An FPT algorithm in polynomial space for the Directed Steiner Tree problem with Limited number of Diffusing nodes

Dimitri Watel[a,c,*], Marc-Antoine Weisser[a,*], Cédric Bentz[b], Dominique Barth[c]

[a] *SUPELEC System Sciences, Computer Science Dpt., 91192 Gif sur Yvette, France*
[b] *CEDRIC-CNAM 292 rue Saint-Martin 75141 Paris, France*
[c] *University of Versailles, 45 avenue des Etats-Unis, 78035, France*

Abstract

Given a directed graph with n nodes, a *root* r, a set X of k nodes called *terminals* and non-negative weights ω over the arcs, the Directed Steiner Tree problem (DST) asks for a directed tree T^* of minimum cost $\omega(T^*)$ rooted at r and spanning X.

If this problem has several applications in multicast routing in packet switching networks, the modeling is no longer adapted in networks based upon the circuit switching principle, in which some nodes, called non-diffusing nodes, are not able to duplicate packets. We study a generalization of DST, called Directed Steiner Tree with Limited number of Diffusing nodes (DSTLD), able to model the multicast routing in a network containing at most d diffusing nodes.

We provide an FPT exact algorithm running in time $O(t_d \cdot d^k \cdot n \cdot (d+k))$ and in polynomial space for DSTLD, where t_d is the number of unordered rooted trees with d unlabelled nodes. Thereby, we also provide the first exact algorithm running in polynomial space and in FPT time with respect to k which returns an optimal solution for DST instead of the optimal cost only.

Keywords: Directed Steiner Tree, Parameterized Complexity, Dynamic programming

1. Introduction

Problem 1. Given a directed graph with n nodes, a *root* r, a set X of k nodes called *terminals* and non-negative weights ω over the arcs, the Directed Steiner Tree problem (DST) asks for a directed tree T^* of minimum cost $\omega(T^*)$ rooted at r and spanning all the terminals.

*Corresponding authors
Email addresses: dimitri.watel@supelec.fr (Dimitri Watel), marc-antoine.weisser@supelec.fr (Marc-Antoine Weisser), cedric.bentz@cnam.fr (Cédric Bentz), dominique.barth@prism.uvsq.fr (Dominique Barth)

As a generalization of the NP-Complete Undirected Steiner Tree problem [8], DST is itself NP-Complete. The Steiner problems are known to have applications essentially in multicast routing where one wants to minimize bandwidth consumption [2, 10, 14], when the graph modelises a network (typically an MPLS network) and a tree modelises a multicast scheme (one-to-many communication) in this network.

The DST model assumes that when a *branching node* (i.e., a node with at least 2 successors) of the tree receives multicast data, it can transmit it to its multiple successors. This is the case in classical packet switching networks. However, previous works emphasize the fact that, in optical networks, this assumption no longer holds as most of the nodes, called *non-diffusing nodes*, can not copy any multicast data [7, 11, 12] but only send it to one of its children. As a consequence, a non-diffusing branching node has to receive p copies of the data to transmit it to p children. The cost of an arc has to be paid each time the data is sent through that arc. Fortunately, some routers, called *diffusing nodes*, can duplicate data and thus need to receive any data only once. Those routers add extra cost to the network due to maintenance and, when splitting packets, they can introduce supplementary delay to the transmission or an attenuation of the optical signal. Therefore they have to be limited in the solution.

Note that the root can generate and send multiple copies of the same data without being a diffusing node. However, in order to simplify some of the demonstrations in this paper, we assume that a non-diffusing root generates and sends the data only once.

The diffusing nodes are not predefined in the graph: we only know the number d of diffusing nodes we can use. Intuitively, a multicast request can be answered by a d-route : a set of at most d chosen diffusing nodes and a set of path $\mathcal{P}$, each one defining the route of non-diffusing nodes followed by a copy of the multicast data from the root or a diffusing node to another diffusing node or a terminal such that each terminal and each diffusing node receives at least one copy of the data. Some arcs may possibly belong to two or more paths of $\mathcal{P}$. The cost of the d-route is the sum of the weights of the paths in $\mathcal{P}$. We want to determine a minimum cost d-route.

In order to describe that route, we introduce the *shortest paths graph*.

