Alcom-FT Technical Report Series
ALCOMFT-TR-02-135

Extending Reduction Techniques for the Steiner Tree Problem *

Tobias Polzin

Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany,
polzin@mpi-sb.mpg.de

Siavash Vahdati Daneshmand

Theoretische Informatik, Universität Mannheim,
68131 Mannheim, Germany
vahdati@informatik.uni-mannheim.de

Abstract

Reduction methods are a key ingredient of the most successful algorithms for the Steiner problem. Whereas classical reduction tests just considered single vertices or edges, recent and more sophisticated tests extend the scope of inspection to more general patterns. In this paper, we present such an extended reduction test, which generalizes different tests in the literature. We use the new approach of combining alternative- and bound-based methods, which substantially improves the impact of the tests. We also present several algorithmic contributions. The experimental results show a large improvement over previous methods using the idea of extension, leading to a drastic speed-up in the optimal solution process and the solution of several previously unsolved benchmark instances.
Keywords: Steiner Problem, Reductions

1 Introduction

The Steiner problem in networks is the problem of connecting a subset of the vertices of a weighted network at minimum cost. This is a classical $\mathcal{NP}$-hard problem [5] with many important applications in network design in general and VLSI design in particular. For background information on this problem see [4].

A key ingredient of the most successful algorithms [1, 7, 10] for the Steiner problem are reduction techniques, i.e., methods to reduce the size of a given instance while preserving at least one optimal solution (or the ability to efficiently reconstruct one). While classical reduction tests just inspected simple patterns (vertices or edges), recent and more sophisticated tests extend the scope of inspection to more general patterns (like trees).

In this paper, we present such an extended reduction test, which generalizes different tests in the literature. We use the new approach of combining alternative- and bound-based

*This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

methods, which substantially improves the impact of the tests. We also present several algorithmic improvements, especially for the computation of the necessary information.

Since reduction techniques cannot be properly evaluated using worst-case analysis, we study the behaviour of the used techniques using the well-established benchmark library SteinLib [6]. The experimental results show a substantial improvement over all previous reduction methods (including our previous results in [10]). In particular, the effect on VLSI-instances is impressive. We achieve far more reductions than in any previous work in relatively short running time. As a consequence, we can now solve all VLSI-instances in SteinLib (including several previously unsolved instances) in tolerable time (few seconds in most cases).

The outline of this paper is as follows. After some preliminaries in the next two subsections, in Section 2 we first describe the test in a generic form. The generic algorithm is substantiated by presenting the applied test conditions (Section 2.1) and criteria for guiding and truncation of expansion (Section 2.2). A very critical issue for the success of such a test is the computation of the necessary information; this and other implementation issues are discussed in Section 2.3. Different design decisions lead to different variants of the test, as described in Section 2.4. Computational experiments on the impact of the tests are reported in Section 3. Finally, Section 4 contains some concluding remarks.

1.1 Definitions

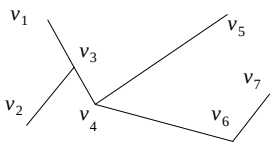
The Steiner problem in networks can be stated as follows (see [4] for details): Given an (undirected, connected) network $G = (V, E, c)$ with vertices $V = \{v_1, \ldots, v_n\}$, edges E and edge weights $c_{ij} > 0$ for all $(v_i, v_j) \in E$, and a set R, $\emptyset \neq R \subseteq V$, of **required vertices** (or **terminals**), find a minimum weight tree in G that spans R (called a **Steiner minimal tree**). If we want to stress that v_i is a terminal, we will write z_i instead of v_i.

We also look at a reformulation of this problem using the (bi-)directed version of the graph: Given $G = (V, E, c)$ and R, find a minimum weight arborescence in $\vec{G} = (V, A, c)$ ($A := \{[v_i, v_j], [v_j, v_i] \ : \ (v_i, v_j) \in E\}$, c defined accordingly) with a terminal (say z_1) as the root that spans $R_1 := R \setminus \{z_1\}$.

