INFORMATIK

Extending Reduction Techniques for the Steiner Tree Problem: A Combination of Alternative- and Bound-Based Approaches

Tobias Polzin   Siavash Vahdati

MPI–I–2001–1–007   December 2001

FORSCHUNGSBERICHT   RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Stuhlsatzenhausweg 85   66123 Saarbrücken   Germany

Authors' Addresses

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

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

Abstract

A key ingredient of the most successful algorithms for the Steiner problem are reduction methods, 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 methods, which substantially improves the impact of the tests. We also present several algorithmic improvements, especially for the computation of the needed information. The experimental results show a substantial improvement over previous methods using the idea of extension.

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. Background information on this problem can be found in [4].

A key ingredient of the most successful algorithms [1, 6, 10] for the Steiner problem are reduction methods, 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 methods, which substantially improves the impact of the tests. We also present several algorithmic improvements, especially for the computation of the needed information. The experimental results show a substantial improvement over previous methods using the idea of extension.

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 needed 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. Some 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, because it yields stronger relaxations: Given $G = (V, E, c)$ and R, find a minimum weight arborescence in $\vec{G} = (V, A, c)$ ($A := \{[v_i, v_j], [v_j, v_i] \mid (v_i, v_j) \in E\}$, 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. 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 is a node which is either a terminal or a nonterminal of degree at least 3. A **key path** in a tree is a path in which (only) the endpoints are key nodes. A **tree bottleneck** between $v_i \in T$ and $v_j \in T$ is a longest key path on the fundamental path between v_i and v_j in T; and t_{ij} denotes the length of such a tree bottleneck.

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. For example 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 co-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 quite generally applicable observation:

Observation 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: $lower' + c'' \cdot x \leq c \cdot x$.

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

A typical application of this observation 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 Observation 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 [17]: 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. 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, which modifies the current graph. For the tests in this paper, two such modifications are relevant: the elimination of an edge (or some edges), 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, 15, 16]. 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 which does not peripherally contain T; otherwise it returns 0.

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 at least one optimal Steiner tree)
  if *RULE-OUT*$(T, L(T))$:
    return 1 (test successful)
  if *TRUNCATE*(T) :
    return 0 (test truncated)
  forall *leaves* v_i *of* T :

if $v_i \notin R$ **and** *PROMISING*(v_i) :
    success $:= 1$
    forall *nonempty extension* $\subseteq \{(v_i, v_j) \mid$ **not** *RULE-OUT*$(T \cup \{(v_i, v_j)\}, L(T) \cup \{v_j\})\}$:
        if not *EXTENDED-TEST*$(T \cup$ *extension*$)$:
            success $:= 0$
    if *success* :
        return 1 (no acceptable extension at v_i)
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 $T^* \setminus T'$.

Obviously 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 [16] the idea of expansion was introduced for the rectilinear Steiner problem.
- In [15] 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 [15].
- In [10], we introduced a different test which 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 complete version of it requires many algorithmic ideas and a careful implementation of them, 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 used criteria for truncation and choice of the leaves for expansion (the functions *TRUNCATE* and *PROMISING*). Finally, we will address some implementation issues, especially by presenting 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:

Observation 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 $T_{L'}$ which is not in any of the paths used for defining the s-values of the edges of T'.

Proof:[1] 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 which 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

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

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 $c(T^*)$ which does not peripherally contain T. $\square$

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.
Another test condition compares Steiner bottlenecks with tree bottlenecks. The proof shows that it is a relaxation of the previous condition.

Observation 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 between v_i and v_j.

Proof: 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 observation. 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 $c(P_{ij}) + s_{ij} - t_{ij}$. $\square$

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''):

Observation 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.

Proof: 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 observation follows directly using Observation 1 as described in Section 1.2. $\square$

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

Observation 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, than no optimal Steiner tree for the reduced network that corresponds to an optimal Steiner tree in the unreduced network contains T.

Proof: 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^*. $\square$

The conditions above cover the calls $RULE\text{-}OUT(T, L)$ 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 observations 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. Of course, 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. Thus 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 saving them partly. Duin [1], building upon a result of Mehlhorn [8], 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 needed queries are known in advance [10]). Unfortunately, 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 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 needed d-values. Here we use three different approaches that reflect the properties of different variants of the test.

1. A simple approach uses the precomputed $\hat{s}$-values and an on-demand distance calculation. This approach is used if we delete edges by an expansion test, because in this case the Steiner bottleneck distances can increase, so a test that uses the old $\tilde{s}$ values may produce wrong results. Therefore, we carefully determine the region where the shortest paths to the three nearest terminals could be destroyed and recompute them. If a deleted edge is contained in the minimum spanning tree, this tree has to be recomputed as well.

