the variety of possible instances and applications.

Our results. We provide an heuristic to build solutions covering only a part of the terminals with small density, following the ideas given in [10] and use it to build a greedy algorithm called $\text{Greedy}_{\text{FLAC}}$ for DST.

Additionally, we build a second approximation $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ by applying $\text{Greedy}_{\text{FLAC}}$ to $\mathcal{I}^{\triangleright}$ as explained with Algorithm 1. Table 1 summarizes the two algorithms properties.

Algorithm	**Approximation ratio**	**Time complexity**
$\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$	$4\sqrt{2k}$	$O(n^3k^2)$
$\text{Greedy}_{\text{FLAC}}$	k	$O(nmk^2)$

Table 1: Approximation algorithms provided in this paper, with their approximation ratios and Time complexities.

We have built three test sets to compare our algorithms, partially derived from the SteinLib test set. We have compared $\text{Greedy}_{\text{FLAC}}$ and $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ with four other algorithms including two versions of the shortest path algorithms. It is highlighted that $\text{Greedy}_{\text{FLAC}}$ returns solutions with nearly same cost than $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ in almost all the cases, that $\text{Greedy}_{\text{FLAC}}$ returns solutions with smaller cost than the four other tested algorithms, and finally that the running time of $\text{Greedy}_{\text{FLAC}}$ rivals that of the fastest shortest path algorithm, surpassing every other algorithms including $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$.

This paper is organized in this way. In the next section, we introduce our algorithms and prove the two approximation ratios. Section 3 is dedicated firstly to improving the implementation of $\text{Greedy}_{\text{FLAC}}$. Finally, in Section 4, we present two new methods to obtain a directed graph from an undirected instance and an evaluation of performances of our algorithms.

2 The algorithm

2.1 The density

The *density* of a tree describes its ability to span the most terminals by paying the least.

Definition 2. *The* density $d(T)$ *of a tree* T *of cost* ω *spanning* $X' \subset X$ *is* $\frac{\omega}{|X'|}$.

From any algorithm $\mathcal{A}$ returning a directed tree spanning any subset X' of X, one can build a greedy algorithm Greedy$_{\mathcal{A}}$ for DST by applying $\mathcal{A}$ to cover some terminals, removing them from X and repeating until X is empty. Algorithm 2 describes this technique.

Algorithm 2 Greedy Algorithm Greedy$_{\mathcal{A}}$ from an algorithm $\mathcal{A}$

Require: A DST instance $\mathcal{I} = (G, r, X, \omega)$, an algorithm $\mathcal{A}$ returning a directed tree rooted in r covering a subset X' of X
Ensure: A feasible solution for $\mathcal{I}$.
1: $T = \emptyset$
2: **while** $X \neq \emptyset$ **do**
3: $\quad T_0 \leftarrow \mathcal{A}(G, r, X, \omega)$
4: $\quad T = T \cup T_0$
5: $\quad X = X \backslash (X \cap T_0)$
return T

The key idea, developed in [10], is that the best is the density of the trees returned by $\mathcal{A}$, the best is the approximation ratio of Greedy$_{\mathcal{A}}$.

In the next subsection, we provide an algorithm to find partial solutions with small density. The next subsection does not describe an algorithm but a thought experiment as it is easier this way to understand why and how our algorithm works to find a low density tree. We will then describe an implementation.

2.2 Finding partial solution with a low density

We define the thought experiment as if each arc were a pipe able to carry water. Each terminal is a source of water. It feeds each of its entering arcs with one litre of water per second. The volume of an arc, in litres, is equal to its cost. When an arc is completely full, it is *saturated*. Each terminal sends one litre of flow per second to each arc it can reach with a path of saturated arcs. Conversely, the flow rate, in litres per second, inside an arc, is the number of terminals that arc is linked to with paths of saturated arcs. More precisely, if a terminal reaches 3 arcs with paths of saturated arcs, each arc is filled with 1 litre per second, and not $\frac{1}{3}$ litre per second.

Definition 3. *If a terminal reaches an arc* a *with two or more distinct paths of saturated arcs or with a path containing a cycle, we say the flow is* degenerate.

At time 0, no flow has come out from the terminals. We increase the flow until the root is reached, we then focus on the subgraph made of saturated arcs linking the root to terminals. If multiple arcs saturates at the same time, we arbitrarily order them. Each time the flow is degenerate, we remove the last saturated arc to cancel the degeneration, so that we assure the flow is never degenerate when the root is reached.

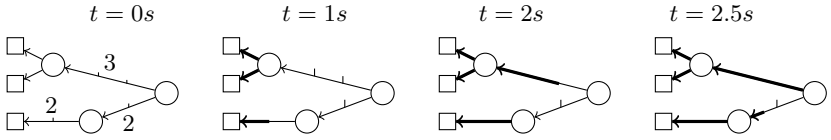

Fig. 1: This example illustrates the flow growing in a tree. Unspecified arc costs are 1. Water is represented by bold arc parts. Note that, at the last step, because the arc is fed with 2 terminals, it takes $0.5s$ to saturate it, and the arc below is only 25% full.

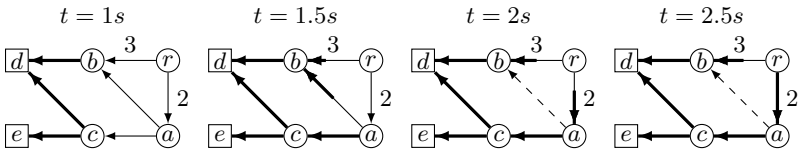

Fig. 2: This example illustrates the flow increasing in a graph. Unspecified arc costs are 1. Note that when (a, b) is saturated at $t = 2s$, the flow is degenerate. We remove (a, b) from the graph and the (r, a) flow rate remains 2. Note also that, at $t = 0s$, the two entering arcs of the terminal d are fed with 1 L/s each and not $\frac{1}{2}$.