For any two vertices v_i and v_j, the **distance** $d_{ij} = d(v_i, v_j)$ between v_i and v_j is the length of a shortest path between v_i and v_j. A **Steiner bottleneck** of a path P_{ij} between v_i and v_j is a longest subpath with (only) endpoints in $R \cup \{v_i, v_j\}$; and the **Steiner bottleneck distance** s_{ij} between v_i and v_j is the minimum Steiner bottleneck length over all P_{ij}.

For every tree T in G, we denote by $L(T)$ the leaves of T, by $V(T)$ the vertices of T, and by $c(T)$ the sum of the costs of edges in T. Let T' be a subtree of T. The **linking set** between T and T' is the set of vertices $v_i \in V(T')$ with at least one fundamental path from v_i to a leaf of T not containing any edge of T'. If the linking set between T and T' is equal to $L(T')$, T' is said to be **peripherally contained** in T (Figure 1). A set $L' \subseteq V(T), |L'| > 1$, induces a subtree $T_{L'}$ of T containing for every two vertices $v_i, v_j \in L'$ the fundamental path between v_i and v_j in T. We define L' to be a **pruning set** if L' contains the linking set between T and $T_{L'}$.

A **key node** in a tree T is a node that is either a a nonterminal of degree at least 3 or a terminal. A **tree bottleneck** between $v_i \in T$ and $v_j \in T$ is a longest subpath on the path between v_i and v_j in T in which only the endpoints may be key nodes; and t_{ij} denotes the length of such a tree bottleneck.



For the depicted tree T, let T' be the subtree of T after removing the edges (v_4, v_6) and (v_6, v_7). The linking set between T and T' is $\{v_1, v_2, v_4, v_5\}$, and therefore T' is not peripherally contained in T. But if we add (v_4, v_6) to T', it is.

Figure 1: Depiction of some central notions

1.2 Preliminaries for Reduction Tests

We distinguish between two major classes of reduction tests, namely the alternative-based tests and the bound-based tests [10].

The **alternative-based** tests use the existence of alternative solutions. In case of exclusion tests, it is shown that for any solution containing a certain part of the graph (e.g. a vertex or an edge) there is an alternative solution of no greater cost without this part; the inclusion tests use the converse argument.

A basic alternative-based test is the so-called SD-test [3] (here called s-test): Any edge (v_i, v_j) with $s_{ij} < c_{ij}$ can be excluded. (This test can be extended to the case of equality, if a path corresponding to s_{ij} does not contain (v_i, v_j).)

The **bound-based** tests use a lower bound for the value of an optimal solution under the assumption that a certain part of the graph is contained (in case of exclusion tests) or is not contained (in case of inclusion tests) in the solution. These tests are successful if such a constrained lower bound exceeds a known upper bound, typically the value of a (not necessarily optimal) Steiner tree. But it is usually too costly to recompute a strong lower bound from scratch for each constraint. Here one can use the following simple, but very useful observation:[1]

Lemma 1 *Let $G = (V, A, c)$ be a (directed) network (with a given set of terminals) and $c'' \leq c$. Let $lower'$ be a lower bound for the value of any (directed) Steiner tree in $G' = (V, A, c')$ with $c' := c - c''$. For each incidence vector x representing a feasible Steiner tree for G, it holds: $lower' + c'' \cdot x \leq c \cdot x$.*

A typical application of this lemma is an approach based on Linear Programming: Any linear relaxation can provide a dual feasible solution of value $lower'$ and reduced costs c''. We can use a fast method to compute a constrained lower bound $lower''_{con}$ with respect to c''. Using Lemma 1, it easily follows that $lower_{con} := lower' + lower''_{con}$ is a lower bound for the value of any solution satisfying the constraint.

As an example for such a relaxation, consider the (directed) cut formulation P_C [16]: The Steiner problem is formulated as an integer program by introducing a binary x-variable for each arc (in case of undirected graphs, the bidirected counterpart is used, fixing a $z_1 \in R$ as the root). For each cut separating the root from another terminal, there is a constraint requiring that the sum of x-values of the cut arcs is at least 1. In this way, every feasible binary solution represents a feasible solution for the Steiner problem and each minimum solution (with value $v(P_C)$) a Steiner minimal tree. Now we can use a dual feasible solution of value $lower'$ for LP_C, the linear relaxation of P_C, to apply the method described above.

[1] All proofs can be found in the appendix.

