On the Minimal Steiner Tree Subproblem and Its Application in Branch-and-Price

Wilhelm Cronholm[1], Farid Ajili[1], and Sofia Panagiotidi[2]

[1] IC-Parc, Imperial College London, London SW7 2AZ, UK
{w.cronholm, f.ajili}@imperial.ac.uk
[2] London e-Science Centre, Imperial College London, London, SW7 2AZ, UK
sp1003@imperial.ac.uk

Abstract. The minimal Steiner tree problem is a classical NP-complete problem that has several applications in the communication and transportation sectors. It has recently emerged as a subproblem in decomposition techniques such as column generation and Lagrangian schemes. This has set new computational challenges to the state of the art solving approaches. Our goal is to improve on existing branch-and-cut algorithms so that our approach successfully serves as a fast subproblem solver in a decomposition context. Compared with existing literature, our technical contributions include 1) a new preflow-push cutting strategy, revisiting a little known graph algorithm, that halves the runtime of the separation step, and 2) a branching scheme that fairly balances the search tree and speeds up the search. An evaluation in a multicast design application shows that the algorithm enhances a column generation hybrid. Moreover, our approach offers a significant speedup factor on a publicly available set of challenging Steiner tree benchmarks.

Keywords: networks, preflow-push algorithms, branch-and-cut, Steiner trees.

1 Introduction

The minimal Steiner Tree Problem (STP) consists of determining the least-cost tree that connects a given set of nodes in a graph. This NP-complete problem was originally formulated by Hakimi [10]. Since then, it has received considerable attention in the literature as it has a wide range of applications, e.g. network optimisation, distribution systems and VLSI layout [10, 18]. It is beyond the scope of this paper to cover all the STP literature. Surveys can be found in e.g. [10, 18]. Although useful in its own right, several network design models embed the STP as a substructure. It then emerges as a *subproblem* in decomposition algorithms such as column generation or Lagrangian relaxation.

If the aim is to compute fairly good solutions to the *stand-alone* Steiner tree problem within a reasonable time, then it has been recommended to use a variety of greedy "tree heuristics" [10, 18]. However, if the STP appears as a subproblem in a branch-and-price setting [6] or Lagrangian context [5, 19],

R. Barták and M. Milano (Eds.): CPAIOR 2005, LNCS 3524, pp. 125–139, 2005.

existing algorithms require exact STP solvers that deliver an optimal tree. In branch-and-price for instance, finding the most profitable tree not only offers the best pricing strategy, but also enables exploiting the lower bound and achieving early termination and cost-based filtering at *any* column generation iteration. This is because the column generation bound plus the subproblem reduced cost is a valid lower bound even at *non-optimal* iterations.

Several authors have proposed approaches for solving the minimal STP, including Lagrangian relaxation [10] and branch-and-cut [3, 12, 15]. Previous empirical evaluation has suggested that the fastest approach is the branch-and-cut solver of Koch & Martin [12]. The authors showed that their solver is superior to other approaches, as it succeeded in solving almost all existing benchmarks for the first time. Several decomposition methods to solve the multicast network design have embedded [12] as a subproblem solver, e.g. [6] deploys column generation and [5, 19] exploit a special Lagrangian scheme.

A necessary ingredient for a decomposition method to be sufficiently competitive is an efficient subproblem solver. However, we have noticed that the hybrid branch-and-price algorithm of [6] spends nearly two thirds of the runtime in solving the Steiner tree subproblems in the pricing phase. Our performance analysis has demonstrated that the separation (i.e. cutting) phase of the branch-and-cut solver [12] is the largest computational bottleneck since the algorithm spends at least half of its runtime in identifying the inequalities whose addition cuts off the relaxed solution.

Another drawback of existing literature is that most exact STP algorithms, [3, 12, 15] included, use traditional Integer Programming (IP) branching on single arc variables. This is not well-suited, as it potentially leads to an unbalanced (binary) search tree: the weak branch forbidding an edge often barely changes the relaxation.

This paper proposes an innovative and non-trivial variation of branch-and-cut STP algorithms that successfully serves as a subproblem solver in decomposition methods. The solver also substantially improves the best known runtimes of a publicly available set of benchmarks as a stand-alone Steiner tree solver.

The contents of the paper are as follows. Section 2 outlines related work and our contributions. Section 3 overviews our branch-and-cut solver, in particular an enhanced branching scheme and a new cost-based filtering rule that exploits reduced costs. Section 4 discusses some separation issues and proposes a new cutting strategy. Section 5 presents a computational discussion of our approach under two use cases 1) run it as a stand-alone STP solver, 2) invoke it as a branch-and-price subproblem. Section 6 concludes the paper.

2 Related Work and Overview of Contributions

Exact algorithms for the minimal Steiner tree problem include Lagrangian relaxation and branch-and-cut, e.g. [10]. Because [12] is considered the most efficient algorithm for solving STP to optimality, multicast network design approaches [5, 6, 19] exploiting decomposition methods have all used it. In [6] it is reported that the invocations of [12] are by far the largest bottleneck on some challenging multicast design datasets. We also note that in the results of [19] around 45 minutes are spent at the root node of the search tree, even for quite small networks and reasonable number of multicast groups. This is likely spent in solving STP subproblems.

Most Steiner tree solvers rely heavily on reduction techniques [10, 12, 18] to reduce graph size (i.e. edge set and vertex set). These act as a preprocessing step that includes (resp. excludes) edges/nodes that do (resp. not) belong to at least one minimal Steiner tree.