An example in a tree is shown in Figure 1, and in a directed graph in Figure 2.

A subgraph with low density emerges, because the time that the root is reached depends on the number of terminals joined to accelerate the flow and on the existence of paths of low cost. Those two properties are strongly associated with a low density. As shown in Figure 1, the flow does not immediately describe a directed Steiner tree but a partial solution spanning only the first terminals reaching the root: we will repeat our algorithm to build a feasible solution.

Remark 1. This flow is not a conservative flow as in the maximum flow problem. Actually, we describe a *dual ascent* heuristic like those described in [21, 24, 25] where primal variables x_a for $a \in A$, between 0 and 1, include a in the feasible solution if $x_a = 1$. This flow describes how the primal variables evolve: x_a is the ratio of flow inside a divided by $\omega(a)$.

2.3 Naïve implementation

We provide here an algorithm implementing the experiment. This implementation does not guarantee the running time given in Section 1 but guarantees the approximation ratio. A faster implementation is given in Section 3.

Let $\mathcal{I} = (G = (V, A), r, X, \omega)$ be an instance of DST. To describe the implementation, we will use the following useful notations and definitions.

The quantity of flow inside the arc a is $f(a)$. We define an auxiliary graph $G_{SAT} = (V, A_{SAT} \subset A)$. An arc is said to be *saturated* if G_{SAT} contains it. The number of terminals that a node v or an arc $a = (u, v)$ is connected to in G_{SAT} is $k(v) = k(a)$. We define the current elapsed time as t and the remaining time before a is saturated by $t(a) = \frac{\omega(a) - f(a)}{k(a)}$. If $k(a) = 0$, $t(a)$ is infinite.

Finally, the flow is said to be *degenerate* if G_{SAT} contains a cycle or two distinct paths from any node to any terminal. To ensure the algorithm returns a tree, any time the flow is degenerate, we remove the last saturated arc from G_{SAT} and add it to the set $\mathcal{M}$ of *marked*. No flow goes through a marked arc.

We use Algorithm 3, called FLAC, to increase the flow in a graph. As explained previously, FLAC does not return a feasible solution for DST but only a partial solution. An example is given in Table 2.

Algorithm 3 FLow Algorithm Computation (FLAC)

Require: A DST instance $\mathcal{I} = (G, r, X, \omega)$.
Ensure: A directed tree rooted in r covering some nodes in X.
1: $G_{SAT} \leftarrow (V, \emptyset)$, $t \leftarrow 0$, $\mathcal{M} \leftarrow \emptyset$ # Marked arcs.
2: **for** $a \in A$ **do** $f(a) = 0$
3: **while** True **do**
4: **for** $(a \in A \backslash (\mathcal{M} \cup G_{SAT}))$ **do** $t(a) \leftarrow (\omega(a) - f(a))/k(a)$
5: $(u, v) \leftarrow \arg\min(t(a), a \in A \backslash (\mathcal{M} \cup G_{SAT}))$
6: **for** $(a \in A \backslash (\mathcal{M} \cup G_{SAT}))$ **do** $f(a) \leftarrow f(a) + k(a) \cdot t(u, v)$
7: $t \leftarrow t + t(u, v)$
8: $G_{SAT} \leftarrow G_{SAT} \cup (u, v)$
9: **if** $(u = r)$ **return** the tree T_0 in G_{SAT} linking r to the terminals
10: **if** the flow is degenerate **then**
11: $\mathcal{M} \leftarrow \mathcal{M} \cup (u, v)$
12: Remove (u, v) from G_{SAT}

At Line 9, there is only one tree T_0 in G_{SAT} linking the root to the terminals, because of the arcs added to $\mathcal{M}$ each time the flow is degenerated. One can build it with breadth-first search from the root to the terminals.

Iteration	(u, v)	t	$\mathcal{M}$	$t(b, d)$	$t(c, d)$	$t(c, e)$	$t(a, c)$	$t(a, b)$	$t(r, a)$	$t(r, b)$
-	-	0	$\emptyset$	1	1	1	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	(b, d)	1	$\emptyset$	0	0	0	$+\infty$	1	$+\infty$	3
2	(c, d)	1	$\emptyset$	-	0	0	1	1	$+\infty$	3
3	(c, e)	1	$\emptyset$	-	-	0	0.5	1	$+\infty$	3
4	(a, c)	1.5	$\emptyset$	-	-	-	0	0.5	1	2.5
5	(a, b)	2	$\{(a, b)\}$	-	-	-	-	0	0.5	2
6	(r, a)	2.5	$\{(a, b)\}$	-	-	-	-	-	0	1.5

Table 2: This example illustrates how Algorithm 3 runs on the graph shown in Figure 2, and how it differs from the experiment. It details the arc saturated at the beginning of each iteration, with the content of $\mathcal{M}$, the values of t and $t(a)$ for each arc a.

We can now build the two approximation algorithms for DST from FLAC. The $\text{Greedy}_{\text{FLAC}}$ algorithm is build using Algorithm 2 with FLAC. We build the $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ algorithm using Algorithm 1 with $\text{Greedy}_{\text{FLAC}}$. We now prove $\text{Greedy}_{\text{FLAC}}$ is a k-approximation and $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ is an $4\sqrt{2k}$-approximation.