For example, a lower bound for any Steiner tree with the additional constraint that it must contain a certain nonterminal v can be computed by adding $lower'$ to the length of a shortest path with respect to the reduced costs c'' from z_1 via v to another terminal, because any optimal Steiner tree including v must contain such a path.

If a test condition is successful, a test action is performed that modifies the current graph. For the tests in this paper, two such modifications are relevant: the elimination of an edge, which is straightforward, and replacing a nonterminal. The latter action is performed once it is established that this vertex has degree at most two in an optimal Steiner tree. The graph is modified by deleting the vertex and the incident edges and introducing a clique over the adjacent vertices in which the cost of each edge is the sum of the costs of the two edges it replaces. Note that in case of parallel edges, only one of minimum cost is kept.

2 Extending Reduction Tests

The classical reduction tests for the Steiner problem inspected just simple patterns (a single vertex or a single edge). There have been some approaches in the literature for extending the scope of inspection [2, 14, 15]. The following function *EXTENDED-TEST* describes in pseudocode a general framework for many of these approaches. The argument of *EXTENDED-TEST* is a tree T that is expanded recursively. For example, to eliminate an edge e, T is initialized with e. The function returns 1 if the test is successful, i.e., it is established that there is an optimal Steiner tree that does not peripherally contain T.

In the pseudocode, the function *RULE-OUT*(T, L) contains the specific test conditions (see Section 2.1): *RULE-OUT*(T, L) returns 1 if it is established that T is not contained with linking set L in at least one optimal Steiner tree.

```
EXTENDED-TEST(T) :
     (returns 1 only if T is not contained peripherally in an optimal Steiner tree)
1      if RULE-OUT(T, L(T)) :
2           return 1            (test successful)
3      if TRUNCATE(T) :
4           return 0            (test truncated)
5      forall leaves v_i of T :
6           if v_i ∉ R and PROMISING(v_i) :
7             success := 1
8             forall nonempty extension ⊆ {(v_i, v_j) :
                    not RULE-OUT(T ∪ {(v_i, v_j)}, L(T) ∪ {v_j})} :
9                 if not EXTENDED-TEST(T ∪ extension) :
10                  success := 0
11            if success :
12                return 1              (no acceptable extension at v_i)
13     return 0          (in all inspected cases, there was an acceptable extension)
```

The correctness can be proven easily by induction using the fact that if T' is a subtree of an optimal Steiner tree T^* and contains no inner terminals, all leaves of T' are connected to some terminal by paths in $T^* \setminus T'$.

Clearly the decisive factor in this algorithm is the realization of the functions *RULE-OUT*, *TRUNCATE* and *PROMISING*.

Using this framework, previous extension approaches can be outlined easily:

- In [15] the idea of expansion was introduced for the rectilinear Steiner problem.
- In [14] this idea was adopted to the Steiner problem in networks. This variant of the test tries to replace vertices with degree three; if this is successful, the newly introduced edges are tested again with an expansion test. The expansion is performed only if there is a single possible extension at a vertex, thus eliminating the need for backtracking.
- In [2] backtracking was explicitly introduced, together with a number of new test conditions to rule out subnetworks, dominating those mentioned in [14].
- In [10], we introduced a different test that tries to eliminate edges. Expansion is performed only if there is at most one possible extension (thus inspecting a path) and only if the elimination of one edge implies the elimination of all edges of the path.

All previous approaches use only alternative-based methods. We present an expansion test that explicitly combines the alternative-based and bound-based methods. This combination is far more effective than previous tests, because the two approaches have complementary strengths. Intuitively speaking, the alternative-based method is especially effective if there are terminals in the vicinity, because it uses the Steiner bottleneck distances. On the other hand, the bound-based method is especially effective if there are no close terminals, because it uses the distances (with respect to reduced costs) to terminals. Furthermore, for the expansion test to be successful, usually many possible extensions must be considered and it is often the case that not all of them can be ruled out using exclusively the alternative- or the bound-based methods, whereas an explicit combination of both methods can do the job.