Definition 1. Let G be a directed graph and ω a weight function over the arcs of G. 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 graph* $G^{\triangleright} = (V, A^{\triangleright})$ is a complete graph where each arc (u, v) is weighted by $\omega^{\triangleright}(u, v)$.

There is a relation between the trees in $G^{\triangleright}$ linking the root to the terminals, using at most d internal branching nodes and the d-routes in G.

Let $T^{\triangleright}$ be a tree in $G^{\triangleright}$ linking the root to the terminals, using at most d internal branching nodes. We build a d-route in G by following the shortest paths described by the arcs of $T^{\triangleright}$. Each time the data encounters an internal branching node of $T^{\triangleright}$, this node is chosen as a diffusing node and the data is copied as many times as necessary to follow the shortest outgoing paths of that node. The cost $\sum_{(u,v)\in T^{\triangleright}} \omega^{\triangleright}(u, v)$ is exactly the cost of the d-route.

Conversely, we now assume we know d-route which cost is ω. We define the tree $T^{\triangleright}$ in $G^{\triangleright}$ with at most d internal branching nodes as the set of arcs (u, v) where u is the root or a diffusing node, v is another diffusing node or a terminal and u send a copy of the multicast data to v. As each arc of $G^{\triangleright}$ is weighted with a shortest path cost, the weight of $T^{\triangleright}$ is less than ω.

Consequently, an optimal solution of the following problem gives an optimal placement of the diffusing nodes and an optimal d-route.

Problem 2. Given a directed graph G with n nodes, a *root* r, a set X of k nodes called *terminals*, non-negative weights ω over the arcs and an integer $d \in [\![1; k-1]\!]$, the Directed Steiner Tree problem with Limited number of Diffusing nodes (DSTLD) asks for a minimum cost directed tree in $G^{\triangleright}$ rooted at r and spanning all the terminals with at most d internal branching nodes.

DSTLD is XP with respect to d and can not be approximated in polynomial time with a better ratio than $1 + (\frac{1}{e} - \varepsilon)\frac{k}{d-1}$ [15].

We assume the shortest paths graph $G^{\triangleright}$ is given. If not, it can be built in polynomial time and space using Dijkstra's algorithm.

Remark 1. DSTLD is a generalization of DST where $d = k - 1$. Indeed, an optimal Steiner tree contains at most $k - 1$ branching nodes. Consequently, if only the branching nodes are defined as diffusing nodes, the cost of the tree remains the same.

In this paper, we are interested in the Fixed-Parameter Tractability of DST with respect to k. That DST belongs to FPT is not new, but, to our knowledge, there is no algorithm running in polynomial space and in FPT time with respect to k, which returns an optimal solution for DST instead of the optimal cost only. Table 1 summarizes known parameterized algorithms for the Undirected Steiner Tree problem, which can be adapted for DST and DSTLD.

Time Complexity	Space Complexity	Returns $\omega(T^*)$	Returns T^*	Reference
$O^*(3^k)$[1]	$O^*(2^k)$	YES	YES	[3]
$O^*((2+\varepsilon)^k)$, $\varepsilon > 0$	$O^*(2^k)$	YES	YES	[6]
$O^*(2^k)$	$O^*(2^k)$	YES	YES	[1]
$O^*(5.96^k n^{O(\log(k))}$	P	YES	YES	[5]
$O^*(2^k)$	P	YES	NO	[9]
$O^*(1 \cdot 3 \cdot 5 \cdots (2k-3))$	P	YES	YES	Theorem 8

Table 1: Parameterized algorithms for the Undirected and Directed Steiner Tree problems. We distinguish algorithms returning only the value $\omega(T^*)$ of an optimal solution T^* and algorithms returning T^*.

[1] The O^* notation omits the polynomial factors.

We provide an FPT exact algorithm for DSTLD running in time $O(t_d \cdot d^k \cdot n \cdot (d+k))$ and using polynomial space where t_d is the number of unordered rooted trees with d unlabelled nodes. As DSTLD is a generalization of DST, we also provide an exact algorithm for DST, FPT in k by using polynomial space.

2. The algorithm description

In this section, we describe an exact algorithm for DSTLD. Let $\mathcal{I} = (G = (V, A), r, X, \omega, d)$ be an instance for DSTLD.