2.4 Approximation Ratio of Greedy$_{\mathbf{FLAC}}$

Assuming we start the j-th iteration of the loop, let $a = (w, x)$ be an arc, we define $f_j(a)$ as the flow inside a, $k_j(x)$ as the value of $k(x)$, $t_j(a) = \frac{\omega(a)-f_j(a)}{k_j(x)}$ as the value of $t(a)$, and finally t_j as the value of t.

Firstly, because $k_i(a)$ can only increase as i grows, with Lines 6 and 7, we can prove the following lemma:

Lemma 1. *If during the $(l-1)$-th iteration, $a \notin G_{SAT} \cup \mathcal{M}$, then for each $j \leq l-1$, $\omega(a) \geq f_l(a) \geq f_j(a) + (t_l - t_j) \cdot k_j(a)$.*

Lemma 2. *If $t_l > t_j + t_j(a)$ for some $j \leq l-1$, then a was saturated or marked during or before the $(l-1)$-th iteration.*

Proof. We assume a was neither saturated nor marked during or before the $(l-1)$-th iteration. By Lemma 1, $\omega(a) \geq f_l(a) \geq f_j(a) + (t_l - t_j)k_j(a) > f_j(a) + t_j(a)k_j(a) = \omega(a)$, which is a contradiction. □

Let $f-1$ be the final iteration when T_0 is returned at Line 9 (t has then value t_f). Let x be the closest terminal to the root r, and P a shortest path linking r to x in $\mathcal{I}$.

Lemma 3. *t_f is no more than the cost of P.*

Proof. Let $v_l = r$, v_{l-1}, …, v_1, $v_0 = x$ be the successive nodes of P and ω_i the cost of the subpath of P from v_i to v_0. By induction on i we prove that either $t_f \leq \omega_i$ or there is an iteration j_i with $k_{j_i}(v_{i+1}, v_i) \leq 1$, and $t_{j_i} \leq \omega_i$.

If $i = 0$, with $j_0 = 1$, the property is true.

If the property is true for some $i-1$, and if $t_f > \omega_i \geq \omega_{i-1}$, by induction, there is an iteration j_{i-1} with $t_{j_{i-1}} \leq \omega_{i-1}$ and $k_{j_{i-1}}(v_i, v_{i-1}) \leq 1$. Consequently, $t_{j_{i-1}}(v_i, v_{i-1}) = (\omega_i - \omega_{i-1} - f_{j_{i-1}}(a))/k_{j_{i-1}}(v_{i-1}) \leq \omega_i - \omega_{i-1}$.

Then, by Lemma 2 and because $t_f > \omega_i$, there is an iteration $j_i < f$ where (v_i, v_{i-1}) is saturated or marked, then $k_{j_i}(v_{i+1}, v_i) \leq 1$, and $t_{j_i} \leq t_{j_{i-1}} + t_{j_{i-1}}(v_i, v_{i-1}) = \omega_i$. By induction on i, $t_f \leq \omega_l$ and the lemma is proved. □

Lemma 4. *t_f is the density of the tree T_0 returned at line 9.*

Proof. We first recall that T_0 is fully saturated by definition, no arc of T_0 is marked, and every arc entering T_0 is either marked in G_{SAT} or not, then there is no iteration during which an arc not included in T_0 sends flow into T_0.

For a node $v \in T_0$, let T_v, k_v, ω_v be the subtree rooted in v, the number of terminals and the cost of T_v. Let t_v be the value of t when the last arc of T_v is saturated. Finally, let ε_v be the flow that went through v at that iteration. One can see ε_v as the quantity of flow inside a fictive arc entering v with infinite cost when the last arc of T_v is saturated.

We prove hereinafter by induction on their height that for every node v, $t_v \cdot k_v = \omega_v + \varepsilon_v$. As no flow came through r before the f-th iteration, $t_r \cdot k_r = \omega_r$ and the lemma is proved.

If v is a leaf, T_v is empty and is then fully saturated at the first iteration when $t = 0$, then the property is true. If now the property is true for every node of height h, we prove it for a node v of height $h + 1$.

Let v_1, v_2, ... v_s be the children of v with $\omega(v, v_i) = \alpha_i$. The inductive hypothesis claims that for each $i \in [\![1; s]\!]$, $t_{v_i} \cdot k_{v_i} = \omega_{v_i} + \varepsilon_{v_i}$.

Let t'_{v_i} be the time when every arc of $(v, v_i) \cup T_{v_i}$ is saturated. Note that (v, v_i) can be saturated before the last saturated arc of T_{v_i}, depending on the value of ε_{v_i}. Firstly, if $\alpha_i \leq \varepsilon_{v_i}$, $t'_{v_i} = t_{v_i}$. In that case, (v, v_i) is not the last saturated arc of $(v, v_i) \cup T_{v_i}$ and $\varepsilon_{v_i} - \alpha_i$ is the volume of flow sent through v before the time t'_{v_i}. Secondly, if $\alpha_i > \varepsilon_{v_i}$, $t'_{v_i} = t_{v_i} + \frac{\alpha_i - \varepsilon_{v_i}}{k_{v_i}}$ by Lemma 1. Indeed, after the time t_{v_i}, every terminal of T_{v_i} feeds (v, v_i). No flow is sent through v before t'_{v_i} as (v, v_i) is in that case the last saturated arc of $(v, v_i) \cup T_{v_i}$.