2. If we just perform replacement operations (and possibly edge deletions for newly inserted edges by the s-test), then the Steiner bottleneck distances do not increase. Therefore, we can use a caching technique: For a small number of vertices we keep a data structure for implementing an interruptible computation of distances by a variant of Dijkstra's algorithm. Upon a query for a $\tilde{s}_{ij}$, it is checked whether v_i or v_j is in this cache. If this is the case and $\tilde{s}_{ij}$ has already been computed, we are done; otherwise we resume the corresponding distance computation until the other vertex is reached, then the computation is interrupted again, and the values $\hat{s}_{ij}$ and $\tilde{s}_{ij}$ are computed and stored. If none of the vertices is in the cache, one cache slot is overwritten with the data for a new distance computation from either v_i or v_j.
For an extension of the test to the case of equality, observe that if neither v_i nor v_j is a terminal, than the edge (v_i, v_j) can even be deleted if the old $\hat{s}_{ij}$ is equal to c_{ij}, and the Steiner bottleneck distances still do not increase. Note that the value of the approximated Steiner bottleneck distances may change due to edge deletions even if the exact Steiner bottleneck distances stay the same.

3. If we limit the number of (backtracking) steps, then we can limit in advance the set of all possible queries. In this case, we use the following strategy: First we compute and store for each vertex the distances to a constant number (e.g. 50) of nearest vertices. When a vertex is considered for replacement by the expansion test, we first compute the set N of possibly visited neighbors (adjacent vertices, and vertices adjacent to them, and so on, up to the limited depth). Then we initialize a distance matrix for this set with the $\hat{s}$-values and use the precomputed distances for vertices in N to decrease these values if possible. Using this matrix, each query can be answered in constant time.

Tree Bottlenecks The tree bottleneck test of Observation 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 only by 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 Observation 4 is the DUAL-ASCENT algorithm described in [10] (a more detailed description of a fast implementation of DUAL-ASCENT is described in [9]). We improve the strength of 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, for each newly inserted edge we store 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 Observation 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 generalizes 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. This is especially expensive, because the mentioned approaches for the precomputation of the (approximate) Steiner bottleneck distances are not applicable.

3 Computational Results

In [10], we reported empirical times for the exact solution of the instances of the benchmark library SteinLib [7], which were by far the best results at the time of their submission (April 1998). In that work, all instances of SteinLib (the version of that time) could be solved fairly fast (at most about one minute) except some large VLSI-instances, which took longer (up to a couple of hours) or could not even be solved in days (four instances). In that work, we predicted that in the short term, major improvements, particularly for those VLSI-instances, are to be expected from further reduction techniques. This turned out to be the case: In the meantime, other researchers [15] and we could achieve major improvements by using extended reduction techniques.
In this paper, we concentrate on the VLSI-Instances of SteinLib (including some new instances added in the meantime). This makes it possible to compare both: Our methods with other approaches using extended reduction tests and different variants of our methods. 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 this techniques. All these results are, to our knowledge, by far the best results achieved for the corresponding instances. In particular, we could solve all original VLSI-instances of SteinLib in at most a couple of minutes, with the average times even much smaller. Also, we could solve several instances of the new version of SteinLib for the first time.
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 Tables 2-9, we report reduction results for different groups of VLSI-instances of SteinLib. For each instance, 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 of these test series, we report the size ($|V|$, $|E|$ and $|R|$) of the reduced instance; the fraction (in percent) of the original edges remained and the time (in seconds) for the corresponding reductions. A stroke means that the instance has been solved with the corresponding reductions alone.

For an easier comparison, we provide in Table 1 an overview of these results together with the best other results [15] we are aware of, giving for each group of instances the average values for running time (in seconds) and the fraction of edges remained after the corresponding reductions. 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.

In Tables 10-17, we report the times (in seconds) for the optimal solution of different groups of instances. The solution method is the same as in [10]; here we have used additionally the extended reduction methods described in this paper (all variants) together with some new methods we described in [12, 13]. Note that this means a different combination of methods with the side-effect that for some instances, the time for the exact solution is smaller than the reduction times reported in Tables 2-9.

instance group	classical reduction		extended reduction, fast variants		extended reduction, strong variants		Uchoa et. al.	
	time	% of $\|E\|$ remained	time	% of $\|E\|$ remained	time	% of $\|E\|$ remained	time	% of $\|E\|$ remained
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

Table 1: Comparison of Reduction Results (averages)

For the results reported in Table 17, a few additional comments are appropriate. This group of instances has been added to SteinLib recently and includes several instances not previously solved (lin31-37). We have solved all instances of this group; but a couple of them (all previously unsolved) needed relatively long computation times (up to a couple of days). For those instances, the weak point of our current approach seems to be the computation of sufficiently good lower bounds; with a solution (or a good approximation of a solution) of a strong relaxation (such as LP_C) at hand, the described reduction methods lead to a relatively fast exact solution of the instance.

The best other optimal solution results we are aware of for the considered instances were reported in [14], where a branch-and-cut algorithm was used after the reduction phase. But because in [14] only results for the more time-consuming instances are reported, we cannot perform a comparison of average times. For the instances considered there, the running times in [14] are typically 100 times or more larger than ours, with the difference growing with the size of the instances (again, the running times in [14] can be divided by at most 10 to count for their slower machine). For example, the average times for solving the largest four instances (alue7065, alue7080, alut2610, alut2625) are 665853 seconds in [14] and 122 seconds in this work.

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 which used the idea of expansion.

A very important issue in the context of reduction tests is the interaction of 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, 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. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[7] T. Koch and A. Martin. SteinLib. `http://elib.zib.de/steinlib/steinlib.php`, 2001.