The preprocessing, which often requires careful implementation, exploits special configurations in the graph. Since those tests often change the set of edges and/or nodes, they are difficult to accommodate in a decomposition-based search as they conflict with the much needed incrementality of the subproblem solver: we may have to re-setup a subproblem model/solver as soon as the graph changes. Indeed, some changes, such as node contraction, could result in deleting and adding variables and make several constraints invalid. In the decomposition methods [5, 6, 19], applying "non-logical" reduction tests every time the subproblem is solved can introduce a substantial overhead as the constraint part of the subproblem is kept unchanged throughout the search: only the cost vector changes.

Painful reduction techniques are not included in our algorithm. Because we are losing their strength, it is crucial to somehow compensate for this. It is done through 1) designing a new separation phase by carefully merging an algorithm proposed in [9] to compute a generalised form of a cut in a graph, 2) introducing a harmless cost-based pruning rule that reasons about optimality to exclude nodes from the solution. In fact, our focus on speeding up the separation phase began when our experimentation clearly revealed that it is very costly.

Despite the efficiency of [9] for computing the minimum unrestricted cut [14], this algorithm is not widely known in the literature. Koch & Martin [12] briefly mention that [9] is modified and used at the separation phase of their STP solver. However, they gave no details of how the algorithm is to be used within their solver and failed to provide a computational evaluation whether such an approach is beneficial. Moreover, recent publications [18, 19] do not seem to be aware of the idea of using [9].

From this regard, our contributions are two-fold. We demonstrate that using [9] as a black-box, or through a trivial adaptation, does not work and identify a straightforward counter-example. Our technical contribution is to detail a careful adaptation of [9]. In fact, the resulting preflow-push cutting strategy could be used in other areas such as survivable networks [11] and the travelling salesman methods, e.g. [16].

Strengthening the cutting strategy of the branch-and-cut was useful in enhancing the performance of a a branch-and-price method even with "easy" STP instances. Our evaluation in the stand-alone setting showed that the speedup factor is much clearer as soon as the instances get harder.

Yet another serious drawback of existing branch-and-cut approaches lies in the branching itself. Our runs showed that branching on a binary variable in the integer programming fashion tends to create search branches that cause unbalanced changes on the linear relaxation. To overcome this, we suggest a new branching scheme that leads to branches potentially having the same likelihood of containing the best solution.

3 Branch-and-Cut Approach

In this section we present our improved branch-and-cut approach. It builds on top of previous literature such as Koch & Martin [12]. We first introduce some notation and definitions.

Let $G = (V, A)$ be a directed graph, where V is its node set and A is its set of arcs. Let $n = |V|$ and $m = |A|$. An arc in A from node i to node j, is denoted (i, j). If H is a subset of nodes, then we denote by $\bar{H} = V \setminus H$ its set complement in V. A cut $(H, \bar{H})$ is a non-trivial partition of V. An $S - t$ cut is a cut $(H, \bar{H})$ such that $S \subseteq H$ and $t \in \bar{H}$. If $S = \{s\}$ then we use the shorter notation $s - t$ cut. Usually H is called the *source side* and $\bar{H}$ the *sink side* of the cut. The set of arcs in the cut is denoted by $\delta(H, \bar{H})$, so $\delta(H, \bar{H}) = \{(i, j)|i \in H, j \in \bar{H}\}$. If w is a set of weights attached to the arcs A we denote by $w(H, \bar{H})$ the weight of the cut, i.e. $w(H, \bar{H}) = \sum_{a \in \delta(H, \bar{H})} w_a$. The minimum $s - t$ cut problem is that of finding the minimum weight $s - t$ cut.

The minimal STP consists of determining the least-cost tree in a graph, that connects a given set of nodes. We use a similar *directed* version instead since it is known to give a tighter bound [4]. The directed version is named the *Steiner arborescence* problem. The transformation from the undirected Steiner tree problem to the directed problem is simple: replace, in the obvious way, every edge by two directed arcs with the same cost and select one terminal as the root. It is well known that this does not alter the solution set. For convenience we shall refer to an arborescence with the more general term tree throughout this document.

The Steiner tree problem can be formulated as follows: given a directed graph $G = (V, A)$, non-negative weights $w : A \to \mathbb{R}^+$, a non-empty subset $T \subseteq V$ of terminals, and a root $r \notin T$, find a directed subset of arcs R such that there is a path from r to every terminal and $\sum_{a \in R} c_a$ is minimised. A node not in $T \cup \{r\}$ is called a *nonterminal.*

We introduce decision variables: let x_a be a binary variable that is 1 if and only if arc a is used in the Steiner tree.