After each time t'_{v_i}, each arc of $(v, v_i) \cup T_{v_i}$ is saturated and the terminals of T_{v_i} sends, by Lemma 1, $k_{v_i} \cdot (t_v - t'_{v_i})$ units of flow through v. Thus, $t_v = \max_{i \in [\![1;s]\!]} t'_{v_i}$.

$$\varepsilon_v = \sum_{i=1}^{s} (k_{v_i} \cdot (t_v - t'_{v_i})) + \sum_{i \setminus \alpha_i \leq \varepsilon_{v_i}} (\varepsilon_{v_i} - \alpha_i)$$

$$\varepsilon_v = t_v \cdot k_v - \sum_{i \setminus \alpha_i \leq \varepsilon_{v_i}} (k_{v_i} \cdot t'_{v_i}) - \sum_{i \setminus \alpha_i > \varepsilon_{v_i}} (k_{v_i} \cdot t'_{v_i}) + \sum_{i \setminus \alpha_i \leq \varepsilon_{v_i}} (\varepsilon_{v_i} - \alpha_i)$$

$$\varepsilon_v = t_v \cdot k_v - \sum_{i \setminus \alpha_i \leq \varepsilon_{v_i}} (\omega_{v_i} + \alpha_i) - \sum_{i \setminus \alpha_i > \varepsilon_{v_i}} (\omega_{v_i} + \alpha_i)$$

Thus $\varepsilon_v = t_v \cdot k_v - \omega_v$. □

We now recall the lemma (adapted with our terminology) from [10].

Lemma 5 ([10], Lemma 1). *Let f be a function such that $\frac{f(k)}{k}$ is non strictly decreasing. Suppose for any DST instance with k terminals and ω^* as the minimum solution cost, that FLAC returns a tree T_0 with $d(T_0) \leq f(k) \cdot \frac{\omega^*}{k}$. Then $Greedy_{FLAC}$ is a $\int_0^k \frac{f(u)}{u} du$-approximation for DST.*

Theorem 1. *$Greedy_{FLAC}$ is a k-approximation for the DST problem.*

Proof. By Lemma 3 and 4, $d(T_0) \leq \omega(P)$. Because any optimal solution contains a path from r to y, $d(T_0) \leq \omega^*$ where ω^* is the optimal cost of the instance. And then $d(T_0) \leq k\frac{\omega^*}{k}$. By Lemma 5, $\text{Greedy}_{\text{FLAC}}$ ratio is $\int_0^k du = k$. □

2.5 Approximation Ratio of $\text{Greedy}_{\text{FLAC}}^{\triangleright}$

We recall $\text{Greedy}_{\text{FLAC}}^{\triangleright}$ applies $\text{Greedy}_{\text{FLAC}}$ in the shortest path instance $\mathcal{I}^{\triangleright}$ in order to build a feasible solution $T^{\triangleright}$ of $\mathcal{I}^{\triangleright}$. From that solution, a feasible solution of $\mathcal{I}$ is build by selecting for each arc (u, v) in $T^{\triangleright}$ a shortest path linking u to v. Note that the cost of $T^{\triangleright}$ is greater than the cost of T.

Let T_2 be in $\mathcal{I}^{\triangleright}$ a minimum density tree of height at most 2. Note that if T_2 is a disjoint union of trees rooted at r, they all have the same density $d(T_2)$. We keep one such tree, so that r has only one child v. Let $x_1, x_2, \cdots, x_l$ be the children terminals of v. Let $\alpha = \omega(r, v)$ and $\beta_i = \omega(v, x_i)$ such that $\beta_1 \leq \beta_2, \leq \cdots \leq \beta_l$.

Lemma 6. *t_f is no more than the density of T_2.*

Proof. As T_2 has minimum density among all the trees of height 2, for all $i \leq l$, $\beta_i \leq d(T_2)$. If $t_f \leq \beta_i \leq d(T_2)$ for some i, the lemma is proved. If not, by Lemma 2, each arc (v, x_i) is saturated at iteration $j_i < f$ and $t_{j_i} \leq \beta_i$. Consequently, at the first iteration j when $t_j \geq \beta_l$, by Lemma 1, v sent at least $\sum_{i \leq l}(t_j - \beta_i)$ units of flow inside (r, v). Thus $t_j + t_j(r, v) \leq t_j + \frac{\alpha - \sum_{i \leq l}(t_j - \beta_i)}{l} = d(T_2)$. By Lemma 2, $t_f \leq d(T_2)$. □

Theorem 2. *Greedy$_{FLAC}{}^{\triangleright}$ is a $4\sqrt{2k}$-approximation for DST.*

Proof. We follow the proof given in [10]. A tree T_2 of height 2 with best density has a lower density that any tree T_2^* of height 2 spanning X with best cost. The cost of the tree T_2^* is a $4\sqrt{k/2}$-approximation of any minimum solution T^* of DST [26, 11]. By Lemma 4 and 6, $d(T_0) \leq d(T_2) \leq 4\sqrt{k/2}\frac{\omega(T^*)}{k} = 2\sqrt{2k}\frac{\omega(T^*)}{k}$. By Lemma 5, Greedy$_{\text{FLAC}}$, applied in $G^{\triangleright}$ by Greedy$_{\text{FLAC}}{}^{\triangleright}$, returns a solution $T^{\triangleright}$ of cost at most $\int_0^k \frac{2\sqrt{2u}}{u} du \cdot \omega(T^*) = 4\sqrt{2k} \cdot \omega(T^*)$.