Although the pseudocode of *EXTENDED-TEST* is simple, designing an efficient and effective implementation requires many algorithmic ideas and has to be done carefully, taking the interaction between different actions into account, which is highly nontrivial. Since writing down many pages of pseudocode would be less instructive, we prefer to explain the main building blocks. In the following, we first describe the test conditions for ruling out trees (the function *RULE-OUT*), using the results of [2] and introducing new ideas. Then we explain the criteria used for truncation and choice of the leaves for expansion (the functions *TRUNCATE* and *PROMISING*). Finally, we will address some implementation issues, particularly data structures for querying different types of distances.

2.1 Test Conditions

For the following test conditions we always consider a tree T with $V(T) \cap R \subseteq L(T)$, i.e., terminals may appear only as leaves of the tree. A very general formulation of the alternative-based test condition is the following:

Lemma 2 *Consider a pruning set L' for T. If $c(T_{L'})$ is larger than the cost of a Steiner tree T' in $G' = (V, \binom{V}{2}, s)$ with L' as terminals, then there is an optimal Steiner tree that does not peripherally contain T. This test can be strengthened to the case of equality if there is a vertex v in $T_{L'}$ that is not in any of the paths used for defining the s-values of the edges of T'.*

A typical choice for L' is $L(T)$, often with some leaves replaced by vertices added to T in the first steps of the expansion. If computing an optimal Steiner tree T' is considered too

expensive, the cost of a minimum spanning tree for L' with respect to s can be used as a valid upper bound.

A relaxed test condition compares Steiner bottlenecks with tree bottlenecks:

Lemma 3 *If $s_{ij} < t_{ij}$ for any $v_i, v_j \in T$, there is an optimal Steiner tree that does not peripherally contain T. Again, the test can be strengthened to the case of equality if a path corresponding to s_{ij} does not contain a tree bottleneck of T between v_i and v_j.*

The bound-based test condition uses a dual feasible solution for LP_C of value $lower'$ and corresponding reduced costs c'' (with resulting distances d''):

Lemma 4 *Let $\{l_1, l_2, \ldots, l_k\} = L(T)$ be the leaves of T. Then*

$$lower_{con} := lower' + min_i\{d''(z_1, l_i) + c''(\vec{T}_i) + \sum_{j \neq i} min_{z_p \in R_1} d''(l_j, z_p)\}$$

defines a lower bound for the cost of any Steiner tree under the assumption that it peripherally contains T, where $\vec{T}_i$ denotes the directed version of T when rooted at l_i.

In the context of replacement of edges, one can use the following lemma.

Lemma 5 *Let e_1 and e_2 be two edges of T in a reduced network. If both edges originate from a common edge e_3 by a series of replacements, then no optimal Steiner tree for the reduced network that corresponds to an optimal Steiner tree in the unreduced network contains T.*

The conditions above cover the calls *RULE-OUT*(T, L) with $L = L(T)$. In case other vertices than the leaves need to be considered in the linking set (as in line 8 of the pseudocode), one can easily establish that all lemmas above remain valid if we treat all vertices of L as leaves.

2.2 Criteria for Expansion and Truncation

The basic truncation criterion is the number of backtracking steps, where there is an obvious tradeoff between the running time and the effectiveness of the test. A typical number of backtracking steps in our implementations is five.

Additionally, there are other criteria that guide and limit the expansion:

1. If a leaf is a terminal, we cannot easily expand over this leaf, because we cannot assume anymore that an optimal Steiner tree must connect this leaf to a terminal by edges not in the current tree. However, if all leaves are terminals (a situation in which no expansion is possible for the original test), we know that at least one leaf is connected by an edge-disjoint path to another terminal (as long as not all terminals are spanned by the current tree). This can be built into the test by another level of backtracking and some modifications of the test conditions. But we do not describe the modifications in detail, because the additional cost did not pay off in terms of significantly more reductions.

2. If the degree *deg* of a leaf is large, considering all $2^{(deg-1)} - 1$ possible extensions would be too costly and the desired outcome, namely that we can rule out all of these extended subtrees, is less likely. Therefore, we limit the degree of possible candidates for expansion by a small constant, e.g. 8.

3. It has turned out that a depth-first realization of backtracking is quite successful. In each step, we consider only those leaves for expansion that have maximum depth in T when rooted at the starting point. In this way, the bookkeeping of the inspected subtrees becomes much easier and the whole procedure can be implemented without recursion. A similar idea was already mentioned (but not explicitly used) in [2].