The solver uses a standard branch-and-cut approach outlined in Figure 1. The inner loop is a separation procedure which finds violated inequalities or proves that none exits. The algorithm calls the reduction technique described in Section 3.3. The LP is a continuous relaxation of the cut formulation of a tree. It is initialised with the flow inequalities described in [12]. The search separates Steiner cuts and adds them to the relaxation. A *Steiner cut* $(H, \bar{H})$ is a cut that separates the root r from *at least* one terminal: $r \in H$ and there exists a $t \in T$ such that $t \in \bar{H}$. To each Steiner cut $(H, \bar{H})$ in G is associated the following inequality:

```
Branch-and-Cut:
apply reduction techniques
initialise search
repeat
   select a leaf from the tree and consider the associated LP
   repeat
      solve the LP relaxation
      separate violated inequalities and add them to the LP
   until there are no violated inequalities
   branch if necessary otherwise remove the leaf from the tree
until there are no leaves in the branch-and-bound tree
```

Fig. 1. A general branch-and-cut algorithm

$$\sum_{a \in \delta(H,\bar{H})} x_a \geq 1 \quad . \tag{1}$$

The Steiner cut inequalities (1) ensure reachability from the root to every terminal. At each iteration of the separation algorithm described in Section 3.1, the LP-relaxation suggests a value $\bar{x}_{ij}$ for every variable x_{ij}. These values are then used to identify violated inequalities to be added to the LP to cut off the current solution. If no violated inequality exists the search branches as in Section 3.2.

3.1 Separation of Violated Inequalities

The separation algorithm is the most critical step in branch-and-cut. It finds, if any, relaxed inequalities (1) violated by the suggested solution $\bar{x}$ of the LP relaxation. The inequalities (1) can be separated as follows: for each $t \in T$ find the minimum cost $r-t$ cut in G using $\bar{x}$ as weights. This is commonly done with any max-flow/min-cut algorithm, see e.g. [1]. If, for some t, the cost w^*_{r-t} of the minimum $r-t$ cut in G is below 1 then the associated inequality is violated. If w^*_{r-t} exceeds 1 for each terminal t, then there is no violated inequalities and the search branches. Inequalities are added only if they are violated by a small tolerance. In fact, we use the "creep-flow" strategy [12] that favours least-cost $r-t$ cuts having minimum number of arcs. This makes the separation harder, but in practice is compensated for by the strength of the inequalities.

Note that all Steiner cuts are found in the input graph G. That is because the graph is not affected by the branching decisions of Section 3.2. Therefore the cuts generated are valid throughout the branch-and-cut search.

Section 4 describes a fast separation algorithm by adapting [9]. It considerably enhances the performance of the branch-and-cut search.

3.2 Branching Strategy

Branching is an essential part of the exact algorithm and has to be carefully designed to reach optimal performance. There are several requirements on the branching strategy. In particular it has to:

- substantially affect the relaxation and the LP cost
- yield a balanced search tree, with equal likelihood of the best solution being in the different branches.

In [12] IP branching is used; it branches on a single arc variable. Disallowing an arc is unlikely to change the problem significantly since most arcs will not be in the optimal tree anyway. On the other hand, including an arc has a larger impact on the LP since it severely restricts the number of feasible (shortest) trees. This results in an unbalanced search tree in practice.

To overcome this difficulty we apply a new hierarchical branching strategy. It commits decisions to several variables by first focusing on node membership. This is motivated by the fact that the difficulty in the STP is to decide which nodes are to appear in the minimal tree. Vertex oriented branching is also mentioned in [10] and used in [18]. However, it has not been used in the context of branch-and-cut.

For each node v, its "likelihood" h_v of being a member of the minimal Steiner tree is estimated by $h_v = \sum_{(j,v)\in A} \bar{x}_{(j,v)}$. In a primal solution, h_v is either 0 or 1. The nonterminal having the maximal integrality violation of h_v is selected, i.e. the node v that minimises $|0.5 - h_v|$ over the nonterminals for which $1 > h_v > 0$. The branching first includes v and excludes v on backtrack. It is accomplished by posting the following inequality with $b = 1$ on the forward branch and $b = 0$ on backtrack:

$$\sum_{(j,v)\in A} x_{jv} = b \quad . \tag{2}$$

The exclusion effectively disallows all arcs having one end in v as a result of flow inequalities [12]. Note that the branching decision is easily incorporated in the LP since each branching decision is the same as adding a cut. Clearly both branches cut off the relaxed solution $\bar{x}$ provided that there exists a nonterminal v such that h_v is fractional.

Whenever the node branching does not apply, $\bar{x}$ is often primal feasible. However, in rare cases it might not be. An example is when there are several optimal Steiner trees spanning the same nodes but crossing different arc sets. In such a case, we resort to IP branching for completeness by branching on the arc variable x_a that minimises $|0.5 - \bar{x}_a|$ over the set of fractional $\bar{x}_a$. The search sets $x_a = 1$ and $x_a = 0$ on backtrack.

3.3 A New Cost-Based Reduction Technique

Preprocessing that alters the graph is difficult to implement when the STP is a subproblem. However, there are still harmless reductions that can be exploited in a decomposition context. Here, the focus is on a reduction technique that only removes nodes/arcs that can not participate in *any* optimal Steiner tree. This type of reduction can be easily encoded by setting some arc variables to zero in the LP.

We now introduce a new reduction rule that exploits reduced costs extracted from solving the linear programming relaxation. We observed that applying standard reduced cost fixing is weak as the cut formulation of a tree is known to be highly degenerate. This same observation is supported by the results of [12].

Our technique is stronger than the standard reduced cost fixing. Its design has been inspired by our previous work in strengthening optimality reasoning in network routing [7]. We exploit the additivity of the reduced costs and the tree structure to infer an estimation of the cost incurred on the relaxation lower bound LB by including a node into the Steiner tree.