Finally, note that the cost of T is lower than the cost of $T^{\triangleright}$, thus is also a $4\sqrt{2k}$-approximation. □

Next Section is dedicated to describing a faster implementation of Greedy$_{\text{FLAC}}$ and, consequently, of Greedy$_{\text{FLAC}}{}^{\triangleright}$.

3 A faster implementation

This section provides a faster implementation of the algorithm described in Section 2.2 and proves that, with this faster version of FLAC, Greedy$_{\text{FLAC}}$ runs in time $O(k^2mn)$. Three steps need to be detailed: saturating an arc, detecting a degenerate flow and building the returned tree T_0.

Note that, at each iteration, all the unsaturated and unmarked entering the arcs of a node v have the same flow $f(u, v)$ and flow rate $k(v)$ and are saturated or marked by order of weight. Thus, instead of storing flow with the arcs, we store them with v. We store the nodes in a Fibonacci heap $\mathcal{F}$, where each node v is associated with the next time t_v one of its entering arcs is saturated.

For each node u, we define T_u as the set of nodes w from which there is a path to u in G_{SAT}. When the flow is degenerate, we mark the arc and remove it from G_{SAT}, thus T_u is a tree. We store the list $\Gamma^-(v)$ of arcs entering v, sorted by weights, the sets $\Gamma^-_{SAT}(v)$ and $\Gamma^+_{SAT}(v)$ of saturated arcs entering v and outgoing from v and the next saturated arc a_v entering v. Note that, for each node u, we can iterate over T_u in linear time by returning u and then recursively iterating over T_w for each node w such that (w, u) is in $\Gamma^-_{SAT}(u)$. We also store

the terminals X_v feeding v in a boolean array of size k: $X_v[y]$ is true if G_{SAT} contains a path from v to y. We finally keep the current time t.

Saturating arcs. We have to find, at each step, the next saturated arc. At each step, we remove the first node v from the heap $\mathcal{F}$ in time $O(\log(n))$. The next saturated arc is a_v. We update the value of t to the key t_v of v in $\mathcal{F}$.

Detecting a degenerate flow. A flow is degenerate if there are two distinct paths in G_{SAT} or a path with a cycle from a node to a terminal. When $a_v = (u, v)$ becomes saturated, there is a new path in G_{SAT} from each node w in T_u to each terminal v is linked to. As a consequence, to detect a degenerate flow, we have to check for each node w in T_u whether X_w and X_v are disjoint or not: is there a terminal y with $X_v[y] = X_w[y] = true$? We check the sets intersection in time $O(k)$. The complete detection takes at most $O(nk)$ steps.

Updating the flow rates. If the flow is not degenerate, we have to update all the nodes affected by the saturation of (u, v). We first add a_v to $\Gamma^-_{SAT}(v)$ and to $\Gamma^+_{SAT}(u)$. If the arc entering v are saturated, we do nothing more. Otherwise, we get the minimum cost unsaturated arc a'_v entering v in $O(1)$ as $\Gamma^-(v)$ is sorted by weight. And v can be reinjected inside $\mathcal{F}$ with $t'_v = t + \frac{\omega(a'_v)-\omega(a_v)}{|X_v|}$.

Then we update all the nodes w in T_u. If w was not in $\mathcal{F}$, if a_w is its minimum cost unsaturated entering arc, $X_w = X_v$ and $t_w = t + \frac{\omega(a_w)}{|X_v|}$. If it was, the new set X'_w is $X_w \uplus X_v$, computed in time $O(k)$. Next we decrease the key t_w in $\mathcal{F}$ to $t + \frac{(t_w - t)\cdot|X_w|}{|X'_w|}$. This takes $O(nk)$ operations.

Building the returned tree. All the saturated arcs to which the root is linked constitute a tree spanning terminals. There are at most $n - 1$ such arcs. Thus this is done once in time $O(n)$ by returning r and recursively iterating over the saturated arcs w is linked to for each node w such that (r, w) is in $\Gamma^+_{SAT}(r)$.

The running time of Algorithm Greedy$_{\mathbf{FLAC}}$. To return a DST feasible solution, the FLAC algorithm is called at most k times as each call spans at least one terminal. For each of them, m arcs at most are saturated, leading to a conflict detection, and updates in time $O(\log(n) + 2nk)$. Finally the solution is built in linear time. Consequently, we return the solution in time $O(k^2mn)$.

Remark 2. If a $O(\sqrt{k})$-approximation is required, we use Greedy$_{\text{FLAC}}{}^{\triangleright}$ and have to compute FLAC in $\mathcal{I}^{\triangleright}$. The Dijkstra algorithm can be used to build $\mathcal{I}^{\triangleright}$ in time $O(nm + n^2\log(n))$, and then the algorithm Greedy$_{\text{FLAC}}{}^{\triangleright}$ runs in time $O(k^2n^3)$.

4 Evaluations of performance

4.1 Ratio Study

We firstly compare Greedy$_{\text{FLAC}}$ with Greedy$_{\text{FLAC}}{}^{\triangleright}$ over practical cases. Surprisingly, Greedy$_{\text{FLAC}}$ returns solutions with smaller or same costs than Greedy$_{\text{FLAC}}{}^{\triangleright}$ in nearly all cases. Considering that the running time of Greedy$_{\text{FLAC}}$ is always smaller than that of Greedy$_{\text{FLAC}}{}^{\triangleright}$, one should use Greedy$_{\text{FLAC}}$ in practice.