[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. Implementing a fast dual-ascent for the Steiner problem. MPI Saarbrücken, Universität Mannheim, 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] T. Polzin and S. Vahdati Daneshmand. Using tighter relaxations for the Steiner problem. MPI Saarbrücken, Universität Mannheim, 2001.

[14] 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.

[15] 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.

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

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

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
taq0014	6466	11046	128	1.61	64.83	4123	7161	115	2.42	0.00		—		2.37	0.00		—	
taq0023	572	963	11	0.06	10.70	68	103	6	0.06	0.00		—		0.05	0.00		—	
taq0365	4186	7074	22	0.31	21.44	918	1517	21	0.69	0.00		—		0.70	0.00		—	
taq0377	6836	11715	136	1.69	76.85	5192	9003	134	5.30	2.54	187	297	66	5.44	2.26	170	265	62
taq0431	1128	1905	13	0.06	0.00		—		0.14	0.00		—		0.14	0.00		—	
taq0631	609	932	10	0.03	8.69	55	81	7	0.02	0.00		—		0.02	0.00		—	
taq0739	837	1438	16	0.08	26.29	233	378	12	0.12	0.00		—		0.12	0.00		—	
taq0741	712	1217	16	0.10	25.80	197	314	14	0.12	0.00		—		0.10	0.00		—	
taq0751	1051	1791	16	0.07	22.78	252	408	16	0.13	0.00		—		0.11	0.00		—	
taq0891	331	560	10	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
taq0903	6163	10490	130	1.40	76.20	4581	7993	109	4.71	0.00		—		4.72	0.00		—	
taq0910	310	514	17	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
taq0920	122	194	17	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
taq0978	777	1239	10	0.02	0.00		—		0.01	0.00		—		0.02	0.00		—	
Average:				0.39	23.83				0.98	0.18				0.99	0.16			

Table 2: Reduction Results for the TAQ-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
diw0234	5349	10086	25	0.33	8.67	496	874	23	1.17	0.00		—		1.12	0.00		—	
diw0250	353	608	11	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0260	539	985	12	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0313	468	822	14	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0393	212	381	11	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0445	1804	3311	33	0.16	17.70	346	586	26	0.09	0.00		—		0.09	0.00		—	
diw0459	3636	6789	25	0.21	4.67	184	317	14	0.12	0.00		—		0.13	0.00		—	
diw0460	339	579	13	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0473	2213	4135	25	0.08	0.00		—		0.08	0.00		—		0.08	0.00		—	
diw0487	2414	4386	25	0.11	0.00		—		0.12	0.00		—		0.12	0.00		—	
diw0495	938	1655	10	0.02	0.00		—		0.04	0.00		—		0.04	0.00		—	
diw0513	918	1684	10	0.03	0.00		—		0.03	0.00		—		0.03	0.00		—	
diw0523	1080	2015	10	0.02	0.00		—		0.03	0.00		—		0.02	0.00		—	
diw0540	286	465	10	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
diw0559	3738	7013	18	0.41	29.59	1130	2075	18	0.34	0.00		—		0.34	0.00		—	
diw0778	7231	13727	24	0.82	26.98	1979	3704	21	0.80	0.00		—		0.81	0.00		—	
diw0779	11821	22516	50	1.90	78.80	9294	17743	50	6.36	0.00		—		6.33	0.00		—	
diw0795	3221	5938	10	0.36	0.00		—		0.84	0.00		—		0.85	0.00		—	
diw0801	3023	5575	10	0.24	0.00		—		0.69	0.00		—		0.68	0.00		—	
diw0819	10553	20066	32	0.70	0.00		—		2.53	0.00		—		2.50	0.00		—	
diw0820	11749	22384	37	1.83	76.47	8987	17117	37	5.27	0.00		—		5.28	0.00		—	
Average:				0.35	11.57				0.88	0.00				0.88	0.00			

Table 3: Reduction Results for the DIW-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
dmxa0296	233	386	12	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
dmxa0368	2050	3676	18	0.15	11.51	254	423	11	0.12	0.00		—		0.12	0.00		—	
dmxa0454	1848	3286	16	0.06	0.00		—		0.13	0.00		—		0.13	0.00		—	
dmxa0628	169	280	10	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
dmxa0734	663	1154	11	0.02	0.00		—		0.01	0.00		—		0.03	0.00		—	
dmxa0848	499	861	16	0.03	8.71	49	75	7	0.05	0.00		—		0.05	0.00		—	
dmxa0903	632	1087	10	0.05	30.73	202	334	9	0.08	0.00		—		0.07	0.00		—	
dmxa1010	3983	7108	23	0.14	0.00		—		0.14	0.00		—		0.14	0.00		—	
dmxa1109	343	559	17	0.02	6.44	23	36	6	0.01	0.00		—		0.01	0.00		—	
dmxa1200	770	1383	21	0.09	26.54	216	367	19	0.04	0.00		—		0.03	0.00		—	
dmxa1304	298	503	10	0.01	0.00		—		0.02	0.00		—		0.01	0.00		—	
dmxa1516	720	1269	11	0.02	0.00		—		0.03	0.00		—		0.02	0.00		—	
dmxa1721	1005	1731	18	0.03	0.00		—		0.04	0.00		—		0.04	0.00		—	
dmxa1801	2333	4137	17	0.22	36.35	842	1504	17	0.32	0.00		—		0.33	0.00		—	
			Average:	0.06	8.59				0.07	0.00				0.07	0.00			