4. In case we do not choose the depth-first strategy, a tree T could be inspected more than once. As an example, consider a tree T resulting from an expansion of T' at leaf v_i and then at v_j. If T cannot be ruled out, it is possible that we return to T', expand it at v_j and then at v_i, arriving at T again. This problem can be avoided by using a (hashing-based) dictionary.

2.3 Implementation Issues

Precomputing (Steiner) Distances: A crucial issue for the implementation of the test is the calculation of Steiner bottleneck distances. An exact calculation of all s_{ij} needs time $O(|V|(|E|+|V|\log|V|))$ [2] and space $\Theta(|V|^2)$, which would make the test impractical even for mid-size instances. So we need a good approximation of these distances and some appropriate data structures for retrieving them. Building upon a result of Mehlhorn [8], Duin [1] gave a nice suggestion for the approximation of the Steiner bottleneck distances, which needs preprocessing time $O(|E|+|V|\log|V|)$ (mainly three shortest paths calculations to determine the three nearest terminals for each nonterminal and a minimum spanning tree calculation) and a small running time for each query (the query time can even be constant, if all necessary queries are known in advance [10]). Although the resulting approximate values $\hat{s}_{ij}$ produce quite satisfactory results for the original s-test of Section 1.2, for the extended test the results are unfortunately much worse than with the exact values. But we observed that $\tilde{s} = \min\{\hat{s}, d\}$ is almost always equal to the exact s-values, and therefore can be used in the extended test as well. Still there remains the problem of computing the d-values: For each vertex v_i we compute and store in a neighbor list the distances to a constant number (e.g. 50) of nearest vertices. But we consider only vertices v_j with $d_{ij} < \hat{s}_{ij}$. This is justified by the observation that in case $d_{ij} \geq \hat{s}_{ij}$, for all descendants v_k of v_j in the shortest paths tree with the root v_i there is a path P_{ik} of Steiner bottleneck length s_{ik} containing at least one terminal, and in such cases s_{ik} is usually quite well approximated by $\hat{s}_{ik}$.

Now, we use different methods for different variants of the test:

1. If we only replace vertices in an expansion test, the d- and s-values do not increase and we can use the precomputed neighbor lists during the whole test using binary search.

2. If we limit the number of (backtracking) steps, then we can confine the set of all possible queries in advance. When a vertex is considered for replacement by the expansion test, we first compute the set of possibly visited neighbors (adjacent vertices, and vertices adjacent to them, and so on, up to the limited depth). Then we compute a distance matrix for this set according to the $\tilde{s}$-values. Using this matrix, each query can be answered in constant time.

3. If we also want to delete edges, we have to store for each vertex in which neighbor list computations it was used. When an edge is deleted, we can redo the $\hat{s}$ computation (or at least those parts that may have changed due to the edge deletion) and restore the

affected neighbor lists. This can even be improved by a lazy calculation of the neighbor lists.

Tree Bottlenecks: The tree bottleneck test of Lemma 3 can be very helpful, because every distance between tree vertices calculated for a minimum spanning tree or a Steiner minimal tree computation can be tested against the tree bottleneck; and in many of the cases where a tree can be ruled out, already an intermediate bottleneck test can rule out this tree, leading to a shortcut in the computation. This is especially the case if there are long chains of nodes with degree two in the tree. We promote the building of such chains while choosing a leaf for extension: We first check whether there is a leaf at which the tree can be expanded by only one edge. In this case we immediately perform this expansion, without creating a new key node and without the need of backtracking through all possible combinations of expansion edges.

The tree bottleneck test can be sped up by storing for each node of the tree the length of a tree bottleneck on the path to the starting vertex. For each two nodes v_i and v_j in the tree, the maximum of these values gives an upper bound for the actual tree bottleneck length t_{ij}. Only if this upper bound is greater than the (approximated) Steiner bottleneck distance, an exact tree bottleneck computation is performed.