We then compared four algorithms with Greedy$_{\text{FLAC}}$: two versions of the mostly used in practice the Shortest Paths algorithm (ShP$_1$ and ShP$_2$), the first

and fastest Dual Ascent algorithm (DuAs) [24] and the CH_2 algorithm [10]. ShP_1 was implemented using Dijkstra's algorithm to compute all shortest paths from the root to all terminals. ShP_2 selects a shortest path from the root to its closest terminal, sets the weights of the arcs of that path to 0, and restarts until all the terminals are reached. CH_2 was implemented with the Roos modified algorithm described in [22][1]. We now detail the test set used during the evaluation.

Description of the test set. To run the experiments, we transformed the SteinLib instances [17] into three directed instances test sets. Each undirected graph was transformed into one or more directed graphs with the same optimal cost. We first created the bidirected test set (BTS) containing 999 instances, where each edge was replaced by two opposite arcs with the same cost and chose any terminal as the root. As those instances seem too specific to reflect the variety of all instances and applications, we also created an acyclic instances test set (ATS) and a strongly connected instances test set (SCTS), containing 353 instance each. In order to build those test sets, we first computed an optimal tree T^* of each instance (currently, we have processed one third of all the instances). Then we transformed T^* into a directed tree rooted at any of its nodes. In (ATS), we directed all other edges in order to get an acyclic digraph with breadth-first search from the root. In (SCTS), every other edge was directed uniformly at random. Then, until we got a strongly connected graph, an arc was added between a random couple of unconnected nodes with cost equal to that of the shortest path cost in the original instance.

The test sets were generated from the following SteinLib groups : WRP3, WRP4, ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ, LIN, SP, X, MC, I080 to I640, ES10FST to ES10000FST, B,C,D,E, P6E and P6Z. Other groups, for instance P4E and P4Z, were not considered because they contain incorrect files.

Comparison of $\mathbf{Greedy_{FLAC}}$ with $\mathbf{Greedy_{FLAC}}^{\triangleright}$. Considering the cost ω of a solution returned by $\text{Greedy}_{\text{FLAC}}$, the cost $\omega^{\triangleright}$ of a solution returned by $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$, and the optimal cost ω^*, we are interested in the relative distance $\delta = \frac{\omega - \omega^{\triangleright}}{\omega^{\triangleright}}$ and the cases when $\omega = \omega^*$ or $\omega^{\triangleright} = \omega^*$.

We removed from the test set (BTS) the biggest instances where $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ was not able to return a result due to required memory. As a consequence, it contains 471 instances for this evaluation. Table 3 summarises the results.

From this evaluation, we conclude that:

- $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ returns strictly smaller costs for less than 10% of the instances;
- according to the columns $\delta = 0$ and $|\delta| < 5\%, 10\%, 15\%$, the costs ω and $\omega^{\triangleright}$ are frequently equal or close;
- $\text{Greedy}_{\text{FLAC}}$ returns an optimal solution in more cases than $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$(on the contrary, the number of instances optimally solved by $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ but not by $\text{Greedy}_{\text{FLAC}}$ is less than 3%).

Due to the computation of $\mathcal{I}^{\triangleright}$ and to the number of degenerate flows that $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ encounters in a directed complete graph, its running time is much

[1] The four algorithms were run with Java 1.7.0_025 on Ubuntu 12.10 with Intel Core 3.10 GHz processors. The code source can be found at https://github.com/mouton5000/DSTAlgoEvaluation

	$\delta < 0$	$\delta = 0$	$\delta > 0$	$\|\delta\| < 5\%$	$\|\delta\| < 10\%$	$\|\delta\| < 15\%$	$\omega = \omega^*$	$\omega^{\triangleright} = \omega^*$
BTS	54%	34%	10%	83%	98%	99%	20%	15%
ATS	17%	71%	10%	94%	98%	99%	48%	46%
SCTS	16%	76%	6%	95%	98%	99%	63%	57%

Table 3: Comparison of the the costs ω and $\omega^{\triangleright}$ over each test set. The left part details the number of instances for which ω is lower, equal or greater than $\omega^{\triangleright}$. The middle part explains how close are those costs. The last part gives the number of instances for which the algorithms return an optimal solution.

greater than that of $\text{Greedy}_{\text{FLAC}}$. Thus it is preferable to use $\text{Greedy}_{\text{FLAC}}$ in practice than $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$. This happens, contrary to what their approximation ratios suggest, because the shortest path instance, as a complete directed graph, contains shortcuts between the terminals and the root that cannot be employed by $\text{Greedy}_{\text{FLAC}}$, this favours solutions where terminals share their flows.

Comparison of $\text{Greedy}_{\text{FLAC}}$ with other approximation algorithms. Considering the cost ω of the returned solution compared to the optimal cost ω^*, we are interested in the relative error $e = \frac{\omega - \omega^*}{\omega^*}$. We represent in Figure 3 the error cumulative distributions : for each e and each algorithm $\mathcal{A}$, we plots the number of instances for which the relative error of $\mathcal{A}$ is lower than e.

We removed from the test set (BTS) the biggest instances where DuAs and CH_2 were not able to return a result due to required memory (contrary to ShP and $\text{Greedy}_{\text{FLAC}}$ which always returned a result). As a consequence, it contains 918 instances for this evaluation.

Note first that Figure 3 aggregates all the instances of each test set and does not take into account the number of nodes or terminals distribution, or the different SteinLib categories. Considering this, $\text{Greedy}_{\text{FLAC}}$ gives better practical results, followed by ShP_2, in all test sets, as most of the instances are solved with less than 5% error. ShP_1 returns the worst results for the precision inferior to 30% and then overtakes CH_2. (ATS) and (SCTS) are better solved by DuAs than the shortest paths algorithms. This is because (ATS) and (SCTS) instances contain fewer feasible solutions than (BTS), but contain almost the same shortest paths from the root to terminals. We conclude that $\text{Greedy}_{\text{FLAC}}$ outperforms the shortest path algorithms, DuAs and CH_2. We study in the next subsection the execution time results to determine the practical interest of $\text{Greedy}_{\text{FLAC}}$.