We firstly regroup every possible solution for DSTLD in classes called *patterns*. The main idea of the algorithm is that determining an optimal solution among all the trees of one pattern can be done in polynomial time. Furthermore, the number of patterns only depends on k and d.

2.1. Patterns

Definition 2. A *pattern* for instance $\mathcal{I}$ is a tree rooted at r with k leaves and d unlabelled internal branching nodes. The root has only one child. Each internal node has at least two children. Each leaf is a distinct terminal in X. The children of any node are not ordered so that swapping two children (and the subtrees rooted at them) does not modify the pattern.

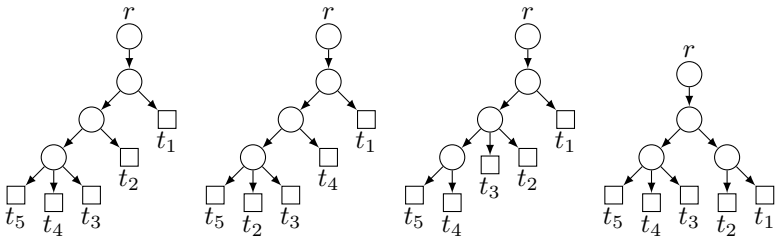

Figure 1: Four examples of distinct patterns for an instance with $d = 3$ and $k = 5$. There are 120 such patterns. Note that on the first pattern, if we exchange t_2 and t_4, we get another pattern, but if we exchange t_3 and t_4, we do not as the children of a same node are not ordered. Similarly, on the last pattern, if we swap the set of terminals $\{t_1, t_2\}$ and $\{t_3, t_4, t_5\}$, as the internal nodes are unlabelled, we get the same pattern.

We now describe how to find a feasible solution of $\mathcal{I}$ from a pattern.

Definition 3. Let P be a pattern for $\mathcal{I}$ and I_P its unlabelled internal nodes. A *labelling* function associates to each node of I_P a node of V and associates each node in $\{r\} \cup X$ to itself.

A labelling function κ of P is associated to a feasible solution of $\mathcal{I}$.

Definition 4. Let P be a pattern for $\mathcal{I}$ and κ a labelling of P. The feasible solution $\kappa(P)$ is in $G^{\triangleright}$ the set of arcs $(\kappa(v_1), \kappa(v_2))$ for each arc (v_1, v_2) in P.

It is important to note that the definition of a labelling function κ of a pattern P allows multiple nodes of P to be labelled with the same node and allows internal nodes to be labelled with the root or terminals. If two nodes, v_1 and v_2, with same label are linked in P, the resulting arc $(\kappa(v_1), \kappa(v_2))$ in $\kappa(P)$ is empty.

Lemma 1. *There are at most $t_d \cdot d^k$ patterns for $\mathcal{I}$ where t_d is the number of unordered rooted trees with d unlabelled nodes*[1]*.*

Proof. A pattern for $\mathcal{I}$ is an unordered rooted tree with root r, d unlabelled internal branching nodes and k labelled leaves. Thus, we can build a set containing all the patterns (but not only the patterns) by iterating over all unordered rooted trees P_0 with d nodes, linking r to the root of P_0 and selecting for each terminal a node of P_0 as father. There are at most $t_d \cdot d^k$ such trees. □

Lemma 2. *If $d = k - 1$, there are $1 \cdot 3 \cdot 5 \cdots 2k - 3$ patterns for $\mathcal{I}$.*

Proof. If $d = k - 1$, the patterns are rooted unordered binary trees with k labelled leaves. To enumerate all those trees, we can adapt the algorithm from [13]: from a pattern with $k-1$ terminals, one can build a unique pattern with k terminals by choosing any node v except the root (v can be an internal node or a leaf), by inserting a new node v' between v and its father, and then inserting the k-th terminal as a second child of v'. There are $2k - 3$ choices for v. Thus if there are $f(k)$ patterns with k leaves and $k - 1$ internal branching nodes, $f(k) = f(k - 1) \cdot (2k - 3)$. As $f(1) = f(2) = 1$, $f(k) = 1 \cdot 3 \cdot 5 \cdots 2k - 3$. This result can also be found in [4, page 129]. □

We now build a dynamic programming algorithm to determine, for a given pattern P, the labelling function κ minimizing the cost of the solution $\kappa(P)$.