The incumbent cost of the branch-and-cut is denoted by UB. Let r_a denote the reduced cost of variable x_a; further let $r_a^+ = \max\{0, r_a\}$. Consider a nonterminal node v that the relaxed solution suggests not to be in the optimal tree, i.e. $h_v = 0$. If v was to be included in the minimal Steiner tree (i.e. the value of h_v switches from 0 to 1), then some path p from the root r to v must exist in any feasible tree. All arcs a in p such that $\bar{x}_a = 0$ have to be included. The cost of including a is r_a (resp. 0) if $\bar{x}_a = 0$ (resp. $\bar{x}_a > 0$). Thus, the incurred cost of ensuring x_a is 1 is r_a^+. The incurred cost of including the path p is the sum of r_a^+ for each a in p. A valid under-estimate of that is the length, say $\psi_v(r)$, of the shortest path from r to v in the graph where the weight of an arc a is r_a^+. This is captured by the following reduction test:

$$\texttt{if} \quad LB + \psi_v(r) \geq UB \quad \texttt{then} \sum_{(j,v)\in A} x_{jv} = 0 \tag{3}$$

This rule can be used every time the relaxation is re-optimised. The effect of (3) can be enhanced by using a CP-relaxation [5]. This would enable the inference of even more fixings. We have used the same constraint programming store as have been used in [5]. It makes some significant inference at the root, but less fixings afterwards. The reduction rule (3) has not been extensively tested and was therefore switched off in our experimentation.

4 A More Efficient Separation Algorithm

This section describes a new separation algorithm for the branch-and-cut Steiner tree solver. Normally, as described in Section 3.1, the separation is done by solving $|T|$ min-cut problems *independently.* Here we adapt ideas presented by Hao & Orlin [9] to be able to solve the separation problem much more efficiently in one go. The speed up is gained by re-using information from previous min-cut computations.

The rest of this section first briefly presents the Hao & Orlin algorithm (HO) in Section 4.1 and then explains in Section 4.2 the adaptation needed to find Steiner cuts.

4.1 Overview of the Hao and Orlin Algorithm

This section briefly describes HO [9]. It is assumed the reader is familiar with the preflow-push algorithm, first presented by Goldberg & Tarjan [8] to some extent. For general graph topics we refer the reader to [1]. Most of the materials in this section are from [9].

HO is a modified version of the algorithm by Goldberg & Tarjan [8]. It finds the minimum cut in a directed graph where only a set S of sources are specified to be in the source side. We denote this by a $S - *$ cut.

The unrestricted min-cut problem is to find the minimum weight cut in the graph without restrictions on any nodes. It has many applications in network reliability and is useful as a separation algorithm for the travelling salesman problem. This can easily be solved by $2(n-1)$ $s-t$ cut computations. However, for a directed graph, the unrestricted min-cut problem can be solved by two invocations of HO, the second where all arcs are reoriented. The total runtime is comparable to the time of one $s-t$ computation.

Recall that the preflow-push algorithm works with a *preflow*. A preflow is a flow except that the entering flow of a node v can exceed the leaving flow v. The *excess* is the difference between the last two. The preflow-push algorithm pushes flow from *active* nodes, nodes with positive excess, along estimated shortest paths in the *residual graph*, the "remaining flow graph". The shortest path estimation is done by *distance labels* of the nodes. The distance label of a node is a lower bound on the length of a path from the node to the sink in the residual graph. An overview of the preflow-push algorithm can be found on the right-hand side of Figure 2. During a min-cut computation of the preflow-push algorithm it is first assumed that all nodes, apart from the source node, are on the sink side W of the cut, but as the computation progresses nodes are transferred to the source side D. Specifically they are moved to D when there is no longer a path from the node to the sink in the residual graph. The algorithm maintains the invariant that there is no arc of the residual graph directed from any node in D to any node in W.

	find-min-cut(G, w, s):		**preflow-push**(G, w, S, t):
	$S = \{s\}$, $Best = \infty$	(iii)	initialise
	repeat		**repeat**
(i)	select a node $t' \in V \setminus S$	(iv)	select an active node i
(ii)	find minimal $S - t'$ cut $(H^*, \bar{H}^*)$		push/relabel (i)
	$z = w(H^*, \bar{H}^*)$		**until** there are no active nodes
	if $z < Best$ **then**		**return** $(H, \bar{H})$
	$Cut = (H^*, \bar{H}^*)$, $Best = z$		
	endif		
	add t' to S		
	until $S = V$		
	return $(Cut, Best)$		

Fig. 2. Outline of the Hao and Orlin algorithm for computation of the minimum $S - *$ cut

HO exploits the preflow-push algorithm to find the least-cost $S - *$ cut. An overview can be found on the left-hand side of Figure 2. The preflow-push algorithm is invoked at (*ii*). The faster runtime of HO stems from the reuse of information from min-cut computations: the ending state of the last preflow-push computation is reused (at (*iii*)). Also, information from the min-cut computation is used to force an ordering in the selection of sinks at (*i*). Specifically, the distance labels are re-used, as well as information about nodes that have been transferred to the source side. These nodes are divided into "dormant" layers according to the time they where moved to the source side. These nodes will be moved back, "awakened", in reverse order at a later stage so all nodes will be selected at (i) during the execution. The new sink is selected as the node in the source side of the last computation with the smallest distance label. If there is no awake node then a layer of dormant nodes are woken up and the selection procedure is repeated. The algorithm terminates when all nodes are in S.

We now present the problems with adapting HO to the context of finding violated inequalities of form (1). In the light of that, we derive a sound adaptation.

4.2 Adapting HO for Computing Steiner Cuts

First, HO cannot be used as it is, since the cut found is not guaranteed to be a Steiner cut. Indeed, it is likely that the minimum cut will have weight 0 as is the case when the sink side consists of a set of nonterminals that are not suggested should be used. Clearly, this is not a Steiner cut.

A trivial modification is to only consider cuts that are found whenever the sink is a terminal. However, this does not guarantee finding a violated cut if there is one [17]. The reason is that the algorithm does not necessarily find the minimal $s-t$ cut when t is the sink. Instead, properties of the algorithm certify that the minimal $s-t$ cut has been found in previous iterations, if the current cut is not the one sought after. However, when considering Steiner cuts these properties do not hold since there is a difference between terminals and nonterminals. The example below illustrates this. In fact, it is not enough to consider *all* Steiner cuts generated during the execution of the algorithm since the optimal cut can be missed by moving a nonterminal to S too early.

Example 1. Consider Figure 3. Node s is the root, t is a terminal and v is a nonterminal. During the execution of HO, v is selected as the first sink. The minimal $r-v$ cut is found, which is $(\{s,t\},\{v\})$ with a weight of 3. However, it is not a valid Steiner cut since there is no terminal on the sink side. Next, v is moved to the set of sources S and t is selected as the next sink. In this iteration the cut $(\{s,v\},\{t\})$ is returned with a weight of 13. The cut is a valid Steiner cut since t is on the sink side. It is also the minimal Steiner cut found by the algorithm. However, as can easily be seen in Figure 3, the optimal Steiner cut is $(\{s\},\{t,v\})$ with a weight of 4. This cut was missed since v was moved into S too early and the cut found when v was sink was not a Steiner cut. ■

It follows from Example 1 that in order for the adaptation to work we need to make sure that every cut found is a Steiner cut. This is ensured by only selecting terminals as sinks at step (i). Then every minimum cut found is a valid Steiner cut. However, there is a problem with this approach. The problem relates to the distance labels. Recall that preflow-push algorithms use distance labels of the nodes to approximate the minimum number of arcs to reach the sink. When a terminal is selected as sink it might not have the minimum distance label, so other awake nodes may have smaller distance labels. This is a problem for two reasons.

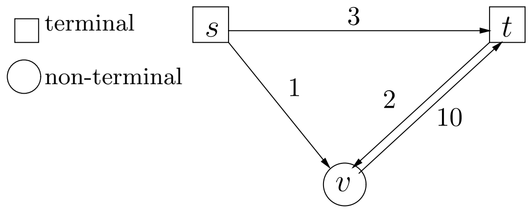

Fig. 3. Counter-example showing how the Hao and Orlin algorithm could miss the optimal Steiner cut. Arcs are labelled with costs

First, it interferes with the widely used and efficient gap-relabelling heuristic [2], which is included in the relabel procedure in [9]. The heuristic moves nodes that are known to be disconnected from the sink into the dormant set by detecting "gaps" in the distance labels. This is much quicker than the naive solution which only moves nodes to the dormant set as the distance label increases above a limit (usually n). However if nodes are allowed to have a smaller distance label than the sink, a gap does not guarantee that nodes are disconnected.

Second, the number of nodes that are not in the source set will be larger than the difference between n and the distance label of the current sink, if it is kept unchanged. This means that the aforementioned limit is hard to establish. The result is a bad performance since efficient data structures as well as the runtime complexity rely on a tight bound on the limit, see [17].

The interference with the gap-relabelling is easy to handle by taking the distance label of the current sink into account. However, the second problem still remains. We overcome both difficulties by setting the distance label of the new sink to 0 and resetting all the awake nodes' distances by the exact distance, in terms of number of arcs, to the new sink in the residual graph. This is called global relabelling and is an important feature in competitive preflow-push algorithms [2].

To speed up the computation even more, we always stop and select a new sink whenever we detect that the minimum cut for the current sink will not be the minimum one, for instance when we have already found a smaller cut. Also, we store all violated sub-optimal cuts found throughout the iterations, since they may benefit the LP.

To summarise: our change to HO is that we select the terminal in the awake set with the minimum distance label. We then reset the distance labels for that node to 0 and recalculate the exact distances for all other awake nodes. We transfer nodes that do not have a residual path to the new sink into the dormant set. If necessary we wake up layers from the dormant set when selecting a new sink until there is a terminal in the awake set, even if there are non-terminal nodes in the awake set.

It is easy to see that the algorithm is still correct with this modification. The dormant nodes do not play a role in finding the current minimum cut so changing the distance labels of the awake nodes does not alter the correctness. Also, whenever nodes are woken up from the dormant set, they are given a new distance label. We note that we might have to wake up several layers before a terminal is found.

5 Experimental Results

This section presents a computational discussion of our approach under two use cases 1) in a stand-alone context and 2) invoked in a branch-and-price solver. The aim is to assess the contribution of the new branching strategy and the improved separation. The techniques have been implemented in the Constraint Logic Programming language ECLiPSe.

We evaluate four parameter settings: `n-ff`, `n-ho`, `a-ff` and `a-ho`, where the first letter of the parameter setting describes the branching strategy: IP branching (`a`) or our branching (`n`). The second part describes the separation strategy: $|T|$ max-flow/min-cut computations (`ff`) or our strategy (`ho`).

5.1 Evaluation in Stand-Alone Context

Results for 42 instances from SteinLib [13] are reported. SteinLib is a set of publicly available benchmarks for the Steiner tree problem. The results can be seen in Table 1. SP-t designates the total separation time in seconds, LP-t is the total LP time in seconds, #N presents the size of the search tree and the upper bound UB is labelled with a star if the instance was proved optimal. First, we compare the contribution of the new separation strategy. Table 1 reveal that our cutting strategy is clearly superior. Indeed, the results show that the `ho`-strategy always enables significantly faster optimality proofs. Compared to `a-ff`, the improved algorithm `a-ho` finds a better primal solution on 2 instances, proves optimality on 2 instances when `a-ff` fails to do so. It also proves optimality on 1 instance when `a-ff` fails to find the optimal solution at all. In fact, the `n-ho` strategy manages to significantly improve the best known runtime on one of the most challenging instances (`bipe2u`) in SteinLib. The latter instance was only recently solved to optimality by [18]. It is not surprising that the `ho` strategy requires more separation iterations since it only generates the most violated Steiner cuts. The `ff` strategy generates almost one $r-t$ cut for each $t \in T$ at every iteration. However, the majority of the latter cuts are useless because for those examples where both cut strategies proved optimality `ho` required considerably less cuts in total. This supports the view that the most violated cut is the most beneficial.

The `ho` strategy more than halves the runtime in some cases. The faster separation enables `ho` to prove optimality quicker or explore a larger part of the search tree. This is especially apparent for the instances with a large number of terminals, suggesting that the gain from the `ho` strategy is clearer as the number of terminals increases.

We now turn our attention to the contribution of the branching. The new branching enables `n-ho` to prove optimality of 3 instances where `a-ho` fails to do so.

Table 1. Results for some SteinLib instances

	a-ff							a-ho							n-ff							n-ho						
Test	UB	Time	SP-t	LP-t	#cuts	#iter	#N	UB	Time	SP-t	LP-t	#cuts	#iter	#N	UB	Time	SP-t	LP-t	#cuts	#iter	#N	UB	Time	SP-t	LP-t	#cuts	#iter	#N
bipe2p	5766	5011	4916	22	137	6	2	5661	4923	2768	391	637	1250	885	5851	5008	4768	21	131	6	2	5653	4940	1706	647	561	744	460
bipe2u	55	5011	4796	23	142	6	2	54	4954	1258	3538	698	398	19	55	5009	4382	19	96	5	2	54*	**2571**	937	786	445	326	85
cc3-4p	2338	4978	1272	2061	8501	4877	3023	2338	4933	572	2372	7094	6700	3634	2338	4961	1537	2778	11251	3497	1600	2338	4915	515	3572	8110	5409	2165
cc3-4u	23*	1937	1551	1090	4864	818	57	23*	1038	179	765	3388	1251	109	23*	1255	1282	578	4295	722	43	23*	632	137	460	3105	987	21
cc3-5p	3676	4980	3400	1527	3242	471	6	3669	4939	766	3250	3435	1056	84	3681	4980	3551	1370	3481	488	5	3670	4912	831	3883	3946	1154	29
cc3-5u	36	4980	2944	1982	3225	431	1	36	4884	1151	3601	4938	1461	1	36	4980	2979	1954	3285	446	1	36	4865	1154	3586	4988	1487	1
cc6-3p	21608	5005	4510	73	599	9	1	21134	4885	4281	522	428	85	1	21608	5003	4467	71	599	9	1	21134	4863	4289	524	436	87	1
cc6-3u	206	5004	4791	140	749	11	1	209	4905	3486	1362	325	82	1	206	5002	4743	134	749	11	1	209	4897	3486	1342	337	84	1
hc6p	4013	4981	4637	134	38390	8009	5126	4011	4758	3483	413	54478	55432	23647	4003*	3992	3732	146	42917	4918	1919	4003*	672	511	65	9901	7385	1767
hc6u	39	4983	4660	200	37539	4526	1496	39*	1511	1155	172	17573	15015	3143	39*	1749	1660	61	14936	1733	309	39*	283	221	34	3618	2762	305
hc7p	7944	4989	4448	297	11131	641	334	7938	4808	2364	983	10691	11228	6629	7905	4977	4516	313	11975	702	224	7905	4761	3234	1016	20941	12653	3109
hc7u	77	4984	4325	437	10101	318	22	77	4766	1937	2224	13158	6649	509	77	4982	4540	308	8724	346	9	77	4784	2084	2360	14794	8385	496
160-041	1494*	50	41	3	41	8	1	1494*	16	6	4	24	11	1	1494*	50	41	3	41	8	1	1494*	15	6	4	24	11	1
160-042	1486*	34	27	1	17	4	1	1486*	9	2	1	10	4	1	1486*	34	27	1	17	4	1	1486*	9	2	1	10	4	1
160-043	1549*	901	833	43	248	59	13	1549*	109	49	38	164	83	13	1549*	1162	1056	72	287	67	17	1549*	119	48	47	150	82	11
160-044	1478*	10	3	1	6	2	1	1478*	7	0	2	2	2	1	1478*	10	3	1	6	2	1	1478*	7	0	1	2	2	1
160-045	1554*	481	446	18	186	43	9	1554*	85	47	22	138	68	7	1554*	443	405	18	186	44	11	1554*	84	45	21	132	68	9
160-111	2869*	18	16	1	66	7	1	2869*	7	5	1	31	11	1	2869*	18	16	1	66	7	1	2869*	7	5	1	31	11	1
160-112	2924*	182	166	10	295	41	11	2924*	56	40	11	130	51	11	2924*	153	143	8	264	27	3	2924*	64	49	10	168	57	7
160-113	2866*	49	46	2	128	13	1	2866*	28	24	3	97	29	1	2866*	48	45	2	128	13	1	2866*	28	23	3	97	29	1
160-114	2989*	177	167	8	280	27	1	2989*	69	55	11	191	60	1	2989*	175	165	7	280	27	1	2989*	67	54	10	191	60	1
160-115	2937*	1241	970	214	1304	183	59	2937*	638	319	242	1081	354	55	2937*	579	418	130	673	96	33	2937*	360	163	164	634	206	37
160-141	2549*	307	287	13	110	11	1	2549*	63	36	19	76	28	1	2549*	302	282	12	110	11	1	2549*	63	36	19	76	28	1
160-142	2562*	1272	1228	31	274	28	3	2562*	178	110	50	237	66	3	2562*	1113	1070	27	252	28	5	2562*	175	109	49	237	66	3
160-143	2557*	194	180	7	76	8	1	2557*	32	17	8	43	12	1	2557*	190	177	76	8	1	1	2557*	31	16	8	43	12	1
160-144	2617	5000	4660	282	737	72	5	2607*	1457	569	707	943	261	15	2607*	4155	3742	384	649	66	7	2607*	1523	561	834	924	260	15
160-145	2578*	782	746	27	209	20	1	2578*	114	69	35	141	41	1	2578*	768	732	26	209	20	1	2578*	113	68	34	141	41	1
160-211	5583*	4520	3780	609	2990	182	29	5583*	1877	1106	671	2259	534	21	5583*	3357	2763	506	2476	137	21	5583*	1369	810	513	1817	407	13
160-212	5643	4993	3724	998	3049	298	95	5643	4882	2206	2230	4046	1054	133	5643	4991	3522	1373	3350	225	49	5643*	3516	1394	1993	3056	683	51
160-213	5647*	2730	2341	345	2217	139	23	5647*	1210	733	426	1433	355	25	5647*	2281	1984	272	1724	110	13	5647*	1471	767	657	1593	399	29
160-214	5720	4994	3831	961	3530	258	64	5720*	3505	1747	1528	3514	867	75	5720	4991	3627	1177	3173	213	47	5720*	4211	1717	2237	3905	924	73
160-215	5518*	1357	1273	69	1189	60	3	5518*	578	446	114	909	211	3	5518*	1278	1217	55	1144	58	3	5518*	551	433	100	904	210	3
320-011	2053*	242	225	11	174	29	3	2053*	77	52	17	146	44	3	2053*	249	233	11	181	30	3	2053*	77	52	16	145	44	3
320-012	1997*	14	10	1	21	5	1	1997*	6	2	1	7	4	1	1997*	14	10	1	21	5	1	1997*	6	2	1	7	4	1
320-013	2072*	292	277	10	161	24	1	2072*	116	89	20	192	64	1	2072*	286	272	9	161	24	1	2072*	114	88	18	192	64	1
320-014	2061*	1092	1021	53	446	72	7	2061*	324	228	78	475	145	5	2061*	525	468	47	320	53	7	2061*	192	131	48	279	99	7
320-015	2059*	754	692	49	355	59	7	2059*	188	121	56	263	91	5	2059*	517	478	31	272	45	5	2059*	192	108	68	247	86	7
320-041	1707*	923	812	54	76	13	1	1707*	175	57	57	47	18	1	1707*	914	804	54	76	13	1	1707*	169	55	56	47	18	1
320-042	1682*	242	168	21	19	4	1	1682*	90	12	21	11	5	1	1682*	239	166	21	19	4	1	1682*	85	11	21	11	5	1
320-043	1723	5040	4868	126	224	62	13	1723*	579	302	160	223	101	7	1726	5039	4703	114	203	38	5	1723*	630	282	222	220	94	7
320-044	1681*	314	238	23	26	5	1	1681*	107	20	28	15	9	1	1681*	310	235	22	26	5	1	1681*	101	19	28	15	9	1
320-045	1686*	96	14	30	6	2	1	1686*	87	1	30	1	2	1	1686*	95	14	30	6	2	1	1686*	82	1	30	1	2	1

The node branching is much faster on some instances and has roughly the same runtime on other ones. On the larger instances, except one, `n-ho` often finds a significantly better solution. This is a clear indication of the benefits of node branching. Moreover, the results show that the search tree is smaller for `n-ho`, compared to `a-ho`. This may be explained by the fact that the new branching evenly strengthens LB on both branches which result in stronger pruning of the search. Also, `n-ho` requires fewer cuts to prove a test optimal. This may be explained by the fact that when an arc is included it makes several previously computed cuts useless.

5.2 Branch-and-Price Evaluation

There are several difficulties that appear during the integration of the Steiner tree algorithm in a branch-and-price framework. We briefly mention some of them.

The Steiner cuts that are generated are valid not only within the branch-and-cut tree, but also throughout the branch-and-price tree itself. It is important that the implementation exploits that by maintaining a "cut pool" containing all previously generated Steiner cuts. These can be re-used at separation.

It is important to exclude trees that correspond to columns that already exist in the branch-and-price master. It is easily done with the solver described in this paper since we can add a linear inequality to the STP for each existing, and therefore non-improving, tree.

We have included the branch-and-cut solver into the branch-and-price approach of [6]. The solver was modified to return the first beneficial Steiner tree to enable fast progress of the column generation. Extra effort in optimising the subproblem is not necessarily beneficial for reaching global convergence fast. Two sets of instances occurring in a multicast network design application were considered. The first one, comprising 28 instances, considers less than 5 multicast commodities and the second one, with 12 instances, has around 19 commodities.

Curiously, there is little difference between the two branching strategies. Only in one case did `n` enable the search to find a slightly better solution. The explanation for this is probably that the emerged Steiner subproblems needed no branching: the root LP was integral. This is consistent with what appears to be the case in [19]. It is worth mentioning that commercial Internet topologies, like the ones we considered, are typically sparse: the number of edges is below $4|V|$. This explains why the branching is not beneficial for Internet-like topologies.

Table 2. Aggregated results for branch-and-price method [6] with instances

	UB ratio			Time ratio			SP-t ratio			LP-t ratio			#cuts ratio			#bp-nodes		
$\|K\|$	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max
18–20	1.00	0.97	1.00	1.00	1.00	1.00	0.88	0.59	1.27	0.94	0.79	1.18	0.59	0.42	0.84	0.90	0.75	1.13
2–5	0.99	0.80	1.00	1.09	0.99	1.45	0.88	0.56	1.51	0.93	0.77	1.12	0.59	0.30	1.00	0.89	0.40	1.10

We now compare two variants `n-ff` and `n-ho` as subproblem solvers. Figure 2 presents an aggregated form of the results. The values are the ratio between `n-ho` and `n-ff`. For example, the average incumbent `n-ho` solution was 1% better than the one found by `n-ff` for small the instances. The column #bp-nodes describes the size of the branch-and-price tree.

Although it is marginally slower on average, the `ho` cutting strategy enables the branch-and-price algorithm to find slightly better solutions on average. The results also show that the separation is quicker and the number of cuts is drastically reduced. Again, this suggests that the most violated cuts are the important ones.

Interestingly, the `ho` cutting strategy enables the search to use fewer number of nodes. This may be because faster subproblem solving enables the encountering of primal solutions earlier and thus makes bound pruning possible.

6 Conclusion

Motivated by a multicast network design application, our work focuses on tailoring an existing Steiner tree approach so that it successfully serves as a fast subproblem solver in a branch-and-price framework. We adapt a little known preflow-push approach and demonstrate how to turn it into an effective separation algorithm. The search is also enhanced with a vertex oriented branching rule. We show that they both improve the performance in the decomposition context. We expect that our techniques will be more beneficial if the Steiner subproblems are *hard* to solve, unlike the reported multicast instances.

Even though our work originated in a decomposition framework, its benefits also unfold when solving stand-alone Steiner tree problems. In particular it succeeds in significantly improving the best known runtime for a test that was recently solved for the first time by [18].

There are many important applications which admit a natural formulation as a collection of cut-based covering inequalities similar to the Steiner tree problem. These include survivable networks e.g. [11]. Such applications often consist of finding several minimum $s-t$ cuts. It may be possible to adapt and exploit our cutting strategy in order to efficiently compute violated cuts.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms and applications*. Prentice-Hall, New Jersey, 1993.
2. B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
3. S. Chopra, E.R. Gorres, and M. R. Rao. Solving the Steiner tree problem on a graph using branch and cut. *Journal on Computing*, 4(3):320–335, 1992.
4. S. Chopra and M.R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–229, 1994.
5. W. Cronholm and F. Ajili. Strong cost-based filtering for Lagrange decomposition applied to network design. In M. Wallace, editor, *CP'04*, number 3258 in LNCS, pages 726–730, Toronto, Canada, September 2004.

6. W. Cronholm and F. Ajili. Hybrid branch-and-price for multicast network design. In *Proceedings of the 2nd International Network Optimization Conference*, March 2005. To appear.
7. W. Cronholm, W. Ouaja, and F. Ajili. Strengthening optimality reasoning for a network routing application. In *Proceedings of the Fourth Workshop on Cooperative Solvers in Constraint Programming, COSOLV'04*, Toronto, Canada, September 2004.
8. A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.
9. J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17:424–446, 1994.
10. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics.* Elsevier Science Publishers B. V., 1992.
11. H. Kerivin and A.R. Mahjoub. Separation of partition inequalities for the $(1, 2)$-survivable network design problem. *Operations Research Letters*, 30(4):265–268, August 2002.
12. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.
13. T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000.
14. M.S. Levine. Experimental study of minimum cut algorithms. Master's thesis, Massachusetts Institute of Technology, 1995.
15. A. Lucena and J. E. Beasley. A branch and cut algorithm for the Steiner problem in graphs. *Networks, an International Journal*, 31(1):39–59, 1998.
16. M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19–36, 1990.
17. S. Panagiotidi. Efficient implementation of a preflow push solver and its application to network design. Master's thesis, Imperial College London, September 2004.
18. T. Polzin. *Algorithms for the Steiner Problem in Networks.* PhD thesis, Faculty of Science and Technology, University of Saarlandes, May 2003.
19. M. Prytz and A. Forsgren. Dimensioning multicast-enabled communications networks. *Networks*, 39(4):216–231, 2002.