4.2 Time study

We used a different test set for this evaluation because we want to study the impact on the running time of 3 parameters : the number of nodes n, of arcs m, and of terminals k. As we do not control those parameters with the SteinLib data set, we generated a set of random instances and ran all algorithms, except $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ which running time is very high compared to others.

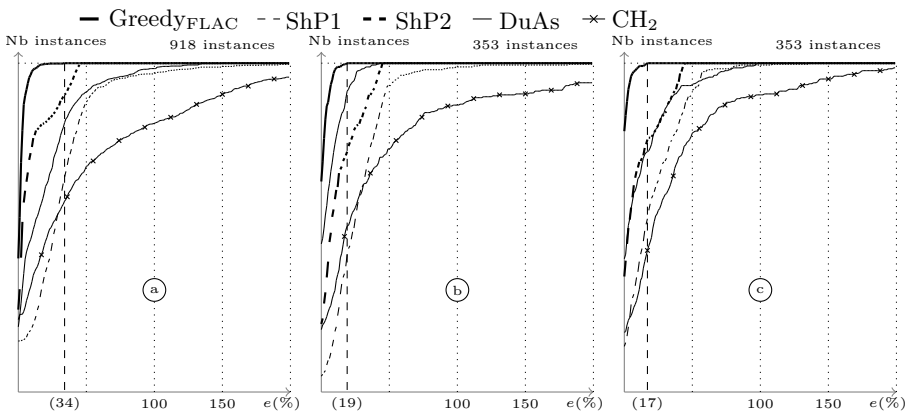

Fig. 3: Cumulative distribution function of the relative error e in percent, for each algorithm. The tree test sets are a) BTS, b) ATS, and c) SCTS. The maximum error of solutions obtained with Greedy$_{\text{FLAC}}$ is indicated in parentheses.

The instances were generated with 3 parameters : n, the probability to link two nodes with an arc p, and the ratio $\frac{k}{n}$. We first add n nodes to an empty instance, then select the first one as root and the k following nodes as terminals. Finally, for each couple (u, v) of nodes, we add the arc (u, v) to the graph with probability p. The weight of the arc is chosen uniformly between 0 and a maximum chosen between 1 and 100. The probability p is the expected value of $\frac{m}{n^2}$. We built 10500 instances of 50 nodes, 5250 instances of 100 nodes and 1050 instances of 250 nodes. Table 4 summarises the results.

It is highlighted that ShP$_1$ is the fastest algorithm, followed by Greedy$_{\text{FLAC}}$ which is competitive, according to its mean and its low standard deviation.

The algorithms ShP$_2$ and CH$_2$ are slower than other algorithms, due to k distinct applications of the Dijkstra algorithm. Finally, note the high standard deviation of DuAs, due to the random part of that algorithm. Indeed, its execution depends on the order in which the terminals are given to the algorithm.

5 Conclusion and perspectives

We introduced a fast greedy algorithm for the Directed Steiner Tree problem. When it is opposed to the fastest Shortest Paths algorithm mostly used in today's applications, it shows lower error rates while maintaining a fast running time.

Even so, this algorithm was implemented assuming a central processing unit exists. As the main application of the Directed Steiner Tree is the multicast tree in networks, it would be interesting to work on a distributed implementation, minimizing the number of exchanged messages, following the technique of [21].

Moreover, we built two new test sets of directed instances, in addition to the bidirected instances, in order to create hard instances for which some of the

n	p	$\frac{k}{n}$	$\text{Greedy}_{\text{FLAC}}$		ShP_1		ShP_2		DuAs		CH_2	
			mean	s. dev.	mean	s. dev.	mean	s. dev.	mean	s. dev.	mean	s. dev.
50	0.3	0.3	< 1		< 1		1	0.5	2	1.9	2	0.8
		0.5	< 1		< 1		1	0.6	2	2.2	3	1.0
		0.8	1	0.5	< 1		2	0.6	2	2.5	5	1.3
	0.5	0.3	1	0.7	< 1		1	0.6	3	3.4	2	0.7
		0.5	1	0.6	< 1		2	0.7	3	3.1	3	1.0
		0.8	1	0.7	< 1		3	1.0	3	2.9	6	1.6
	0.8	0.3	1	0.7	< 1		2	0.6	4	4.5	2	0.8
		0.5	1	0.8	< 1		3	0.8	4	4.9	4	1.5
		0.8	1	0.8	< 1		5	0.7	4	4.2	7	1.8
100	0.3	0.3	2	1.0	< 1		8	1.1	15	20.9	11	2.3
		0.5	3	1.2	< 1		13	1.8	14	14.0	19	3.1
		0.8	4	1.5	< 1		21	3.5	15	14.9	31	4.0
	0.5	0.3	4	1.5	1	0.5	13	2.1	20	20.9	16	2.7
		0.5	5	1.5	1	0.4	21	3.0	17	22.8	26	3.7
		0.8	6	1.6	1	0.5	34	5.2	19	24.2	44	5.4
	0.8	0.3	6	1.7	2	0.5	20	2.1	48	56.4	25	2.8
		0.5	7	2.0	2	0.6	34	3.5	33	39.3	41	3.3
		0.8	8	2.2	2	0.6	53	6.2	27	35.4	66	6.7
250	0.3	0.3	31	14.7	7	0.8	343	39.1	515	921.3	354	39.3
		0.5	37	14.4	7	0.7	513	74.9	542	728.8	615	74.4
		0.8	46	18.5	7	0.8	807	136.2	398	441.0	1021	203.4
	0.5	0.3	60	28.3	13	1.3	639	46.5	948	1217.1	587	73.2
		0.5	75	36.7	13	1.2	1073	103.7	777	1137.8	980	133.5
		0.8	82	33.6	13	1.5	1638	201.2	781	1111.2	1551	299.2
	0.8	0.3	95	46.1	22	2.0	1116	80.6	1763	2789.1	867	166.2
		0.5	128	68.3	22	2.1	1819	193.3	1697	2510.1	1511	216.0
		0.8	127	65.5	23	4.0	2859	289.0	1226	1903.6	2331	463.3