2.2. A dynamic programming algorithm

Let P be a pattern for $\mathcal{I}$. We want to find the labelling function κ of P minimizing the cost of the solution $\kappa(P)$.

We associate, to each node v of P, two arrays:

- C_v is an array of size n containing, for each node $u \in V$, the cost of the subtree of P rooted at v if $\kappa(v) = u$,
- D_v is a matrix of size $n \cdot |ch(v)|$, where $ch(v)$ are the children of v in P. For each node $u \in V$ and each child $w \in ch(v)$, $D_v[u, w]$ contains the most *fitting* label $\kappa(w)$ of w if $\kappa(v) = u$.

For terminal (and leaf) t, we initialize the array C_t with $C_t[t] = 0$ and $C_t[u] = +\infty$ for $u \neq t$. The array D_t is empty as t has no child.

[1] The sequence can be found at `http://oeis.org/A000081` and in [4]

For an internal node v of P or the root, we compute D_v and C_v for each $u \in V$ and each child $w \in ch(v)$ with:

$$D_v[u,w] = \arg\min_{x \in V}(\omega^{\triangleright}(u,x) + C_w[x]) \tag{1}$$

$$C_v[u] = \sum_{w \in ch(v)} \min_{x \in V}(\omega^{\triangleright}(u,x) + C_w[x]) \tag{2}$$

2.3. The final procedure

The *final procedure* builds a labelling function κ recursively. It requires a node v of P and a node u of V to label the subtree of P rooted at v. It sets $\kappa(v) = u$ and, for each child $w \in ch(v)$, applies the final procedure with w and $D_v[u,w]$ as inputs. To build the full labelling function, we call this procedure giving the root of P and r as parameters. Note that the final procedure may produce a solution $\kappa(P)$ with cycles instead of a tree. A feasible solution with no cycles can be extracted from $\kappa(P)$ in polynomial time.

2.4. The pattern algorithm

We use Algorithm 1 to return an optimal solution of $\mathcal{I}$. In the description, for a feasible solution T, $\omega^{\triangleright}(T) = \sum_{(u,v)\in T} \omega^{\triangleright}(u,v)$ is the cost of T.

Algorithm 1 Pattern algorithm

```
1: Initialize T^0 with any feasible solution.
2: for each pattern P for I do
3:     Build arrays C and D in reverse breadth-first-search order
4:     Use the final procedure to build a labelling function κ
5:     T ← κ(P)
6:     if ω▷(T) < ω▷(T^0) then
7:         T^0 = T
8: return T^0
```

3. Correctness and properties of the algorithm

This section is dedicated to proving the correctness of the algorithm and to giving an upper bound of its time complexity.

Lemma 3. *There is one pattern P^* for $\mathcal{I}$ and a labelling function κ^* for which $\kappa^*(P^*)$ is an optimal solution of $\mathcal{I}$.*

Proof. Let T^* be an optimal solution of $\mathcal{I}$. We need to create a labelled pattern from T^*. Firstly, we guarantee there is a one-to-one correspondence between the leaves of the pattern and the terminals labelling them. If a leaf of T^* is not a terminal, it is linked to the tree with a path containing arcs of

weight 0 only as T^* is optimal: we can remove that path. If a terminal t is an internal node, we create a duplicate t' of t, and link t to it.

Secondly, we guarantee the pattern contains d internal branching nodes. We contract to a single arc each path linking the root or a branching node of T^* to another branching node or a terminal. As the arcs are weighted with the costs of shortest paths, and as the solution is optimal, this does not change its cost.

If T^* contains less than $d \leq k-1$ internal branching nodes, at least one node v has a minimum of 3 children. We create a duplicate v' of v and link v to v', we choose 2 children of v and set v' as their father. We repeat this until T^* contains d branching nodes. The resulting is a pattern P for $\mathcal{I}$ labelled with κ such that $\kappa(P) = T^*$. □

We now assume we have built the arrays C and D for the pattern P^*. We will prove that, in this way, we effectively build an optimal labelling function. We firstly prove that the final procedure builds a feasible solution of cost $C_r[r]$.