Table 4: Reduction Results for the DMXA-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
gap1307	342	552	17	0.01	0.00		—		0.02	0.00		—		0.02	0.00		—	
gap1413	541	906	10	0.01	0.00		—		0.02	0.00		—		0.01	0.00		—	
gap1500	220	374	17	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
gap1810	429	702	17	0.02	0.00		—		0.02	0.00		—		0.01	0.00		—	
gap1904	735	1256	21	0.02	0.00		—		0.06	0.00		—		0.06	0.00		—	
gap2007	2039	3548	17	0.13	16.80	355	596	15	0.10	0.00		—		0.12	0.00		—	
gap2119	1724	2975	29	0.09	0.00		—		0.14	0.00		—		0.14	0.00		—	
gap2740	1196	2084	14	0.04	0.00		—		0.09	0.00		—		0.07	0.00		—	
gap2800	386	653	12	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
gap2975	179	293	10	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
gap3036	346	583	13	0.02	0.00		—		0.02	0.00		—		0.03	0.00		—	
gap3100	921	1558	11	0.04	0.00		—		0.10	0.00		—		0.10	0.00		—	
gap3128	10393	18043	104	1.28	24.95	2590	4501	89	3.39	0.00		—		3.42	0.00		—	
			Average:	0.13	3.21				0.31	0.00				0.31	0.00			

Table 5: Reduction Results for the GAP-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$
msm0580	338	541	11	0.04	12.01	44	65	7	0.03	0.00		—		0.02	0.00		—	
msm0654	1290	2270	10	0.02	0.00		—		0.07	0.00		—		0.06	0.00		—	
msm0709	1442	2403	16	0.03	0.00		—		0.09	0.00		—		0.08	0.00		—	
msm0920	752	1264	26	0.04	0.00		—		0.07	0.00		—		0.06	0.00		—	
msm1008	402	695	11	0.06	23.02	96	160	9	0.02	0.00		—		0.03	0.00		—	
msm1234	933	1632	13	0.02	0.00		—		0.04	0.00		—		0.04	0.00		—	
msm1477	1199	2078	31	0.05	0.00		—		0.06	0.00		—		0.04	0.00		—	
msm1707	278	478	11	0.01	0.00		—		0.02	0.00		—		0.01	0.00		—	
msm1844	90	135	10	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm1931	875	1522	10	0.01	0.00		—		0.04	0.00		—		0.04	0.00		—	
msm2000	898	1562	10	0.03	0.00		—		0.02	0.00		—		0.03	0.00		—	
msm2152	2132	3702	37	0.38	23.10	501	855	25	0.24	0.00		—		0.23	0.00		—	
msm2326	418	723	14	0.01	0.00		—		0.01	0.00		—		0.02	0.00		—	
msm2492	4045	7094	12	0.21	6.74	288	478	11	0.32	0.00		—		0.34	0.00		—	
msm2525	3031	5239	12	0.10	0.00		—		0.24	0.00		—		0.23	0.00		—	
msm2601	2961	5100	16	0.22	23.12	676	1179	14	0.38	0.00		—		0.36	0.00		—	
msm2705	1359	2458	13	0.02	7.08	107	174	10	0.05	0.00		—		0.05	0.00		—	
msm2802	1709	2963	18	0.09	0.00		—		0.06	0.00		—		0.09	0.00		—	
msm2846	3263	5783	89	0.47	59.10	1939	3418	86	0.61	1.57	58	91	22	0.62	1.57	58	91	22
msm3277	1704	2991	12	0.04	0.00		—		0.05	0.00		—		0.05	0.00		—	
msm3676	957	1554	10	0.02	0.00		—		0.02	0.00		—		0.03	0.00		—	
msm3727	4640	8255	21	0.21	9.50	473	784	19	0.51	0.00		—		0.50	0.00		—	
msm3829	4221	7255	12	0.57	24.66	1049	1789	12	1.39	0.00		—		1.35	0.00		—	
msm4038	237	390	11	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm4114	402	690	16	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm4190	391	666	16	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm4224	191	302	11	0.02	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm4312	5181	8893	10	1.05	44.57	2290	3964	10	2.79	0.00		—		2.75	0.00		—	
msm4414	317	476	11	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
msm4515	777	1358	13	0.03	0.00		—		0.05	0.00		—		0.05	0.00		—	
			Average:	0.13	7.76				0.24	0.05				0.24	0.05			