Table 4: Average running times in milliseconds and standard deviations for each class of graphs, depending on their number of nodes, their expected number of arcs and their number of terminals.

approximation algorithms would return high relative errors. However, the three test sets seems equally hard to solve. Further works are needed to build instances for which approximation algorithms could be efficiently compared and clustered.

Finally, in Section 2, to be competitive and simple to implement, FLAC avoids degenerate flows by removing the last saturated arc. This solution is not sufficient to reduce the approximation ratio of $\text{Greedy}_{\text{FLAC}}{}^{\triangleright}$ under $O(\sqrt{k})$, but other ways could be explored to achieve better theoretical performances.

References

1. Karp, R.M.: Reducibility among combinatorial problems. Springer (1972)
2. Kou, L., Markowsky, G., Berman, L.: A fast algorithm for steiner trees. Acta informatica **15**(2) (1981) 141–145
3. Zelikovsky, A.Z.: An 11/6-approximation algorithm for the network steiner problem. Algorithmica **9**(5) (1993) 463–470
4. Byrka, J., Grandoni, F., Rothvoss, T., Sanità, L.: Steiner tree approximation via iterative randomized rounding. Journal of the ACM (JACM) **60**(1) (2013) 6:1–33

5. Cheng, X., Du, D.Z.: Steiner trees in industry. Volume 11. Springer (2001)
6. Voß, S.: Steiner tree problems in telecommunications. In: Handbook of optimization in telecommunications. Springer (2006) 459–492
7. Novak, R., Rugelj, J., Kandus, G.: A note on distributed multicast routing in point-to-point networks. Computers & Operations Research **28**(12) (2001) 1149–1164
8. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM (JACM) **45**(4) (1998) 634–652
9. Halperin, E., Krauthgamer, R.: Polylogarithmic inapproximability. In: Proc. of the thirty-fifth annual ACM symposium on Theory of computing. (2003) 585–594
10. Charikar, M., Chekuri, C., Cheung, T.y., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed steiner problems. Journal of Algorithms **33**(1) (1999) 73–91
11. Helvig, C.S., Robins, G., Zelikovsky, A.: An improved approximation scheme for the group steiner problem. Networks **37**(1) (2001) 8–20
12. Johnson, D.S.: Approximation algorithms for combinatorial problems. In: Proc. of the fifth annual ACM symposium on Theory of computing. (1973) 38–49
13. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of operations research **4**(3) (1979) 233–235
14. Olsson, P.M., Kvarnstrom, J., Doherty, P., Burdakov, O., Holmberg, K.: Generating uav communication networks for monitoring and surveillance. In: Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on, IEEE (2010) 1070–1077
15. Gundecha, P., Feng, Z., Liu, H.: Seeking provenance of information using social media. In: Proc. of the 22nd ACM international conference on Conference on information & knowledge management, ACM (2013) 1691–1696
16. Lappas, T., Terzi, E., Gunopulos, D., Mannila, H.: Finding effectors in social networks. In: Proc. of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2010) 1059–1068
17. Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on Steiner tree problems in graphs. Springer (2001)
18. Chimani, M., Woste, M.: Contraction-based steiner tree approximations in practice. In: Algorithms and Computation. Springer (2011) 40–49
19. Stanojevic, M., Vujosevic, M.: An exact algorithm for steiner tree problem on graphs. International Journal of Computers, Communications & Control **1**(1) (2006) 41–46
20. Uchoa, E., Werneck, R.F.F.: Fast local search for steiner trees in graphs. In: ALENEX. Volume 10., SIAM (2010) 1–10
21. Drummond, L., Santos, M., Uchoa, E.: A distributed dual ascent algorithm for steiner problems in multicast routing. Networks **53**(2) (2009) 170–183
22. Hsieh, M.I., Wu, E.H.K., Tsai, M.F.: Fasterdsp: A faster approximation algorithm for directed steiner tree problem. J. Inf. Sci. Eng. (2006) 1409–1425
23. de Aragão, M.P., Uchoa, E., Werneck, R.F.: Dual heuristics on the exact solution of large steiner problems. Electronic Notes in Discrete Mathematics **7** (2001) 150–153
24. Wong, R.T.: A dual ascent approach for steiner tree problems on a directed graph. Mathematical programming **28**(3) (1984) 271–287
25. Melkonian, V.: New primal–dual algorithms for steiner tree problems. Computers & operations research **34**(7) (2007) 2147–2167
26. Zelikovsky, A.: A series of approximation algorithms for the acyclic directed steiner tree problem. Algorithmica **18**(1) (1997) 99–110