Lemma 4. *For a node v of P^*, let P^*_v be the subtree of P^* rooted at v. If $C_v[u]$ is not infinite, if we call the final procedure giving v and u as parameters, it builds a labelling function κ of P^*_v such that the cost of $\kappa(P^*_v)$ is $C_v[u]$.*

Proof. We prove the lemma by induction on the height of v (i.e. the length of a longest downward path to a leaf from v). If t is a leaf, the only box of C_t which is finite is $C_t[t] = 0$. The cost of $\kappa(P^*_t)$ is 0 as $\kappa(P^*_t)$ has no arcs.

If v is an internal node, let $u \in V$ be such that $C_v[u]$ is finite. By Equation (2), $C_v[u] = \sum_{w \in ch(v)} \min(\omega^{\triangleright}(u,x) + C_w[x], x \in V)$. As $C_v[u]$ is finite, for each $w \in ch(v)$, the chosen box $C_w[x]$ is finite. By Equation (1), the chosen x is $D_v[u,w]$. Then $C_v[u] = \sum_{w \in ch(v)} (\omega^{\triangleright}(u, D_v[u,w]) + C_w[D_v[u,w]])$.

By the inductive hypothesis, the final procedure with w and $D_v[u,w]$ as parameters builds a labelling function κ of P^*_w such that the cost of $\kappa(P^*_w)$ is $C_w[D_v[u,w]]$. In addition, by definition, given v and u as parameters, the final procedure, labels each node $w \in ch(v)$ with $D_v[u,w]$, thus $\kappa(P^*_v)$ costs $\sum_{w \in ch(v)} \omega^{\triangleright}(u, D_v[u,w]) + C_w[D_v[u,w]]$. The lemma is then true for v. □

We now prove that $C_r[r]$ is the cost of T^*. As Lemma 4 assures we build a solution of cost $C_r[r]$, the two lemmas assure we build an optimal solution.

Lemma 5. *$C_r[r]$ is the cost of an optimal solution for $\mathcal{I}$.*

Proof. By induction on the height of the nodes of P, we will prove the following property: "Let P^*_v be the subtree of P^* rooted at v. Then $C_v[\kappa^*(v)]$ is the cost of $\kappa^*(P^*_v)$". If t is a leaf and a terminal, $\kappa^*(t) = t$. As $C_t[t] = 0$, and as P^*_t does not contain any arc, the property is proven for any leaves.

Assume v is an internal node. $C_v[\kappa^*(v)] \leq \sum_{w \in ch(v)} \omega^{\triangleright}(\kappa^*(v), \kappa^*(w)) + C_w[\kappa^*(w)]$, by Equation (2). By induction, for each $w \in ch(v)$, $C_w[\kappa^*(w)]$ is the cost of $\kappa^*(P^*_w)$. So $C_v[\kappa^*(v)]$ is at most the cost of $\kappa^*(P^*_v)$.

If $C_v[\kappa^*(v)]$ were strictly smaller, the final procedure would build a labelling function κ' for P^*_v of cost strictly smaller than $\kappa^*(P^*_v)$, by Lemma 4. As a

consequence, a labelling function κ equal to κ' on the nodes of P_v^*, and κ^* on others produces a feasible solution $\kappa(P^*)$ of cost strictly smaller than $\kappa^*(P^*)$. This would contradict the optimality of T^* by Lemma 3. The property is then proved for any node v. □

Theorem 6. *The pattern algorithm returns an optimal solution.*

PROOF. By Lemmas 4 and 5. □

Theorem 7. *The pattern algorithm runs in time $O(t_d \cdot d^k \cdot n \cdot (d+k))$ and space $O((d+k) \cdot n + n^2)$, where t_d is the number of unordered rooted trees with d unlabelled nodes.*

PROOF. According to the lemma 1, there are at most $t_d \cdot d^k$ patterns for $\mathcal{I}$. Iterating from one pattern to the next one takes at most $d+k$ operations. For each pattern P, the arrays C_v and D_v are computed in time $(n^2+n) \cdot ch(v)$ for each node v using equations (1) and (2). As P contains $d+k$ arcs, this algorithm takes at most $(n^2+n) \cdot (d+k)$ operations. The final procedure in Line 4 is applied recursively on each node only once. At the same time, it builds the solution T in Line 5. The comparison in Line 6 is made in constant time.