Table 6: Reduction Results for the MSM-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	time	% of $\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$
alue2087	1244	1971	34	0.05	0.00		—		0.07	0.00		—		0.07	0.00		—	
alue2105	1220	1858	34	0.02	0.00		—		0.05	0.00		—		0.06	0.00		—	
alue3146	3626	5869	64	0.30	26.02	889	1527	57	0.29	0.00		—		0.32	0.00		—	
alue5067	3524	5560	68	0.29	21.60	737	1201	65	0.70	0.00		—		0.71	0.00		—	
alue5345	5179	8165	68	0.59	55.59	2717	4539	68	3.19	1.90	98	155	36	3.14	1.87	98	153	36
alue5623	4472	6938	68	0.65	51.67	2150	3585	66	1.69	1.07	49	74	25	1.60	1.07	49	74	25
alue5901	11543	18429	68	1.03	22.73	2487	4188	68	2.72	0.83	96	153	31	2.84	0.80	94	148	31
alue6179	3372	5213	67	0.30	12.01	403	626	54	0.61	0.00		—		0.61	0.00		—	
alue6457	3932	6137	68	0.62	27.90	1060	1712	63	0.80	0.00		—		0.83	0.00		—	
alue6735	4119	6696	68	0.38	26.28	1065	1760	64	0.69	0.00		—		0.71	0.00		—	
alue6951	2818	4419	67	0.30	30.21	831	1335	67	0.57	0.00		—		0.58	0.00		—	
alue7065	34046	54841	544	10.50	90.00	28859	49355	497	88.14	1.05	353	576	138	88.32	1.00	340	547	138
alue7066	6405	10454	16	0.84	64.65	3935	6758	11	6.08	0.00		—		6.15	0.00		—	
alue7080	34479	55494	2344	27.53	88.80	28838	49276	1856	45.24	4.67	1600	2593	613	49.44	4.54	1562	2520	605
alue7229	940	1474	34	0.02	0.00		—		0.04	0.00		—		0.04	0.00		—	
			Average:	2.89	34.50				10.06	0.63				10.36	0.62			

Table 7: Reduction Results for the ALUE-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
alut0787	1160	2089	34	0.05	0.00		—		0.04	0.00		—		0.05	0.00		—	
alut0805	966	1666	34	0.11	2.70	30	45	7	0.07	0.00		—		0.07	0.00		—	
alut1181	3041	5693	64	0.37	49.22	1500	2802	57	0.36	0.00		—		0.36	0.00		—	
alut2010	6104	11011	68	0.91	41.05	2523	4520	66	1.04	0.00		—		1.08	0.00		—	
alut2288	9070	16595	68	1.45	39.69	3661	6587	67	2.54	0.00		—		2.57	0.00		—	
alut2566	5021	9055	68	0.53	43.25	2192	3916	67	1.98	0.82	50	74	22	1.98	0.82	50	74	22
alut2610	33901	62816	204	11.69	74.72	23823	46933	198	130.63	0.87	313	545	75	132.52	0.51	194	319	68
alut2625	36711	68117	879	21.08	93.81	34221	63899	824	156.91	5.05	1985	3442	457	165.68	3.10	1281	2111	425
alut2764	387	626	34	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
			Average:	4.02	38.27				32.62	0.75				33.81	0.49			

Table 8: Reduction Results for the ALUT-Instances

instance	original size			classical reductions					extended reductions, fast variants					extended reductions, strong variants				
	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$	time	% of $\|E\|$	$\|V\|$	$\|E\|$	$\|R\|$
lin01	53	80	4	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
lin02	55	82	6	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
lin03	57	84	8	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
lin04	157	266	6	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
lin05	160	269	9	0.01	0.00		—		0.01	0.00		—		0.02	0.00		—	
lin06	165	274	14	0.01	0.00		—		0.01	0.00		—		0.01	0.00		—	
lin07	307	526	6	0.01	0.00		—		0.02	0.00		—		0.02	0.00		—	
lin08	311	530	10	0.01	0.00		—		0.03	0.00		—		0.03	0.00		—	
lin09	313	532	12	0.01	0.00		—		0.05	0.00		—		0.04	0.00		—	
lin10	321	540	20	0.01	0.00		—		0.03	0.00		—		0.03	0.00		—	
lin11	816	1460	10	0.05	7.05	65	103	6	0.17	0.00		—		0.17	0.00		—	
lin12	818	1462	12	0.05	0.00		—		0.22	0.00		—		0.21	0.00		—	
lin13	822	1466	16	0.03	0.00		—		0.12	0.00		—		0.12	0.00		—	
lin14	828	1472	22	0.04	0.00		—		0.18	0.00		—		0.19	0.00		—	
lin15	840	1484	34	0.04	0.00		—		0.17	0.00		—		0.16	0.00		—	
lin16	1981	3633	12	0.16	12.00	265	436	11	0.44	0.00		—		0.43	0.00		—	
lin17	1989	3641	20	0.10	0.00		—		0.54	0.00		—		0.55	0.00		—	
lin18	1994	3646	25	0.38	44.93	925	1638	24	1.10	0.80	21	29	10	1.15	0.80	21	29	10
lin19	2010	3662	41	0.28	23.51	500	861	28	1.09	0.00		—		1.08	0.00		—	
lin20	3675	6709	11	0.18	0.00		—		1.33	0.00		—		1.30	0.00		—	
lin21	3683	6717	20	0.39	21.85	851	1468	18	0.97	0.00		—		1.01	0.00		—	
lin22	3692	6726	28	0.41	27.43	1087	1845	27	1.85	0.00		—		1.77	0.00		—	
lin23	3716	6750	52	0.97	59.16	2252	3993	49	2.60	0.00		—		2.60	0.00		—	
lin24	7998	14734	16	1.16	37.79	3106	5568	16	8.87	0.00		—		8.81	0.00		—	
lin25	8007	14743	24	1.20	48.27	3974	7117	24	12.28	0.00		—		12.23	0.00		—	
lin26	8013	14749	30	1.99	61.28	4973	9038	30	15.21	0.00		—		15.14	0.00		—	
lin27	8017	14753	36	1.25	43.35	3597	6396	36	13.29	0.57	54	84	19	13.45	0.56	54	83	19
lin28	8062	14798	81	2.44	66.85	5494	9892	78	61.20	12.13	1026	1795	63	154.73	2.99	273	442	58
lin29	19083	35636	24	6.46	43.60	8555	15536	24	41.20	0.00		—		41.16	0.00		—	
lin30	19091	35644	31	4.55	48.79	9591	17392	31	82.22	0.00		—		82.32	0.00		—	
lin31	19100	35653	40	6.49	58.10	11342	20714	39	359.70	12.03	2362	4288	31	1084.54	0.47	97	166	26
lin32	19112	35665	53	6.29	62.32	12168	22226	53	200.88	26.40	5188	9414	52	3514.53	0.42	94	150	31
lin33	19177	35730	117	7.50	70.66	13803	25246	115	203.70	23.09	4592	8251	99	1680.11	1.11	239	395	66
lin34	38282	71521	34	19.52	59.91	23332	42850	34	900.92	16.97	6605	12137	30	6490.52	0.16	69	111	24
lin35	38294	71533	45	22.09	64.90	24322	46427	45	834.63	23.36	9140	16707	44	9246.80	0.24	104	175	30
lin36	38307	71546	58	18.50	57.30	22420	40995	57	451.85	32.28	12628	23097	57	10805.50	23.83	9345	17047	57
lin37	38418	71657	172	27.60	81.46	31647	58375	170	731.21	43.14	16858	30914	168	16019.90	27.89	10955	19983	156
			Average:	3.52	27.04				106.17	5.16				1329.21	1.58			