Computations for the Bound-Based Tests: An efficient method for generating the dual feasible solution needed for the bound-based test of Lemma 4 is the DUAL-ASCENT algorithm described in [10]. We improve the test by calculating a lower bound and reduced costs for different roots. Although the optimal value of the dicut relaxation does not change with the choice of the root, this is not true concerning the value of the dual feasible solution generated by DUAL-ASCENT and, more importantly, the resulting reduced costs can have significantly different patterns, leading to a greater potential for reductions. Even more reductions can be achieved by using stronger lower bounds, as computed with a row generating algorithm [10]. Concerning the tests for the replacement of vertices, we use only the result of the final iteration, which provides an optimal dual solution of the underlying linear relaxation. The dual feasible solutions of the intermediate iterations are used only for the tests dealing with the deletion of edges, because the positive effect of the replacement of a vertex (Section 1.2) cannot be translated easily into linear programs.

Replacement History: Our program package can transform a tree in a reduced network back into a tree in the original instance. For this purpose, we assign a unique ID number to each edge. When a vertex is replaced, we store for each newly inserted edge a triple with the new ID and the two old IDs of the replaced edges. We use this information to implement the test described in Lemma 5. First we do some preprocessing, determining for each ID the edges it possibly originate from (here called ancestors); this can be done in time and space linear in the number of IDs. Later, a test for a conflict between two edges (i.e., they originate from the same edge) can be performed by marking the IDs of the ancestors of one edge and then checking the IDs of ancestors of the other edge; so each such test can be done in time linear in the number of ancestors. We perform this test each time the current tree T is to be extended over a leaf v_i (with (v_k, v_i) in T) by an edge (v_i, v_j). Then we check for a conflict between (v_i, v_j) and (v_k, v_i). This procedure implements an idea briefly mentioned in [2], where a coloring scheme was suggested for a similar purpose. Our scheme has the advantage

that it may even discover conflicts in situations where an edge is the result of a series of replacements.

2.4 Variants of the Test

A general principle for the application of reduction tests is to perform the faster tests first so that the stronger (and more expensive) tests are applied to (hopefully) sufficiently reduced graphs. In the present context, different design decisions (e.g. trying to delete edges or replace vertices) lead to different consequences for an appropriate implementation and quite different versions of the test, some faster and some stronger.

We have implemented four versions of expansion tests and integrated them into the reduction process described in [10]. Some details of the corresponding implementations were already given in Section 2.3.

1. For a fast preprocessing we use the linear time expansion test that eliminates paths, as described in [10].
2. A stronger variant tries to replace vertices, but only expands at leaves that are most edges away from the starting vertex.
3. Even stronger but more time-consuming is a version that performs full backtracking.
4. The most time-consuming variant tries to eliminate edges.

3 Computational Results

In this paper, we concentrate on the VLSI-Instances of the benchmark library SteinLib [6]. In a different context, we report in [11] results on the so-called geometric instances of SteinLib using (among other techniques) the methods described in this paper. Other groups of instances could be either solved fairly fast already without these new techniques (as we reported in [10]), or (as for some groups of instances added meanwhile to SteinLib) are deliberately constructed to be hard for the known methods, with the consequence that the impact of the described methods on them is not decisive.

We report results both for the extended reduction techniques described in this paper and for the optimal solution using these techniques. All these results are, to our knowledge, by far the best results achieved for the corresponding instances. In particular, we could solve all VLSI-instances of SteinLib, including several previously unsolved instances, thereby substantially improving the solution times for the previously solved instances (up to some orders of magnitude for the larger instances).

Each of the test series was performed with the same parameters for all instances of all groups. Although in some cases individual parameter tuning could lead to some improvements, the used methods turn out to be fairly robust, so not much is lost by using the same parameters for all instances. All tests were performed on a PC with an AMD Athlon 1.1 GHz processor and 768 MB of main memory, using the operating system Linux 2.2.13. We used the GNU egcs 1.1.2 compiler and CPLEX 7.0 as LP-solver.

In Table 1, we report reduction results for different groups of VLSI-instances of SteinLib. The last group (LIN) was not present in the older versions of SteinLib and included several unsolved instances that we could solve with the help of the techniques described in this paper. For each group, three kinds of results are given:

1. using classical (not extended) reduction tests, as already described in [10];
2. using fast variants of the extended tests (variants 1 and 2 in Section 2.4 in addition to 1 above);
3. using strong variants of the extended tests (variants 3 and 4 in Section 2.4 in addition to 1 and 2 above).

For each group of instances, we give the average values for running time (in seconds) and the fraction of edges remaining after the corresponding reductions. Detailed results for single instances can be found in [9]. For the sake of comparison, we also provide the best other results [13, 14] we are aware of. Those other results were produced on a Sun ULTRA 167 MHz, but even if one divides the corresponding running times by 10, our strong variants are still much faster, while already our fast variants achieve better reduction results in all cases.

instance group	classical reductions		extended red., fast variants		extended red., strong variants		Uchoa et. al. [14]	
	time	% of $\|E\|$ remaining	time	% of $\|E\|$ remaining	time	% of $\|E\|$ remaining	time	% of $\|E\|$ remaining
ALUE	2.89	34.50	10.06	0.63	10.36	0.62	1310.93	14.25
ALUT	4.02	38.27	32.62	0.75	33.81	0.49	1806.44	11.76
DIW	0.35	11.57	0.88	0.00	0.88	0.00	214.67	4.10
DMXA	0.06	8.59	0.07	0.00	0.07	0.00	4.14	2.55
GAP	0.13	3.21	0.31	0.00	0.31	0.00	60.54	2.62
MSM	0.13	7.76	0.24	0.05	0.24	0.05	12.90	2.66
TAQ	0.39	23.83	0.98	0.18	0.99	0.16	69.29	11.33
LIN	3.52	27.04	106.17	5.16	1329.21	1.58	—	—

Table 1: Comparison of Reduction Results (Averages)

In Table 2, we report the average times (in seconds) for the optimal solution of different groups of instances. The solution method is the same as in [10]; here we have additionally used the extended reduction methods described in this paper (all variants) together with some new methods we described in [12]. Again, for comparison we have included the best other results we are aware of [13], where a branch-and-cut algorithm was used after the reduction phase.

4 Concluding Remarks

In this paper, we described a generic algorithm and some concrete variants of it for extending the scope of reduction tests. The new approach of explicitly combining alternative- and bound-based methods, together with many algorithmic ideas, lead to substantial improvement over previous tests that used the idea of expansion.

A very important issue in the context of reduction tests is the interaction between different methods. This is especially important for the tests described in this paper, because they tend to transform instances not only in their size, but also in their type. In particular, the success of the described tests for replacing nonterminals, in cooperation with the edge-elimination tests, tend to transform graphs of high connectivity to graphs with many small vertex separators,

instance group	our results	Uchoa [13]
ALUE	13	12560
ALUT	35	275817
DIW	0.89	243
DMXA	0.07	4.2
GAP	0.31	61
MSM	0.24	15
TAQ	0.99	163
LIN	20032	—

Table 2: Comparison of Times for Exact Solution (Averages)

often consisting of terminals alone. This prepares the ground for another group of reduction tests, which we describe in [12].

References

[1] C. W. Duin. *Steiner's Problem in Graphs.* PhD thesis, Amsterdam University, 1993.

[2] C. W. Duin. Preprocessing the Steiner problem in graphs. In D-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*, pages 173–233. Kluwer Academic Publishers, 2000.

[3] C. W. Duin and T. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989.

[4] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics.* North-Holland, Amsterdam, 1992.

[5] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[6] T. Koch and A. Martin. SteinLib. `http://elib.zib.de/steinlib`, 1997.

[7] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[8] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.

[9] T. Polzin and S. Vahdati Daneshmand. Extending reduction techniques for the Steiner tree problem: A combination of alternative- and bound-based approaches. Research Report MPI-I-2001-1-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, 2001.

[10] T. Polzin and S. Vahdati Daneshmand. Improved algorithms for the Steiner problem in networks. *Discrete Applied Mathematics*, 112:263–300, 2001.

[11] T. Polzin and S. Vahdati Daneshmand. On Steiner trees and minimum spanning trees in hypergraphs. Research Report MPI-I-2001-1-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, 2001.

[12] T. Polzin and S. Vahdati Daneshmand. Partitioning techniques for the Steiner problem. Research Report MPI-I-2001-1-006, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, 2001.