At each iteration, the algorithm needs the shortest paths graph of size n^2, the solution T_0 of size at most $d+k$ and its cost, the current pattern P, and for each node v of P, the arrays C_v and D_v of size $n \cdot (ch(v)+1)$. Thus the pattern algorithm runs in time $O(t_d \cdot d^k \cdot n^2 \cdot (d+k))$ and space $O(n^2 + n \cdot (d+k))$. □

Theorem 8. *If $\mathcal{I}$ is a DST instance, the pattern algorithm runs in time $O^*(1 \cdot 3 \cdot 5 \cdots 2k-3)$.*

PROOF. If $\mathcal{I}$ is a DST instance, $d = k-1$. By Lemma 2, there are $1 \cdot 3 \cdot 5 \cdots 2k-3$ patterns for $\mathcal{I}$, reducing the running time to $O^*(1 \cdot 3 \cdot 5 \cdots 2k-3)$. □

4. Conclusions and perspectives

We have proposed an algorithm running in polynomial space and in FPT time with respect to k and d for DSTLD. This gave the first FPT algorithm in k for the Directed Steiner Tree problem in polynomial space, building and returning an optimal solution. However, considering the multicast application, the current running time complexities of our algorithms are acceptable for small networks. Table 1 suggests it is hard to overcome the $O^*(2^k)$ running time complexity for DST. A challenging problem would be to reduce the exponential part of our algorithms from $O^*(k^k 2^k)$ to $O^*(2^k)$.

Furthermore, our pattern algorithm could give approximation algorithms if we were able to select a polynomial part of interesting patterns, on which we would apply our dynamic programming algorithm.

[1] BJÖRKLUND, A., HUSFELDT, T., KASKI, P., AND KOIVISTO, M. Fourier meets möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing* (2007), ACM, pp. 67–74.

[2] Cheng, X., and Du, D.-Z. *Steiner trees in industry*, vol. 11. Springer, 2001.

[3] Dreyfus, S. E., and Wagner, R. A. The steiner problem in graphs. *Networks 1*, 3 (1971), 195–207.

[4] Flajolet, P., and Sedgewick, R. *Analytic combinatorics*. cambridge University press, 2009.

[5] Fomin, F., Grandoni, F., Kratsch, D., Lokshtanov, D., and Saurabh, S. Computing optimal steiner trees in polynomial space. *Algorithmica 65*, 3 (2013), 584–604.

[6] Fuchs, B., Kern, W., Molle, D., Richter, S., Rossmanith, P., and Wang, X. Dynamic programming for minimum steiner trees. *Theory of Computing Systems 41*, 3 (2007), 493–500.

[7] Gargano, L., Hammar, M., Hell, P., Stacho, L., and Vaccaro, U. Spanning spiders and light-splitting switches. *Discrete Mathematics 285*, 1–3 (2004), 83 – 95.

[8] Karp, R. M. *Reducibility among combinatorial problems*. Springer, 1972.

[9] Nederlof, J. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica 65*, 4 (2013), 868–884.

[10] Novak, R., Rugelj, J., and Kandus, G. A note on distributed multicast routing in point-to-point networks. *Computers & Operations Research 28*, 12 (2001), 1149–1164.

[11] Reinhard, V., Cohen, J., Tomasik, J., Barth, D., and Weisser, M.-A. Optimal configuration of an optical network providing predefined multicast transmissions. *Comput. Netw. 56*, 8 (May 2012), 2097–2106.

[12] Reinhard, V., Tomasik, J., Barth, D., and Weisser, M.-A. Bandwidth optimization for multicast transmissions in virtual circuit networks. In *NETWORKING 2009*, L. Fratta, H. Schulzrinne, Y. Takahashi, and O. Spaniol, Eds., vol. 5550 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 859–870.

[13] Rémy, J.-L. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Informatique théorique 19*, 2 (1985), 179–195.

[14] Voss, S. Steiner tree problems in telecommunications. In *Handbook of optimization in telecommunications*. Springer, 2006, pp. 459–492.

[15] Watel, D., Weisser, M.-A., Bentz, C., and Barth, D. Directed steiner tree with branching constraint. In *Computing and Combinatorics*, Z. Cai, A. Zelikovsky, and A. Bourgeois, Eds., vol. 8591 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 263–275.