Table 9: Reduction Results for the LIN-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
taq0014	6466	11046	128	5326	2.38
taq0023	572	963	11	621	0.07
taq0365	4186	7074	22	1914	0.69
taq0377	6836	11715	136	6393	5.45
taq0431	1128	1905	13	897	0.14
taq0631	609	932	10	581	0.02
taq0739	837	1438	16	848	0.11
taq0741	712	1217	16	847	0.11
taq0751	1051	1791	16	939	0.11
taq0891	331	560	10	319	0.01
taq0903	6163	10490	130	5099	4.70
taq0910	310	514	17	370	0.01
taq0920	122	194	17	210	0.01
taq0978	777	1239	10	566	0.02
				Average:	0.99

Table 10: Optimal Solution of the TAQ-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
diw0234	5349	10086	25	1996	1.15
diw0250	353	608	11	350	0.01
diw0260	539	985	12	468	0.01
diw0313	468	822	14	397	0.01
diw0393	212	381	11	302	0.01
diw0445	1804	3311	33	1363	0.07
diw0459	3636	6789	25	1362	0.12
diw0460	339	579	13	345	0.01
diw0473	2213	4135	25	1098	0.08
diw0487	2414	4386	25	1424	0.13
diw0495	938	1655	10	616	0.04
diw0513	918	1684	10	604	0.03
diw0523	1080	2015	10	561	0.03
diw0540	286	465	10	374	0.01
diw0559	3738	7013	18	1570	0.35
diw0778	7231	13727	24	2173	0.81
diw0779	11821	22516	50	4440	6.39
diw0795	3221	5938	10	1550	0.86
diw0801	3023	5575	10	1587	0.68
diw0819	10553	20066	32	3399	2.52
diw0820	11749	22384	37	4167	5.37
				Average:	0.89

Table 11: Optimal Solution of the DIW-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
dmxa0296	233	386	12	344	0.02
dmxa0368	2050	3676	18	1017	0.12
dmxa0454	1848	3286	16	914	0.14
dmxa0628	169	280	10	275	0.01
dmxa0734	663	1154	11	506	0.03
dmxa0848	499	861	16	594	0.05
dmxa0903	632	1087	10	580	0.07
dmxa1010	3983	7108	23	1488	0.14
dmxa1109	343	559	17	454	0.02
dmxa1200	770	1383	21	750	0.03
dmxa1304	298	503	10	311	0.01
dmxa1516	720	1269	11	508	0.02
dmxa1721	1005	1731	18	780	0.02
dmxa1801	2333	4137	17	1365	0.34
				Average:	0.07

Table 12: Optimal Solution of the DMXA-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
gap1307	342	552	17	549	0.02
gap1413	541	906	10	457	0.02
gap1500	220	374	17	254	0.01
gap1810	429	702	17	482	0.02
gap1904	735	1256	21	763	0.05
gap2007	2039	3548	17	1104	0.12
gap2119	1724	2975	29	1244	0.13
gap2740	1196	2084	14	745	0.08
gap2800	386	653	12	386	0.02
gap2975	179	293	10	245	0.01
gap3036	346	583	13	457	0.02
gap3100	921	1558	11	640	0.10
gap3128	10393	18043	104	4292	3.36
				Average:	0.30