[13] E. Uchoa. *Algoritmos Para Problemas de Steiner com Aplicações em Projeto de Circuitos VLSI (in Portuguese).* PhD thesis, Departamento De Informática, PUC-Rio, Rio de Janeiro, April 2001.

[14] E. Uchoa, M. Poggi de Arago, and C. C. Ribeiro. Preprocessing Steiner problems from VLSI layout,. Technical Report MCC. 32/99, Departamento de Informtica, PUC-Rio, Rio de Janeiro, Brasil, 1999.

[15] P. Winter. Reductions for the rectilinear Steiner tree problem. *Networks*, 26:187–198, 1995.

[16] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

5 Appendix

Proof of Lemma 1: $c \cdot x = c' \cdot x + c'' \cdot x \geq lower' + c'' \cdot x$. □

Proof of Lemma 2: [2] Assume that T is peripherally contained in an (optimal) Steiner tree T^* in G. As L' is a pruning set for T and the leaves of T are a pruning set for T^*, L' is also a pruning set for T^*. It follows that after removing the edges of $T_{L'}$ from T^*, each of the remaining subtrees contains one vertex of L'. The plan is to reconnect these subtrees to a new Steiner tree by replacing each necessary edge of T' with a path in G of no larger cost. Consider the forest F consisting of these subtrees together with the remaining nodes of T' (i.e., nodes that are not in any of these subtrees). Merge all vertices of T' that are in one component of F, breaking emerging cycles by deleting an arbitrary edge of each cycle. This operation does not increase the cost of T'. Now, each component C_i of F corresponds to one vertex t_i of T'. We will ensure this invariant during the whole process of updating T' and F.

Choose a shortest edge (t_i, t_j) of T'. Let P_{ij} be a path of Steiner bottleneck length s_{ij} between v_i and v_j, vertices of V corresponding to t_i and t_j (before merging). Let P_{kl} be a subpath of P_{ij} in which only the endpoints v_k and v_l are in $R \cup \{v_i, v_j\}$, and v_k and v_l are in different components C_k and C_l of F. Remove an arbitrary edge on the fundamental path in T' between t_k and t_l and merge t_k and t_l in T'. Finally, connect C_k and C_l in F by adding the necessary edges from P_{kl}. The sum of the costs of these edges is not larger than s_{ij}. Because (t_i, t_j) was a shortest edge of T', the added cost in F is also not larger than the cost of the edge that was removed from T'.

Repeating this procedure leads to a new network that connects all terminals of G and has cost at most $c(T^*) - c(T_{L'}) + c(T')$. If $c(T') < c(T_{L'})$ or $c(T') = c(T_{L'})$ and there is at least one vertex in $V(T_{L'})$ that is not in the new tree (because it was not in any of the paths that were used for defining the s-values of the edges in T'), we have a Steiner tree of cost not larger then $c(T^*)$ that does not peripherally contain T. □

Proof of Lemma 3: Consider v_i, v_j and all key nodes on the fundamental path P_{ij} between v_i and v_j in T as the pruning set L' in the previous lemma. The induced subtree $T_{L'}$ is the path P_{ij} itself. Removing a tree bottleneck from P_{ij}, inserting an edge (v_i, v_j) of cost s_{ij} and substituting the c-values for the other edges with the (not larger) s-values leads to a Steiner tree for L' in G' with cost at most $c(P_{ij}) + s_{ij} - t_{ij}$. □

Proof of Lemma 4: If T is peripherally contained in an optimal Steiner tree T^*, then there is a path in T^* from the root terminal z_1 to a leaf l_i of T. After rooting T from l_i, each (possibly single-vertex) subtree of T^* corresponding to other leaves l_j contains a terminal. Now the lemma follows directly using Lemma 1 as described in Section 1.2. □

Proof of Lemma 5: Assume that there is an optimal Steiner tree T^* for the reduced network containing both e_1 and e_2. Back-substituting the edges of T^* leads to a solution in the original network in which e_3 is used twice. This means that the solution value in the unreduced and consequently in the reduced network can be decreased by $c(e_3)$, which contradicts the optimality of T^*. □

[2] There are proofs in [2, 4] for similar (but weaker) conditions, but they are not complete.