Table 13: Optimal Solution of the GAP-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
msm0580	338	541	11	467	0.02
msm0654	1290	2270	10	823	0.06
msm0709	1442	2403	16	884	0.09
msm0920	752	1264	26	806	0.06
msm1008	402	695	11	494	0.03
msm1234	933	1632	13	550	0.04
msm1477	1199	2078	31	1068	0.07
msm1707	278	478	11	564	0.02
msm1844	90	135	10	188	0.01
msm1931	875	1522	10	604	0.04
msm2000	898	1562	10	594	0.03
msm2152	2132	3702	37	1590	0.23
msm2326	418	723	14	399	0.02
msm2492	4045	7094	12	1459	0.34
msm2525	3031	5239	12	1290	0.24
msm2601	2961	5100	16	1440	0.38
msm2705	1359	2458	13	714	0.04
msm2802	1709	2963	18	926	0.06
msm2846	3263	5783	89	3135	0.67
msm3277	1704	2991	12	869	0.05
msm3676	957	1554	10	607	0.03
msm3727	4640	8255	21	1376	0.48
msm3829	4221	7255	12	1571	1.38
msm4038	237	390	11	353	0.01
msm4114	402	690	16	393	0.02
msm4190	391	666	16	381	0.01
msm4224	191	302	11	311	0.01
msm4312	5181	8893	10	2016	2.77
msm4414	317	476	11	408	0.01
msm4515	777	1358	13	630	0.05
				Average:	0.24

Table 14: Optimal Solution of the MSM-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
alue2087	1244	1971	34	1049	0.07
alue2105	1220	1858	34	1032	0.05
alue3146	3626	5869	64	2240	0.30
alue5067	3524	5560	68	2586	0.70
alue5345	5179	8165	68	3507	3.35
alue5623	4472	6938	68	3413	1.64
alue5901	11543	18429	68	3912	2.89
alue6179	3372	5213	67	2452	0.62
alue6457	3932	6137	68	3057	0.83
alue6735	4119	6696	68	2696	0.68
alue6951	2818	4419	67	2386	0.57
alue7065	34046	54841	544	23881	83.21
alue7066	6405	10454	16	2256	6.14
alue7080	34479	55494	2344	62449	98.02
alue7229	940	1474	34	824	0.04
				Average:	13.27

Table 15: Optimal Solution of the ALUE-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
alut0787	1160	2089	34	982	0.05
alut0805	966	1666	34	958	0.06
alut1181	3041	5693	64	2353	0.36
alut2010	6104	11011	68	3307	1.06
alut2288	9070	16595	68	3843	2.64
alut2566	5021	9055	68	3073	2.02
alut2610	33901	62816	204	12239	92.49
alut2625	36711	68117	879	35459	215.48
alut2764	387	626	34	640	0.01
				Average:	34.91

Table 16: Optimal Solution of the ALUT-Instances

instance	size			opt	time
	$\|V\|$	$\|E\|$	$\|R\|$		
lin01	53	80	4	503	0.01
lin02	55	82	6	557	0.01
lin03	57	84	8	926	0.01
lin04	157	266	6	1239	0.01
lin05	160	269	9	1703	0.01
lin06	165	274	14	1348	0.02
lin07	307	526	6	1885	0.03
lin08	311	530	10	2248	0.03
lin09	313	532	12	2752	0.05
lin10	321	540	20	4132	0.05
lin11	816	1460	10	4280	0.18
lin12	818	1462	12	5250	0.22
lin13	822	1466	16	4609	0.13
lin14	828	1472	22	5824	0.18
lin15	840	1484	34	7145	0.17
lin16	1981	3633	12	6618	0.44
lin17	1989	3641	20	8405	0.55
lin18	1994	3646	25	9714	1.17
lin19	2010	3662	41	13268	1.08
lin20	3675	6709	11	6673	1.28
lin21	3683	6717	20	9143	1.00
lin22	3692	6726	28	10519	1.81
lin23	3716	6750	52	17560	2.63
lin24	7998	14734	16	15076	8.89
lin25	8007	14743	24	17803	12.19
lin26	8013	14749	30	21757	15.24
lin27	8017	14753	36	20678	13.37
lin28	8062	14798	81	32584	85.26
lin29	19083	35636	24	23765	41.26
lin30	19091	35644	31	27684	82.08
lin31	19100	35653	40	31696	1117.09
lin32	19112	35665	53	39832	3497.84
lin33	19177	35730	117	56061	1682.76
lin34	38282	71521	34	45018	6518.15
lin35	38294	71533	45	50559	9214.52
lin36	38307	71546	58	55608	564932.00
lin37	38418	71657	172	99560	153972.00
				Average:	20032.53

Table 17: Optimal Solution of the LIN-Instances

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-2001-4-005	H.P.A. Lensch, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models
MPI-I-2001-4-004	S.W. Choi, H. Seidel	Linear One-sided Stability of MAT for Weakly Injective Domain
MPI-I-2001-4-003	K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel	Efficient Light Transport Using Precomputed Visibility
MPI-I-2001-4-002	H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web
MPI-I-2001-4-001	H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, H. Seidel	Image-Based Reconstruction of Spatially Varying Materials
MPI-I-2001-2-006	H. Nivelle, S. Schulz	Proceeding of the Second International Workshop of the Implementation of Logics
MPI-I-2001-2-005	V. Sofronie-Stokkermans	Resolution-based decision procedures for the universal theory of some classes of distributive lattices with operators
MPI-I-2001-2-004	H. de Nivelle	Translation of Resolution Proofs into Higher Order Natural Deduction using Type Theory
MPI-I-2001-2-003	S. Vorobyov	Experiments with Iterative Improvement Algorithms on Completely Unimodel Hypercubes
MPI-I-2001-2-002	P. Maier	A Set-Theoretic Framework for Assume-Guarantee Reasoning
MPI-I-2001-2-001	U. Waldmann	Superposition and Chaining for Totally Ordered Divisible Abelian Groups
MPI-I-2001-1-004	S. Hert, M. Hoffmann, L. Kettner, S. Pion, M. Seel	An Adaptable and Extensible Geometry Kernel
MPI-I-2001-1-003	M. Seel	Implementation of Planar Nef Polyhedra

MPI-I-2001-1-002	U. Meyer	Directed Single-Source Shortest-Paths in Linear Average-Case Time
MPI-I-2001-1-001	P. Krysta	Approximating Minimum Size 1,2-Connected Networks
MPI-I-2000-4-003	S.W. Choi, H. Seidel	Hyperbolic Hausdorff Distance for Medial Axis Transform
MPI-I-2000-4-002	L.P. Kobbelt, S. Bischoff, K. Kähler, R. Schneider, M. Botsch, C. Rössl, J. Vorsatz	Geometric Modeling Based on Polygonal Meshes
MPI-I-2000-4-001	J. Kautz, W. Heidrich, K. Daubert	Bump Map Shadows for OpenGL Rendering
MPI-I-2000-2-001	F. Eisenbrand	Short Vectors of Planar Lattices Via Continued Fractions
MPI-I-2000-1-005	M. Seel, K. Mehlhorn	Infimaximal Frames A Technique for Making Lines Look Like Segments
MPI-I-2000-1-004	K. Mehlhorn, S. Schirra	Generalized and improved constructive separation bound for real algebraic expressions
MPI-I-2000-1-003	P. Fatourou	Low-Contention Depth-First Scheduling of Parallel Computations with Synchronization Variables
MPI-I-2000-1-002	R. Beier, J. Sibeyn	A Powerful Heuristic for Telephone Gossiping
MPI-I-2000-1-001	E. Althaus, O. Kohlbacher, H. Lenhof, P. Müller	A branch and cut algorithm for the optimal solution of the side-chain placement problem
MPI-I-1999-4-001	J. Haber, H. Seidel	A Framework for Evaluating the Quality of Lossy Image Compression
MPI-I-1999-3-005	T.A. Henzinger, J. Raskin, P. Schobbens	Axioms for Real-Time Logics
MPI-I-1999-3-004	J. Raskin, P. Schobbens	Proving a conjecture of Andreka on temporal logic
MPI-I-1999-3-003	T.A. Henzinger, J. Raskin, P. Schobbens	Fully Decidable Logics, Automata and Classical Theories for Defining Regular Real-Time Languages
MPI-I-1999-3-002	J. Raskin, P. Schobbens	The Logic of Event Clocks
MPI-I-1999-3-001	S. Vorobyov	New Lower Bounds for the Expressiveness and the Higher-Order Matching Problem in the Simply Typed Lambda Calculus
MPI-I-1999-2-008	A. Bockmayr, F. Eisenbrand	Cutting Planes and the Elementary Closure in Fixed Dimension
MPI-I-1999-2-007	G. Delzanno, J. Raskin	Symbolic Representation of Upward-closed Sets
MPI-I-1999-2-006	A. Nonnengart	A Deductive Model Checking Approach for Hybrid Systems
MPI-I-1999-2-005	J. Wu	Symmetries in Logic Programs
MPI-I-1999-2-004	V. Cortier, H. Ganzinger, F. Jacquemard, M. Veanes	Decidable fragments of simultaneous rigid reachability
MPI-I-1999-2-003	U. Waldmann	Cancellative Superposition Decides the Theory of Divisible Torsion-Free Abelian Groups
MPI-I-1999-2-001	W. Charatonik	Automata on DAG Representations of Finite Trees

MPI-I-1999-1-007	C. Burnikel, K. Mehlhorn, M. Seel	A simple way to recognize a correct Voronoi diagram of line segments
MPI-I-1999-1-006	M. Nissen	Integration of Graph Iterators into LEDA
MPI-I-1999-1-005	J.F. Sibeyn	Ultimate Parallel List Ranking ?
MPI-I-1999-1-004	M. Nissen, K. Weihe	How generic language extensions enable "open-world" desing in Java
MPI-I-1999-1-003	P. Sanders, S. Egner, J. Korst	Fast Concurrent Access to Parallel Disks
MPI-I-1999-1-002	N.P. Boghossian, O. Kohlbacher, H.-. Lenhof	BALL: Biochemical Algorithms Library
MPI-I-1999-1-001	A. Crauser, P. Ferragina	A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory
MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic