Math. Prog. Comp. (2018) 10:487–532
https://doi.org/10.1007/s12532-018-0135-8

FULL LENGTH PAPER

The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study

Daniel Juhl[1] · **David M. Warme**[2] · **Pawel Winter**[1] · **Martin Zachariasen**[1]

Received: 15 May 2015 / Accepted: 11 January 2018 / Published online: 20 February 2018

Abstract The GeoSteiner software package has for about 20 years been the fastest (publicly available) program for computing exact solutions to Steiner tree problems in the plane. The computational study by Warme, Winter and Zachariasen, published in 2000, documented the performance of the GeoSteiner approach—allowing the exact solution of Steiner tree problems with more than a thousand terminals. Since then, a number of algorithmic enhancements have improved the performance of the software package significantly. We describe these (previously unpublished) enhancements, and present a new computational study wherein we run the current code on the largest problem instances from the 2000-study, and on a number of larger problem instances. The computational study is performed using the commercial GeoSteiner 4.0 code base, and the performance is compared to the publicly available GeoSteiner 3.1 code base as well as the code base from the 2000-study. The software studied in the paper is being released as GeoSteiner 5.0 under an open source license.

Keywords Euclidean Steiner tree problem · Rectilinear Steiner tree problem · Fixed orientation Steiner tree problem · Exact algorithm · Computational study

✉ Daniel Juhl
juhl.daniel@gmail.com

David M. Warme
david@warme.net

Pawel Winter
pawel@di.ku.dk

Martin Zachariasen
martinz@di.ku.dk

[1] Department of Computer Science, University of Copenhagen, 2100 Copenhagen Ø, Denmark

[2] Group W. Inc., 2650 Park Tower Drive, Suite 500, Vienna, VA 22180, USA

Mathematics Subject Classification 90C10 Integer programming · 90C27 Combinatorial optimization · 05C05 Trees · 05C65 Hypergraphs · 51N20 Euclidean analytic geometry · 68W35 VLSI algorithms

1 Introduction

The Steiner tree problem in the plane asks for a shortest possible interconnection of a set of points under some given metric. The Euclidean and rectilinear Steiner tree problems in the plane are by far the most studied geometric Steiner tree problem variants [11,12,14]. Recently, the uniform and fixed orientation metrics have also received some attention due to applications in the physical design of integrated circuits [4,5,26]. A uniform orientation metric is given by a set of $\lambda \geq 2$ uniformly distributed orientations in the plane, and the goal is to compute a shortest possible interconnection where all line segments have one of the given orientations. The rectilinear metric is a uniform orientation metric with two legal orientations, namely the horizontal and vertical orientations. The Steiner tree problem in the plane is NP-hard for all interesting metrics [6,9,10]. For a comprehensive introduction to these problems, see the book by Brazil and Zachariasen [7].

Historical context Winter's groundbreaking work [27] introduced the present two-phase approach (consisting of FST generation followed by FST concatenation, described below) for the Euclidean Steiner tree problem, naming his software GeoSteiner. Salowe adapted this same two-phase approach to the rectilinear Steiner tree problem, with Warme providing an efficient implementation [21]. Winter and Zachariasen then presented an improved Euclidean algorithm called GeoSteiner96 [28]. The publication of Warme's dissertation [24] in 1998 provoked some controversy. The computational bottleneck for these methods had been the FST concatenation phase. The breakthrough reported in [24] was to: (a) identify FST concatenation as an instance of the minimum spanning tree in hypergraph problem; (b) formulate MST in hypergraph as an integer program; and (c) solve this integer program via branch-and-cut. The previous computational state of the art for the rectilinear problem had been about 16 points [23], which Warme's Master's work increased to 35 points [21]. For the Euclidean problem, the state of the art was about 150 points [28]. Warme's dissertation increased this to 1000 points rectilinear and 2000 points Euclidean—results that some researchers apparently felt might be "too good to be true." The controversy ended after Althaus independently implemented Warme's methods [1], successfully reproducing Warme's computational results. The Euclidean results reported in [24] used the recent Euclidean full Steiner tree generator of Winter and Zachariasen [28], who kindly agreed to collaborate with Warme. Shortly thereafter, Warme, Winter and Zachariasen published an extensive computational study of these methods [25], and then combined their software into a single integrated open-source package known as GeoSteiner 3.0. The code of Althaus has fallen by the wayside, while GeoSteiner has continued to improve by several orders of magnitude (versions 3.1, 4.0 and 5.0). The 11th DIMACS challenge reaffirmed GeoSteiner's preeminence on two-dimensional geometric Steiner tree problems—nobody else chose to compete

in *any* of the problem categories that GeoSteiner entered, which is why GeoSteiner does not appear in any of the official results of the DIMACS challenge.

Background and motivation Since the publication of the computational study by Warme, Winter and Zachariasen [25] in 2000, the GeoSteiner software package—first released in 1999—has been the fastest (publicly available) program for computing exact solutions to Steiner tree problems in the plane. The GeoSteiner approach has successfully been extended to uniform orientation metrics [19], to Euclidean and rectilinear problems with obstacles [13,32], and to rectilinear group interconnection problems [30].

The computational effectiveness of the GeoSteiner approach is largely due to the fact that a minimum Steiner tree in the plane can be decomposed into *small* so-called full Steiner trees (FSTs); these are subtrees where Steiner points are interior vertices and terminals are leaves [25]. The GeoSteiner algorithm has two phases: FST generation and FST concatenation. In the first phase, a (small) superset of the FSTs in a minimum Steiner tree is determined. In the second phase, a subset of the generated FSTs is chosen such that their union forms a minimum Steiner tree. The first phase is very metric dependent, while the second phase is purely combinatorial (and metric independent).

Since the publication of the 2000-paper [25], a number of (unpublished) algorithmic enhancements have improved the performance of the software package significantly. Some of these enhancements were part of the most recent public release in 2001, while others have been added in later versions. These algorithmic engineering efforts cover the use of faster data structures, new heuristics for better pruning of FSTs, and better cutting methods in the branch-and-cut concatenation algorithm. The motivation of the current paper is to describe these improvements and document their effects, both for the publicly available GeoSteiner 3.1 code base and the commercial GeoSteiner 4.0 code base. During the final preparation of this paper, GeoSteiner version 5.0 was released under an open source license. GeoSteiner 5.0 is functionally identical to version 4.0 as described in Sect. 5.

Our contribution We present an updated computational study on Steiner tree problems in the plane. Also, we document the main algorithmic enhancements made to the GeoSteiner software package. In the updated computational study we run the current code on the largest problem instances from the 2000-study, and on a number of even larger problem instances. The computational study is performed using the commercial GeoSteiner 4.0 code base, and the performance is compared to the publicly available GeoSteiner 3.1 code base as well as the code base from the 2000-study.

In most cases, the FST generation phase has a more predictable running time than the FST concatenation phase. As a consequence, the FST concatenation phase is usually the bottleneck when solving large-scale problem instances. The FST concatenation problem can either be solved as a minimum spanning tree problem in a hypergraph (as in the 2000-paper), or as a Steiner tree problem in an ordinary graph. The latter approach was studied and experimentally evaluated by Polzin and Vahdati Daneshmand [20] in 2003. At that time it appeared to have superior performance when compared the FST concatenation code of GeoSteiner 3.1. The present version of GeoSteiner studied herein vastly outperforms GeoSteiner 3.1, so those comparisons have little if any meaning today. We did not study the graph approach in the current

paper because no code for the Steiner problem in graphs was available to us prior to the DIMACS Challenge Workshop (including [20]), so we have restricted our experiments to solving the FST concatenation problem as a minimum spanning tree problem in a hypergraph. The competitiveness of the Steiner problem in graphs approach remains an important open question, however, and we strongly encourage researchers studying the Steiner problem in graphs to test their methods on these same instances. We have taken the following steps to support such efforts:

- We have made four series of large-scale Euclidean Steiner tree problems available as instances of the Steiner tree problem in graphs.[1]
- The latest publicly released versions of GeoSteiner provide the capability of generating such graph instances (both rectilinear and Euclidean metrics) for arbitrary point sets.

Organisation of the paper In Sect. 2 we give a brief introduction to the GeoSteiner approach. A presentation of the major algorithmic enhancements made since the publication of the 2000-paper are given in Sect. 3. Computational results are presented in Sect. 4, and concluding remarks are given in Sect. 5.

2 GeoSteiner approach

Let N be a finite set of points, so-called terminals, in the plane, and assume that some metric is given. A naive algorithm for computing a minimum Steiner tree for N under the given metric is to enumerate all full Steiner topologies for every subset of terminals, and then compute an FST for the given topology, or to decide that no such tree exists [7,15]. A minimum Steiner tree is then obtained by identifying a subset among the constructed FSTs that interconnects N and has minimum length. However, the number of terminal subsets is exponential in the size of N—and the number of full Steiner topologies for each subset is super-exponential—so this algorithm would scale poorly.

GeoSteiner follows the same two-phase approach as the naive algorithm, but reduces the work significantly by implicit instead of explicit enumeration of FSTs (for all subsets and all full Steiner topologies). Groups of FSTs that do not fulfill necessary structural properties are eliminated early—and in most cases without direct geometric construction.

2.1 FST generation

The task of the FST generation phase is to determine an as small as possible superset of FSTs of a minimum Steiner tree. Consider some FST T spanning a subset of k terminals in N, where $3 \leq k \leq |N|$. We can assume that the Steiner points in T have degree 3. Steiner points with degree 4 or more can only appear under certain metrics, and in these cases they can be assumed to appear in FSTs that are easily identified [5].

[1] Link to graph instances: http://dimacs11.zib.de/downloads.html.

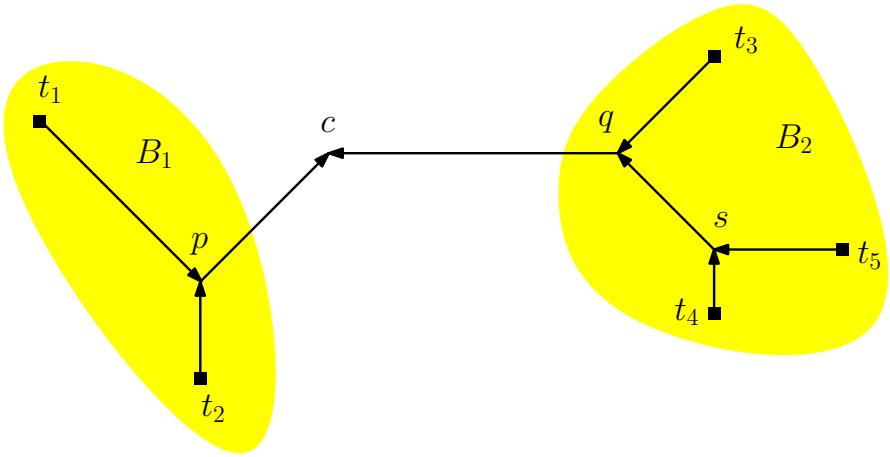

Fig. 1 An FST T spanning five terminals $t_1, \ldots, t_5$ with a bent edge pq. T can be obtained by joining two branch trees B_1 and B_2 with roots p and q, respectively. The arrows indicate the paths from the terminals to the root of each branch tree. Branch tree B_1 is obtained by joining branch trees for terminals t_1 and t_2. Branch tree B_2 is obtained by joining a branch tree for terminal t_3 with a branch tree having root s that spans terminals t_4 and t_5. The example is given for a fixed orientation metric

The FST T therefore has k terminals as leaf nodes (with degree 1) and $k-2$ Steiner points as interior nodes (with degree 3).

For the Euclidean metric the edges in T—which interconnect terminals and/or Steiner points—are *straight* line segments, and the edges at a Steiner point in T meet at 120° angles. For the rectilinear metric, and more generally for any fixed orientation metric, the edges of T can also be assumed to be straight edges, except possibly one so-called *bent* edge pq, which interconnects nodes p and q using exactly two straight line segments that meet at a *corner point* [5].

FST generation in GeoSteiner is performed by enumerating so-called *branch trees*. Consider an FST T. If T has a bent edge pq, then let c be the corner point of pq; otherwise let c be the midpoint of any (straight) edge pq in T. Imagine that we cut edge pq at point c. We obtain two branch trees: B_1 rooted at p having a stem (or ray) leaving p along pc, and B_2 rooted at q having a stem leaving q along qc (Fig. 1). Note that all edges in a branch tree are *straight* edges.

Intuitively, we can think of a branch tree as an FST spanning a set of terminals and a single point at infinity. We define the *size* of a branch tree to be the number of terminals spanned by the branch tree. A branch tree of size 1 consists of a single terminal with a stem; such a branch tree has no Steiner points and the terminal is the root of the branch tree. A branch tree of size $k \geq 2$ has k terminals and $k-1$ Steiner points, and can naturally be represented as a binary tree. Branch trees for the Euclidean problem are represented by *equilateral points*, and the root of a branch tree can be located anywhere on a so-called *Steiner arc* [28].

The generation phase of GeoSteiner enumerates branch trees of increasing size—essentially using a dynamic programming approach. Branch trees of size k are obtained by joining branch trees of size l with branch trees of size $k-l$, where l runs from

1 to $\lfloor k/2 \rfloor$. An FST is obtained by identifying two branch trees where the stems can be made to overlap in opposite directions (for the Euclidean metric) or intersect at an appropriate angle (for fixed orientation metrics).

Branch trees and FSTs are eliminated from consideration by performing a number of *pruning tests*. These tests are metric dependent, but are based on similar structural properties such as empty regions and upper bounds on edge lengths in an FST [19,28,29]. A simple upper bound on the length of *any* edge in a minimum Steiner tree for N is to take the longest edge in a minimum spanning tree for N [28]; in the following we let b_{max} denote this upper bound for a given set of terminals N.

2.2 FST concatenation

The FST concatenation problem can be modelled as an instance of the minimum spanning tree in hypergraph (MSTHG) problem: given a hypergraph $\mathbf{H} = (V, \mathbf{E})$, and a weight function $w : \mathbf{E} \mapsto \mathbb{R}$, find a subset $\mathbf{T} \subseteq \mathbf{E}$ such that $\mathbf{T}$ is a spanning tree of $\mathbf{H}$ that minimizes $w(\mathbf{T}) = \sum_{\mathbf{e}\in\mathbf{T}} w(\mathbf{e})$. FST concatenation instantiates V as the set of terminals N, $\mathbf{E}$ as the set of FSTs (each FST is a hyperedge connecting 2 or more vertices in the hypergraph), and $w(\mathbf{e})$ as the geometric length of FST $\mathbf{e}$ using the appropriate distance metric. Note that even deciding the existence (or not) of a spanning tree within a general hypergraph is NP-complete [24], so FST concatenation is in general an NP-hard optimization problem.

The MSTHG problem is solved using the following integer program (IP):

$$\text{Minimize} \quad \sum_{\mathbf{e}\in E} c_{\mathbf{e}} x_{\mathbf{e}}$$

Subject to

$$\sum_{\mathbf{e}\in\mathbf{E}} (|\mathbf{e}| - 1)\, x_{\mathbf{e}} \;=\; |V| - 1, \tag{1}$$

$$\sum_{\mathbf{e}\in\mathbf{E}} \max(|\mathbf{e} \cap S| - 1,\, 0)\, x_{\mathbf{e}} \;\le\; |S| - 1, \quad \forall S \subset V,\ |S| \ge 2 \tag{2}$$

$$0 \le x_{\mathbf{e}} \le 1, \quad \forall \mathbf{e} \in \mathbf{E} \tag{3}$$

$$x_{\mathbf{e}} \in \mathbb{Z}, \quad \forall \mathbf{e} \in \mathbf{E} \tag{4}$$

where $c \in \mathbb{R}^{|\mathbf{E}|}$ is the cost vector such that $c_{\mathbf{e}} = w(\mathbf{e})$ for all $\mathbf{e} \in \mathbf{E}$, and $x \in \mathbb{R}^{|\mathbf{E}|}$ is the solution vector. Constraint (1) ensures that the spanning tree has the right number and sizes of hyperedges, while the so-called subtour constraints (2) ensure that the tree has no cycles. The following two theorems were proved by Warme [24]:

Theorem 1 *Let $\mathbf{T} \subseteq \mathbf{E}$ be any spanning tree of $\mathbf{H} = (V, \mathbf{E})$. Let $\tilde{x} \in \mathbb{R}^{|\mathbf{E}|}$ such that $\tilde{x}_{\mathbf{e}} = 1$ if $\mathbf{e} \in \mathbf{T}$ and $\tilde{x}_{\mathbf{e}} = 0$ otherwise ($\tilde{x}$ is the incidence vector of $\mathbf{T}$). Then $\tilde{x}$ is an integer feasible solution to IP.*

Theorem 2 *Let $\tilde{x}$ be any feasible integer solution to IP. Let $\mathbf{T} = \{\mathbf{e} \in \mathbf{E} : x_{\mathbf{e}} = 1\}$, so that $\tilde{x}$ is the incidence vector of $\mathbf{T}$. Then $\mathbf{T}$ is a spanning tree of $\mathbf{H}$.*

The LP relaxation of IP is obtained by removing constraint (4). This relaxation produces very tight lower bounds in practice. Integrality is recovered via branch-and-bound upon fractional variables of the LP relaxation. Note that there are exponentially many constraints (2). These constraints are added to the formulation dynamically as violations are discovered. This separation problem can be solved in polynomial time by reduction to flows using the techniques described in [24].

3 Algorithmic enhancements

The 2000-paper [25] combined the Euclidean FST generation algorithm from [28], the rectilinear FST generation algorithm from [29], and the branch-and-cut based FST concatenation algorithm described in [24]. All generated FSTs were considered in the branch-and-cut algorithm. No FST pruning (or hypergraph reduction) was used, and the algorithms were not integrated at that time. The GeoSteiner implementation has improved considerably since the publication of [25], rendering those results obsolete.

Table 1 summarizes the major versions of GeoSteiner, their release dates, and major feature improvements, and gives the subsequent section number that describes each feature in more detail. In order to adequately explain the improved computational results, we catalog *all* of the changes made to GeoSteiner across these versions regardless of whether previously published or not. Except for Sects. 3.2 and 3.9, all enhancements described in Sect. 3 have never previously been published.

Note that GeoSteiner 4.0 is a proprietary product (not open source), but has been supplanted by GeoSteiner 5.0 that is functionally identical and available under an open source license as described in Sect. 5.

Table 1 GeoSteiner versions, release dates, enhancements and section number where corresponding enhancement is described

Version	Date	Major new features and improvements	Section
[25]	March 1998		
3.0	January 1999	New implementation of FST generators	3.1
		No compiled-in limits on problem size	
		Save/restore LP basis when switching nodes	
		Handle abstract hypergraph problems	
3.1	February 2001	New greedy Euclidean heuristic	3.2
		Numeric stability improvements	3.3
		FST pruning added	3.4
		Improved variable fixing in branch-and-cut	3.5
		Improved strong branching heuristics	3.6
4.0	February 2006	Callable library	3.7
		New bottleneck Steiner distance data structure	3.8
		Uniform orientations FST generator	3.9
		Cut generation improvements	3.10
		Local cuts	3.11

Several of these improvements were empirically motivated based upon computational experience—as the problem instances that we attempted to solve grew in size, new computational bottlenecks appeared within the code, whether in time or space.

3.1 New FST generator implementations

The original Euclidean and rectilinear FST generators were implemented in C++ using LEDA [17,18]. These were replaced with new highly tuned implementations written in C. Both of the new FST generators replaced the LEDA and other data structures with more efficient, low-level data structures that are customized to provide only the necessary operations, and are much more friendly to modern cache hierarchies and deeply pipelined CPU architectures. They also use a much better hash table and linear time check for FSTs spanning the same subset of terminals (only the shortest FST is retained for a given subset of terminals). The reimplemented rectilinear FST generator was about 50–70 times faster than the original at 10,000 points. Similar speedups were achieved in the Euclidean FST generator.

3.1.1 Improvements to the Euclidean FST generator

The new Euclidean FST generator uses a range search idea originally suggested by Althaus [1] to identify relevant pairwise branch trees as follows. Let b_{max} be the length of the longest edge in a minimum spanning tree for N—which is the same as the maximum bottleneck Steiner distance (see Sect. 3.8). Then no edge in any minimum Steiner tree for N can be longer than b_{max}.

Consider two branch trees B_1 and B_2. Let γ be the distance between the roots of B_1 and B_2 (see Sect. 2.1); if B_1 and B_2 are terminals, γ is simply the distance between these terminals. Clearly, if $\gamma > 2b_{max}$ then no feasible combination of B_1 and B_2 exists, since at least one of the new edges would have a length greater than b_{max}. Thus, given a branch tree B_1, we can use a range search data structure to identify all branch trees that are within distance $2b_{max}$ of B_1 (where distance is defined as above). A simple bucket structure is used as data structure: the minimum rectangle containing the set of terminals N is divided into $K \times K$ subrectangles. One bucket structure is constructed for each size of branch trees, allowing fast identification of branch trees of appropriate size. This method significantly improves the running time for large problem instances, especially those with a uniform distribution of the terminals.

In addition, many uses of trigonometric functions were replaced with vector operations (e.g., dot and cross product), and more efficient data structures were implemented for representing the terminals spanned by branch trees (or equilateral points).

3.1.2 Improvements to the rectilinear FST generator

The so-called empty rectangles matrix was reorganized as a lower-triangular bit matrix to reduce its size and improve locality (see [29] for details). In addition, several crucial low-level geometric primitives (macros) were replaced with new implementations that use no conditional branches. By reducing pipeline flushes, these macro changes

produced about a 3-fold additional speed improvement on top of all the other improvements.

3.2 New greedy Euclidean Steiner tree heuristic

The Zachariasen–Winter greedy heuristic for Euclidean Steiner trees [31] was added as an alternative to the existing Smith–Lee–Liebman heuristic [22].

3.3 Numerical stability improvements to Euclidean FST generator

These improvements were empirically motivated by discovering "supposedly optimal" solutions that violated certain easily checkable optimality conditions, e.g., the requirement that all angles be ≥ 120 degrees.

Several rounds of changes were made to improve the numerical stability of the Euclidean FST generator. First, dynamically computed (relative) tolerances are used for all floating-point comparisons. Behavior for point sets located away from the origin is improved by translating the entire point set to a new origin computed to be both "central" to the point set and to have relatively few significant bits. Next, all computations for a single equilateral point are performed with respect to an origin located at one of its terminals, thereby increasing the number of significant bits available for these computations. All movement of Steiner arc endpoints are encapsulated into new routines that are much more careful to make sure that the arc is never shortened any more than can be justified numerically.

Finally, the code now uses the GNU multi-precision arithmetic package (GMP) under control of a *Level* switch, which defaults to level 0. At Levels 1 and 2, GMP exact rational arithmetic is used to compute: (1) the exact coordinates (in $\mathbb{Q}(\sqrt{3})$) of each equilateral point; (2) the closest double-precision representation of each equilateral point's coordinates; and (3) the closest double-precision representation of the length of each FST (in $\sqrt{\mathbb{Q}(\sqrt{3})}$). Conversion of $\mathbb{Q}(\sqrt{3})$ into numeric form is accomplished using a Newton iteration, with initial approximation computed via floating-point. At Level 1, a single high-precision Newton iteration is used to refine this value before converting to the nearest double-precision representation. At Level 2, Newton iterations are continued until a formal convergence test has verified that the value is correct to at least 1/2 ULP of double-precision. These computations are performed only on equilateral points and FSTs that have passed all screening tests, as a part of storing them in their final form. This prevents floating-point errors from accumulating as equilateral points are recursively constructed from smaller equilateral points. This in turn permits more conventional relative numeric tolerancing techniques to be used in a sound manner in the more intensively traversed portions of the code. On large random instances, GMP Level 2 adds only about 1–2% to the run time, but significantly improves the numerical accuracy and stability of the generated FSTs.

Prior to these improvements, FST candidates were occasionally eliminated incorrectly, possibly causing the true optimal solution to be missed. Other FST candidates were sometimes retained when they should have been eliminated, making the final set

of FSTs unnecessarily large. Even the length of individual FSTs could be incorrect by substantial factors, once again causing the true optimal solution to be missed.

The improved FST generator assures that only provably sub-optimal FSTs are eliminated, as many FSTs as validly possible *are* eliminated, and that the computed length of every FST is correct to within 1/2 ULP of double-precision arithmetic.

3.4 New FST pruning

The FST pruning algorithm of Fößmeier and Kaufmann [8] was implemented for all metrics. (The previous pruning codes were slow, relatively ineffective, and completely specific to either the Euclidean or rectilinear metric [21,28].) The idea of the Fößmeier and Kaufmann pruning algorithm is to test whether an FST T must be part of a larger full component in a minimum Steiner tree. More specifically, the pruning algorithm searches for an FST T and a terminal t with the following properties:

1. Terminal t is not spanned by T, and has (minimum) distance γ to some edge (p, q) in T.
2. The longest edge on every path P from t to p, or from t to q, is longer than γ (where the edges of P come from the set of generated FSTs).

Assuming that T appears in a minimum Steiner tree leads to a contradiction, since we can add a connection of length γ from t to (p, q), and remove an edge of length more than γ on the cycle that is created. Therefore a shorter tree can be constructed, and T can be eliminated from consideration.

Efficient implementation of this pruning algorithm requires significant care, but the method is very powerful in practice and typically removes at least half of the generated FSTs. Here we give some of the most important implementation details:

1. First the so-called *pruning graph* $G_P = (V_P, E_P)$ is computed. The vertex set V_P consists of the terminals N and the Steiner points in all generated FSTs. The edge set E_P consists of the edges of all generated FSTs. The edge set is sorted by non-decreasing length (under the given metric), and necessary information is attached to the edge set such that a sorted list of edges efficiently can be generated for any given *subset* of FSTs.
2. For each FST T the list of *compatible* FSTs is generated. These are FSTs that can appear together with T in some minimum Steiner tree. Consider some FST $T' \neq T$. If T' has two or more terminals in common with T, then T' is not compatible with T. Consider the case where T' has a single terminal in common with T; note that $T \cup T'$ forms a tree that interconnects a subset of terminals. If a shorter tree interconnecting the terminals spanned by $T \cup T'$ can be constructed, then T' is not compatible with T; first simple heuristics and finally an exact algorithm based on the FSTs already generated is used for constructing a tree interconnecting the terminals spanned by $T \cup T'$.
3. For each FST T the set of terminals that are within distance b_{max} to T, so-called *close* terminals, are identified (see Sect. 2.1). The distance from a terminal t to an edge (p, q) in T is computed as the difference in length between a minimum Steiner tree for t, p and q, and the length of edge (p, q), that is, the increase in

length that is needed to insert t on edge (p, q). The list of close terminals is sorted by their minimum distance to T (shortest distance first).

4. When attempting to prune an FST T, the sorted list of close terminals is traversed. Consider a terminal t having distance γ to some edge (p, q) in T. A (forest) graph $G_T = (V_T, E_T)$ with vertex set $V_T = V_P$ is constructed; the edge set E_T is formed by using Kruskal's minimum spanning tree algorithm on the sorted edge set E_P—but only including edges from FSTs that are compatible with T. As soon as an edge of length γ or more is reached, Kruskal's algorithm stops. Now, if t and p, as well as t and q, are in different connected components in G_T, then T can be pruned. Note that the graph G_T has to be built only once for the sorted list of close terminals to T.

Clearly, as FSTs are pruned, the list of compatible FSTs for each remaining FST must be recomputed. This is done at appropriate intervals in order to save running time.

In addition to pruning FSTs, the pruning algorithm identifies FSTs that are *required* to be part of any minimum Steiner tree. If a terminal t only is spanned by a single remaining FST T, then T must be part of any minimum Steiner tree. More generally, if a connected component of required FSTs has a single adjacent FST, then this FST is required. Finally, any FST that is not compatible with a required FST can be pruned.

3.5 Improved variable fixing in branch-and-cut

Variable fixing in the branch-and-cut algorithm for solving the FST concatenation problem was improved (see Sect. 2.2). Let Z be the current optimal LP objective value for a node, $d_{\mathbf{e}}$ be the reduced costs for edge $\mathbf{e}$, and U be the best upper-bound available. Previous variable fixing was activated when a non-basic edge variable $x_{\mathbf{e}}$ satisfied $Z + |d_{\mathbf{e}}| > U + \epsilon$, with small $\epsilon > 0$. The new implementation maintains new vectors Z_{lb0} and Z_{lb1}. Element $\mathbf{e}$ of these vectors stores the highest value of $Z + |d_{\mathbf{e}}|$ ever seen while $x_{\mathbf{e}}$ is non-basic at 0 (or 1), respectively, and represent lower bounds on the objective value obtained by forcing $x_{\mathbf{e}} = 0$ (or $x_{\mathbf{e}} = 1$), respectively. This allows variables to be fixed even when the instantaneous value of $Z + |d_{\mathbf{e}}|$ is not sufficiently high to produce a cutoff. The quantities Z_{lb0} and Z_{lb1} are also used in the branch variable selection methods, as discussed below.

3.6 Improved strong branching heuristics

A new heuristic ranking of branch variable candidates in the branch-and-cut algorithm for the FST concatenation problem was introduced. Each node maintains a vector $bheur$. Element $bheur_{\mathbf{e}}$ indicates how much the corresponding value of $x_{\mathbf{e}}$ has been changing in recent iterations. We set $bheur = 0$ initially. Let x be the current LP solution, x' be the previous LP solution, and $bheur'$ be the previous $bheur$. Then

$$bheur = 0.75\, bheur' + |x - x'|.$$

For each fractional branch variable candidate $x_{\mathbf{e}}$, we compute a closest rational approximation $N_{\mathbf{e}}/D_{\mathbf{e}}$, and then

$$rank_{\mathbf{e}} = bheur_{\mathbf{e}}\left((D_{\mathbf{e}} - 1) + |N_{\mathbf{e}} - D_{\mathbf{e}}/2|\right).$$

Candidate branch variables are sorted into increasing order by $rank_{\mathbf{e}}$. Strong branch testing is performed in that order. This favors variables that have been "stuck" at "simple fractional values" (e.g., 1/2, 3/4, etc.) for several iterations. We test variables in this order until either: (1) one or more variables are fixed; (2) a cutoff is obtained; (3) all candidates are tested; or (4) when K consecutive variables have been tested without finding an improving candidate, where $K = 2\lfloor \log_2(NumFrac) \rfloor$, and $NumFrac$ is the number of fractional branch variable candidates.

Finally, improvements were made to the interplay between the primal upper bound heuristic, variable fixing, and branch variable testing. The primal upper bound heuristic is applied to every LP solution obtained while testing branch variables. (This heuristic seeks to construct a good integer feasible solution from a given fractional LP solution, and is based upon a simple generalization of Kruskal's algorithm to hypergraphs.) New upper bounds discovered in this way provide opportunities for additional variable fixing. Any branch variable candidate that gets fixed in this manner while testing the candidates causes branch variable selection to abort, allowing cut generation to resume at the current node in lieu of branching.

3.7 Callable library

In GeoSteiner 4.0, the code was completely restructured and re-factored to encapsulate the algorithms as a subroutine library, allowing GeoSteiner 4.0 to be much more easily used as a "black box" within other applications.

3.8 New bottleneck Steiner distance data structure

Consider a minimum spanning tree $\overline{T}$ for N. The bottleneck Steiner distance between two terminals $t_1, t_2 \in N$ is the length of the *longest edge* on the path between t_1 and t_2 in $\overline{T}$. Bottleneck Steiner distances between every pair of terminals can trivially be determined in $O(n^2)$ time by doing a depth-first traversal in $\overline{T}$ from every terminal. This gives constant-time lookup of bottleneck Steiner distances, but requires $\Theta(n^2)$ space. For large problem instances, it is more efficient to use the data structure suggested in [16] which uses $\Theta(n)$ space; the preprocessing time is $O(n \log n)$, and queries can be made in $O(\log n)$ time. The FST generators use this $\Theta(n)$ space data structure when the number of terminals is greater than 100.

3.9 FST generator for uniform orientation metrics

A new FST generator for uniform orientation metrics was added. A description of this algorithm appears in [19]. This FST generator was coded in C using similar techniques as the existing Euclidean and rectilinear FST generators.

3.10 Improved cut generation

Significant enhancements were made to the separation procedures in the branch-and-cut algorithm that affect which constraints are (and are *not*) generated.

First, Eq. (1) in the integer programming formulation (see Sect. 2.2) offers an opportunity to re-express inequalities (2) in other equivalent forms. In particular, the code now chooses either inequality (2), or the equivalent form obtained by subtracting (2) from Eq. (1)—the form having the least number of non-zero coefficients is used.

Second, cutset contraints [24] are never used any more. These were previously generated in response to zero weight cuts (i.e., two or more connected components within the support hypergraph). Instead of cutset ($S : V \setminus S$), the code now generates two subtour constraints, one for S and the other for $V \setminus S$. The equivalent form encoding technique ensures that these two subtour constraints have approximately the same number of non-zero coefficients even though $|S|$ and $|V \setminus S|$ may be vastly different. This helps keep the LP sparse.

These changes significantly decrease the number of optimize/separate iterations needed to converge to an optimal subtour relaxation, and also result in LP instances that are more sparse and that solve more quickly.

3.11 Local cuts

The branch-and-cut algorithm was improved by adding *local cuts* — originally devised by Applegate et al. [2] for the traveling salesman problem. Local cuts are significant in that they do not conform to the "template paradigm" wherein violations are sought among a family of inequalities a having a certain fixed structure. Given a fractional solution $\bar{x}$, local cuts attempt to generate a cutting plane by defining a linear projection from the original problem space U to a new space U' of significantly lower dimension. Let $\bar{x}'$ be the projection of $\bar{x}$ onto U'. The separation problem is then solved in U', for which two outcomes are possible: (a) a hyperplane h is found in U' that separates $\bar{x}'$ from the projections of *all* integer feasible solutions—in this case, hyperplane h is lifted from the low dimensional space U' back to the original problem space U; or (b) $\bar{x}'$ is found to be a convex combination of some subset of the projected integer feasible solutions, in which case no separating hyperplane exists, and generation of local cuts fails.

An iterative procedure is used to decide which of these cases is true. It maintains a set F containing a "reasonably small number" of projected integer feasible solutions, and proceeds by alternating two steps: (1) Identify a prospective hyperplane h by solving an LP in U' that separates $\bar{x}'$ from F. If the optimal solution shows that $\bar{x}'$ is a convex combination of F, then $\bar{x}'$ is in the convex hull of projected integer feasible solutions and local cut generation fails. Otherwise, h is a hyperplane that separates $\bar{x}'$ from F. (2) Test the validity of h by solving an IP that maximizes the violation of h over *all* of the projected integer feasible solutions. The optimal solution f of this IP either proves that hyperplane h is a valid inequality, or f is a projected integer feasible solution that violates hyperplane h—in which case f is added to set F before returning to step (1). This iteration terminates after a reasonable number of iterations because U' is a space of relatively low dimension. There are conditions that can assure that the generated hyperplane h is always facet-defining in space U', but this does not guarantee that the final inequality is facet-defining when lifted back into the original space U.

We adapted local cuts for the spanning tree in hypergraph problem as follows. For a suitably chosen subset $S \subset V$ of vertices, the projection is obtained by considering the sub-hypergraph induced by S. Let $\mathbf{H} = (V, \mathbf{E})$ be a hypergraph. Let $STP(\mathbf{H})$ be the spanning tree in hypergraph $\mathbf{H}$ polytope (i.e., the convex hull of all incidence vectors corresponding to spanning trees of $\mathbf{H}$).

Let $\bar{x}$ be a fractional LP solution vector to separate from $STP(\mathbf{H})$. Let $S \subset V$, and consider the sub-hypergraph $\mathbf{H}' = (S, \mathbf{E}')$ of $\mathbf{H}$ induced by S. Let $\mathbf{T} \subseteq \mathbf{E}$ be any spanning tree of $\mathbf{H}$, and consider its image in $\mathbf{H}'$, which will always be a forest (possibly empty, i.e., no edges). Let $FP(\mathbf{H}')$ be the forest in hypergraph $\mathbf{H}'$ polytope (i.e., the convex hull of all incidence vectors corresponding to forests in hypergraph $\mathbf{H}'$). Let $\bar{x}'$ be the image of $\bar{x}$ with respect to $\mathbf{H}'$. We produce a constraint $\bar{y}\,x \leq 1$ that separates $\bar{x}'$ from $FP(\mathbf{H}')$, or prove that none exists because $\bar{x}' \in FP(\mathbf{H}')$.

We start with a set F of forests in $\mathbf{H}'$, represented as incidence vectors. (Initially, F contains single-edge forests, one for each edge in $\mathbf{E}'$.) Consider the following linear program:

$$\begin{aligned} &\text{Maximize} \quad Z(y) = \bar{x}'\,y \\ &\text{Subject to} \\ &\qquad f\,y \leq 1 \qquad \forall f \in F \end{aligned} \tag{LP1}$$

Let $\bar{y}$ be an optimal solution to this LP with objective value $\bar{Z}$. If $\bar{Z} \leq 1$, then $\bar{x}' \in \text{conv}(F) \subset FP(\mathbf{H}')$, and we are done—no separating hyperplane exists. Otherwise $\bar{Z} > 1$ and $\bar{y}\,x \leq 1$ separates $\bar{x}'$ from F—but this constraint may not be valid for $FP(\mathbf{H}')$. We test validity by solving the following maximum-weight forest problem:

$$\begin{aligned} &\text{Maximize} \quad Q(f) = \bar{y}\,f \\ &\text{Subject to} \\ &\qquad f \in FP(\mathbf{H}') \\ &\qquad f \in \{0,1\}^{|\mathbf{E}'|} \end{aligned} \tag{IP1}$$

This is an NP-hard optimization problem that we easily reduce to the MST in hypergraph problem as follows. First produce a minimization problem by negating the edge weights. Construct a new hypergraph $\mathbf{H}''$ from $\mathbf{H}'$ by adding 1 new vertex t and a zero weight edge from t to each $v \in S$. There is an obvious one-to-one correspondence between each spanning tree in $\mathbf{H}''$ and the corresponding forest in $\mathbf{H}'$. Let $\bar{f}$ be any feasible integer solution to IP1 having the objective

$$Q(\bar{f}) = \bar{y}\,\bar{f} > 1.$$

Then $\bar{f} \in FP(\mathbf{H}')$ is a certificate that the constraint $\bar{y}\,x \leq 1$ is not valid for $FP(\mathbf{H}')$. So add $\bar{f}$ to set F and start over—solve a new instance of LP1, find a new constraint $\bar{y}\,x \leq 1$, and test its validity against the forests of $\mathbf{H}'$. If the *optimal* solution of IP1 has objective $Q(f) \leq 1$, then this serves as a certificate to the validity of constraint $\bar{y}\,x \leq 1$ with respect to polytope $FP(\mathbf{H}')$. This constraint lifts directly back into the

larger space of $STP(\mathbf{H})$ as follows. Let $\mathbf{e}' \in \mathbf{E}'$. Copy the coefficient $\bar{y}_{\mathbf{e}'}$ to every edge $\mathbf{e} \in \mathbf{E}$ such that $\mathbf{e} \cap S = \mathbf{e}'$. ($|\mathbf{e}'| \geq 2$ by definition.) Let the coefficient be zero for every edge $\mathbf{e} \in \mathbf{E}$ such that $|\mathbf{e} \cap S| \leq 1$. Note that it is not always necessary to obtain "optimal" solutions for subproblem IP1—any solution f having $Q(f) > 1$ suffices. We use greedy heuristics to try and find a suitable forest f. Only when these heuristics fail do we resort to invoking the branch-and-cut recursively to solve IP1 (after reducing it to MST in $\mathbf{H}''$). This recursive invocation of the branch-and-cut terminates early (before proving optimality) whenever a forest f satisfying $Q(f) > 1$ is discovered. Note further that it is generally more efficient to compute $\bar{y}$ by solving the dual formulation of LP1, since the basis matrix for the dual remains fixed in size instead of growing with $|F|$.

This procedure can be quite efficient when $\dim(\mathbf{H}') \ll \dim(\mathbf{H})$, meaning that we must focus on making $|S|$ sufficiently small. We apply this procedure only to LP solutions having no violated subtour constraints. The separation routines for subtour constraints perform various reductions on the support hypergraph before the formal flow-based separation algorithms are applied. When no subtour violations are present, many of these same reductions can be applied to arrive at "fractional components" upon which generation of local cuts might be attempted. By default, local cuts are attempted on any component for which the induced $\mathbf{H}' = (S, \mathbf{E}')$ satisfies $|S| \leq 80$ and $|\mathbf{E}'| \leq 256$.

These cuts can be very strong in practice. It is not unusual for a single round of local cuts to close 10% or more of the gap at the root node. Rounds are repeated until either the gap is closed or no further local cuts can be generated (either because the $\bar{x}'$ is already a convex combination of forests, or because the components become too big to attempt local cuts upon). The default settings for maximum $|S|$ and $|\mathbf{E}'|$ are a tradeoff between the desire to obtain strong cuts, the cost of obtaining the cuts, and the effectiveness of the branching. It is useful to remember that expending large amounts of CPU time to compute a single cut does not automatically make that cut "strong."

4 Computational experience

In this section we present our computational results. First we describe the experimental setup. Then we describe the performance of the best configuration, GeoSteiner 4.0 using FST pruning, on four sets of benchmark instances. Then we compare some older code bases to the most recent one; the goal is to illustrate the algorithmic improvements of the GeoSteiner software package.

4.1 Experimental setup

The computational benchmarking was performed on twelve paravirtualized guests sharing a HP ProLiant BL685c generation 7 server. This model has 256 GB of memory and four AMD Opteron 6380 CPUs, each having sixteen cores running at a base clock rate of 2.5 GHz. The twelve guests were running the 64-bit version of Debian 7.4.0 while the host was running OpenSuse 12.3. CPLEX version 12.5.1 was used as LP solver. The source code was compiled with GNU C 4.4.5-8 for Debian, with

optimization flag -O2. Because GeoSteiner implements its own native branch-and-cut framework, it uses CPLEX only as an LP solver,[2] and generally uses default settings for all CPLEX LP solver algorithmic parameters.

For the generation of Euclidean FSTs, we use the `-g` and `-m 2` flags. These indicate, respectively, that the Zachariasen–Winter greedy heuristic (see Sect. 3.2) should be used instead of the Smith–Lee–Liebman heuristic, and that precision should be increased by using GMP Level 2 (see Sect. 3.3). For solving the FST concatenation problem, the `-L` flag was used, which enables the usage of local cuts (see Sect. 3.11). No other parameters were used in the benchmark runs.

In the tables of computational results, certain columns have the same meaning within every table in which the column appears. These columns are defined as follows:

FST counts	Gen	Number of generated FSTs
	Prun	Number of FSTs after FST pruning
	Req	Number of required FSTs (FSTs that belong to any minimum Steiner tree)
SMT properties	NumF	Number of FSTs in minimum Steiner tree
	SizeF	Average size of FSTs in minimum Steiner tree
	Red	Reduction over minimum spanning tree (in percent)
FST conc	Gap	Root LP value vs. optimal value (gap in percent)
	Nds	Number of branch-and-bound nodes
	LPs	Number of LPs solved
CPU time	Gen	CPU time for FST generation (seconds)
	Prun	CPU time for FST pruning (seconds)
	Conc	CPU time for FST concatenation (seconds)
	Total	Total CPU time for FST generation, FST pruning and FST concatenation (seconds).

4.2 Benchmark instances

The benchmark instances are divided into four main sets.

1. *Randomly generated instances.* A series of instances of size 1000, 2000, ... 10,000, with 15 randomly generated instances of each size (150 instances in total). These were generated using the `rand_points` program of the GeoSteiner package (randomly and uniformly distributed points in a square).
2. *OR-Library instances.* A subset of the *estein* instance sets from the *OR-Library* [3] consisting of 31 randomly generated instances: 15 containing 500 terminals, 15 containing 1000 terminals and a single instance containing 10,000 terminals. All instances are randomly and uniformly distributed points in a square.
3. *TSPLIB instances.* A set consisting of 46 problem instances from TSPLIB, a library of instances for the traveling salesman problem which are mainly drawn from the

[2] The GeoSteiner branch-and-cut framework was written in 1995–1998. At that time the CPLEX MIP solver was too primitive to provide good support for user-provided separation algorithms, primal heuristics and other plugins currently taken for granted—so we wrote our own branch-and-cut framework using CPLEX as just an LP solver. This also made it easier to support other LP solvers such as lp_solve.

real world. This benchmark sets contain all instances with 500 or more terminals used in [20,25] (plus the smaller TSPLIB instances from [25] for comparison purposes).

4. *Large-scale randomly generated instances.* A series of instances of size 25,000 (25k), 50,000 (50k) and 100,000 (100k), with 15 randomly generated instances of each size (45 instances in total). These were generated using the `rand_points` program of the GeoSteiner package (randomly and uniformly distributed points in a square).

All problem instances, except the large-scale instances, were solved under four different metrics: the *Euclidean* metric, the *rectilinear* metric (two uniform orientations), the *hexagonal* metric (three uniform orientations) and the *octilinear metric* (four uniform orientations). The large-scale instances were only solved for the Euclidean metric.

4.3 Performance of best configuration across all metrics

In this section we report the computational performance using the best configuration of the code, namely GeoSteiner 4.0 using FST pruning and local cuts. Each of the three phases—FST generation, FST pruning and FST concatenation—was allowed to run for at most *24 h* for each problem instance. However, for the TSPLIB instances and the large-scale randomly generated instances, the timeout for each phase was increased to *7 days* for the *Euclidean* metric.

4.3.1 Randomly generated instances

For randomly generated problem instances, the number of FSTs follows a very regular pattern (see Fig. 2 and Tables 2, 3, 4 and 5). For the Euclidean problem, around $2.50n$ FSTs are generated for n terminals; after pruning around $0.76n$ FSTs remain, and among these $0.41n$ FSTs have been identified as required FSTs. Thus FST pruning removes more than two-thirds of the FSTs that were originally generated, and only $0.35n$ FSTs remain undecided in the FST concatenation problem. A minimum Steiner tree has around $0.60n$ FSTs, so approximately $0.19n$ FSTs need to be selected among the $0.35n$ undecided FSTs.

For the rectilinear problem, the FST counts are somewhat higher, but they are still linear in n: around $4.07n$ FSTs are generated, $1.41n$ remain after FST pruning, and $0.12n$ FSTs are identified as required; this leaves $1.29n$ undecided FSTs. For the hexogonal and octilinear metrics, the FST counts are sandwiched between the FST counts for the Euclidean and rectilinear metrics.

The average size of FSTs in a minimum Steiner tree is 2.63 for the hexagonal metric, 2.70 for both the Euclidean and octilinear metrics, and 2.94 for the rectilinear metric. The larger number of generated FSTs for the rectilinear metric can therefore partially be explained by the fact that FSTs in a rectilinear minimum Steiner tree span slightly more terminals on average when compared to the other metrics.

The LP-relaxation of the integer programming formulation for the minimum spanning tree in hypergraph problem provides excellent lower bounds for the problem.

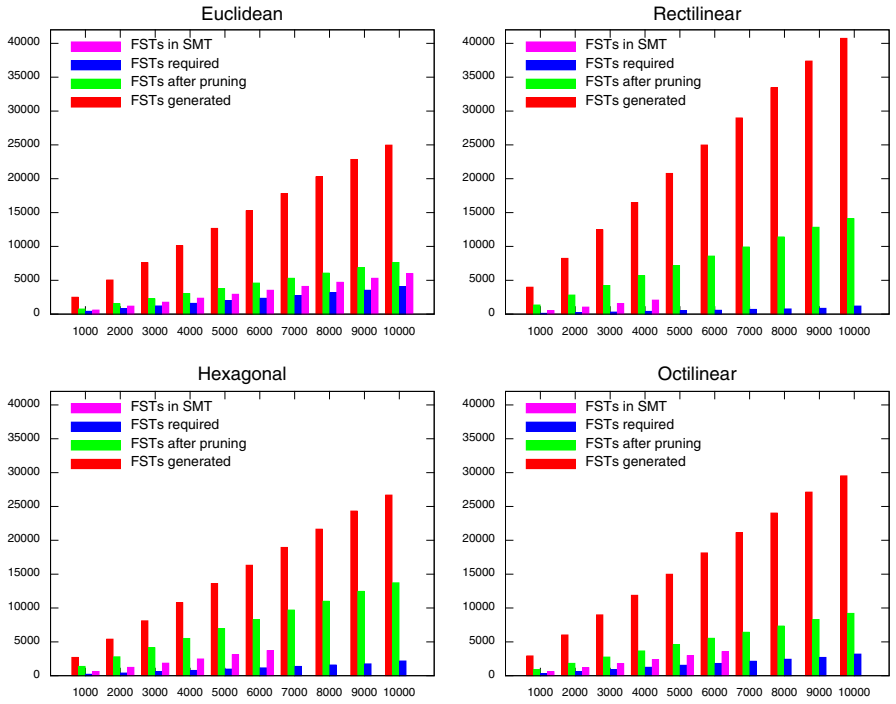

Fig. 2 FST counts for randomly generated problem instances. Averages taken over 15 problem instances for each size

In most cases the gap between the LP-solution and the IP-solution is zero, and the problem is solved at the root node. The number of LPs solved is basically proportional to the running time of the branch-and-cut algorithm. In Fig. 3 we present plots showing the number of LPs solved for the randomly generated problem instances. Several of the larger problem instances did not solve within the time limit of 24 hours, and therefore no data is shown for these problem instances. For the Euclidean metric, all problem instances were solved within 1,000 LP-iterations. For the remaining metrics, some instances required 100,000 or more iterations.

The running time for the randomly generated problem instances, divided into each of the three phases of FST generation, FST pruning and FST concatenation, is shown in Fig. 4. For the Euclidean metric, all problem instances were solved within the time limit of 24 h. For the rectilinear metric, all problem instances of size 4000 or less were solved within the time limit; for the hexagonal and octilinear metrics, all problem instances of size 6000 or less were solved within the time limit.

The FST generation phase dominates the total running time for the smaller problem instances under all metrics except the rectilinear metric. The running time of FST generation and FST pruning scales in a fairly regular way, and is approximately quadratic. For larger problem instances, FST concatenation dominates the total running time for all metrics.

Table 2 Euclidean metric. Randomly generated instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000	2490	751	415	593	2.68	3.19	0.000	1.0	7.0	15.0	2.1	0.1	17.2
	± 51	± 28	± 20	± 8	± 0.03	± 0.11	± 0.000	± 0.0	± 2.8	± 1.1	± 0.2	± 0.0	± 1.2
2000	5036	1504	803	1178	2.70	3.29	0.000	1.0	12.9	37.6	6.1	0.5	44.3
	± 114	± 36	± 44	± 19	± 0.03	± 0.11	± 0.000	± 0.0	± 11.8	± 3.8	± 0.5	± 0.4	± 4.2
3000	7594	2247	1202	1759	2.71	3.30	0.000	1.0	28.0	65.6	11.8	2.3	79.7
	± 102	± 35	± 39	± 16	± 0.02	± 0.06	± 0.000	± 0.0	± 34.7	± 3.4	± 0.8	± 3.0	± 6.1
4000	10,093	3018	1582	2348	2.70	3.27	0.000	1.0	47.9	101.6	20.2	10.7	132.5
	± 137	± 43	± 41	± 16	± 0.01	± 0.06	± 0.000	± 0.0	± 85.2	± 8.0	± 1.1	± 23.0	± 27.6
5000	12,661	3773	1977	2929	2.71	3.29	0.000	1.0	19.4	138.0	29.6	3.0	170.6
	± 169	± 71	± 59	± 29	± 0.02	± 0.07	± 0.000	± 0.0	± 16.8	± 5.2	± 1.0	± 3.0	± 7.0
6000	15,240	4577	2333	3520	2.71	3.30	0.000	1.0	50.6	191.5	43.0	14.4	248.9
	± 178	± 74	± 76	± 25	± 0.01	± 0.05	± 0.000	± 0.0	± 49.3	± 10.8	± 3.1	± 16.0	± 22.7
7000	17,794	5294	2752	4098	2.71	3.32	0.000	1.0	66.5	240.8	57.7	73.5	372.0
	± 211	± 62	± 92	± 35	± 0.01	± 0.06	± 0.000	± 0.0	± 155.9	± 12.2	± 3.1	± 259.6	± 257.1
8000	20,278	6053	3159	4702	2.70	3.30	0.000	1.0	62.0	302.5	73.6	41.6	417.7
	± 175	± 87	± 75	± 30	± 0.01	± 0.04	± 0.000	± 0.0	± 96.1	± 8.7	± 3.0	± 100.9	± 102.7
9000	22,840	6850	3520	5287	2.70	3.27	0.000	1.0	51.5	364.9	94.1	43.1	502.0
	± 149	± 111	± 97	± 25	± 0.01	± 0.04	± 0.000	± 0.0	± 78.2	± 15.5	± 6.6	± 114.0	± 109.3
10,000	24,966	7632	4071	5978	2.68	3.31	0.000	1.0	113.7	423.9	109.7	550.0	1083.5
	± 1563	± 235	± 616	± 402	± 0.09	± 0.04	± 0.000	± 0.0	± 202.2	± 66.3	± 16.1	± 1833.5	± 1843.3

Averages over 15 instances for each size; standard deviations on the second line of each row

Table 3 Rectilinear metric. Randomly generated instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000	3978	1344	115	517	2.93	11.33	0.003	1.8	35.2	0.0	13.6	2.3	16.1
	±98	±51	±13	±10	±0.04	±0.26	±0.009	±2.2	±11.6	±0.0	±1.5	±1.4	±2.3
2000	8239	2820	198	1031	2.94	11.67	0.001	1.7	71.3	0.1	47.5	264.7	312.3
	±291	±145	±23	±16	±0.03	±0.24	±0.003	±2.6	±41.0	±0.0	±7.4	±790.7	±792.6
3000	12,446	4225	300	1544	2.94	11.76	0.001	1.7	134.8	0.3	99.8	537.1	637.2
	±335	±133	±32	±20	±0.03	±0.16	±0.002	±1.2	±66.0	±0.0	±13.0	±963.5	±961.8
4000	16,483	5645	396	2060	2.94	11.63	0.003	7.5	295.7	0.4	168.1	4200.9	4369.4
	±454	±212	±38	±14	±0.01	±0.17	±0.004	±7.9	±288.3	±0.0	±21.0	±11,666.0	±11,663.1
5000	20,775	7134	483	–	–	–	–	–	–	0.6	267.8	–	–
	±440	±213	±41							±0.0	±29.3		
6000	24,981	8576	565	–	–	–	–	–	–	0.9	386.4	–	–
	±496	±237	±40							±0.0	±40.4		
7000	28,973	9914	694	–	–	–	–	–	–	1.1	526.2	–	–
	±614	±220	±43							±0.0	±60.8		
8000	33,429	11,396	761	–	–	–	–	–	–	1.4	708.4	–	–
	±447	±219	±41							±0.0	±94.1		
9000	37,375	12,836	857	–	–	–	–	–	–	1.8	931.3	–	–
	±437	±230	±49							±0.0	±70.4		
10,000	40,740	14,097	1192	–	–	–	–	–	–	2.1	1082.7	–	–
	±3394	±590	±915							±0.3	±206.8		

Averages over 15 instances for each size; standard deviations on the second line of each row

Table 4 Hexagonal metric. Randomly generated instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000	2692	1369	203	614	2.63	4.40	0.000	1.1	61.7	146.3	370.5	45.5	562.2
	± 146	± 49	± 18	± 12	± 0.03	± 0.20	± 0.001	± 0.3	± 125.2	± 27.8	± 118.6	± 164.2	± 258.9
2000	5397	2792	392	1226	2.63	4.48	0.000	1.0	80.3	702.9	918.5	4.4	1625.8
	± 259	± 109	± 40	± 25	± 0.03	± 0.21	± 0.000	± 0.0	± 77.4	± 130.2	± 233.4	± 4.4	± 325.6
3000	8111	4132	587	1850	2.62	4.50	0.000	1.0	151.1	2008.4	1600.5	17.7	3626.5
	± 240	± 81	± 29	± 32	± 0.03	± 0.11	± 0.000	± 0.0	± 234.3	± 274.6	± 306.4	± 32.3	± 517.5
4000	10,776	5497	776	2460	2.62	4.45	0.000	1.0	275.5	3492.0	2323.7	192.8	6026.6
	± 321	± 111	± 44	± 26	± 0.02	± 0.13	± 0.000	± 0.0	± 523.1	± 687.0	± 666.8	± 522.0	± 1584.2
5000	13,609	6942	968	3100	2.61	4.48	0.000	1.0	1596.3	6050.3	2692.5	609.1	9351.8
	± 351	± 142	± 38	± 39	± 0.02	± 0.11	± 0.000	± 0.0	± 3471.6	± 825.9	± 622.2	± 1001.2	± 1402.4
6000	16,318	8298	1152	3691	2.63	4.52	0.000	1.1	1208.8	10,252.2	4093.2	2716.6	17,244.7
	± 441	± 159	± 61	± 45	± 0.02	± 0.10	± 0.000	± 0.3	± 3161.4	± 1618.6	± 1171.3	± 8207.8	± 8230.8
7000	18,953	9693	1362	–	–	–	–	–	–	14,448.6	4652.2	–	–
	± 310	± 155	± 51							± 1650.1	± 1060.9		
8000	21,649	10,999	1581	–	–	–	–	–	–	20,039.4	5443.6	–	–
	± 408	± 181	± 46							± 2472.6	± 874.7		
9000	24,311	12,458	1743	–	–	–	–	–	–	31,418.7	5765.9	–	–
	± 450	± 162	± 58							± 2883.3	± 1227.9		
10,000	26,680	13,726	2172	–	–	–	–	–	–	32,871.7	7660.6	–	–
	± 2153	± 525	± 947							± 13,695.0	± 2388.0		

Averages over 15 instances for each size; standard deviations on the second line of each row

Table 5 Octilinear metric. Randomly generated instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000	2920	891	321	593	2.69	4.49	0.000	1.0	9.9	102.3	218.7	0.5	321.5
	± 115	± 29	± 22	± 10	± 0.03	± 0.15	± 0.000	± 0.0	± 5.2	± 5.0	± 40.7	± 1.4	± 44.1
2000	6017	1824	625	1178	2.70	4.64	0.000	1.0	22.3	479.1	564.6	0.7	1044.5
	± 174	± 52	± 40	± 14	± 0.02	± 0.13	± 0.000	± 0.0	± 16.4	± 31.3	± 130.3	± 0.4	± 144.0
3000	8979	2749	918	1768	2.69	4.63	0.000	1.0	135.3	1229.6	830.8	63.4	2123.9
	± 167	± 52	± 38	± 22	± 0.02	± 0.07	± 0.000	± 0.0	± 200.0	± 41.7	± 109.1	± 165.1	± 132.3
4000	11,869	3648	1237	2362	2.69	4.58	0.000	1.0	266.7	2444.3	1263.3	997.5	4705.1
	± 241	± 70	± 52	± 18	± 0.01	± 0.07	± 0.000	± 0.0	± 917.2	± 113.5	± 274.9	± 3772.9	± 3753.7
5000	14,990	4569	1532	2945	2.70	4.62	0.000	1.0	174.7	5366.7	1412.8	155.6	6935.0
	± 259	± 84	± 67	± 22	± 0.01	± 0.09	± 0.000	± 0.0	± 327.4	± 222.3	± 157.6	± 558.8	± 651.8
6000	18,135	5530	1812	3533	2.70	4.62	0.000	1.0	1196.5	8203.7	2092.3	14,015.6	24,311.6
	± 350	± 115	± 71	± 23	± 0.01	± 0.06	± 0.000	± 0.0	± 4269.7	± 320.1	± 327.3	± 54,200.5	± 53,944.6
7000	21,111	6414	2129	–	–	–	–	–	–	8598.7	2330.2	–	–
	± 451	± 83	± 91							± 344.9	± 389.5		
8000	24,023	7331	2429	–	–	–	–	–	–	11,408.2	2855.4	–	–
	± 355	± 126	± 91							± 372.7	± 409.8		
9000	27,113	8303	2698	–	–	–	–	–	–	20,094.5	3084.9	–	–
	± 381	± 146	± 86							± 626.0	± 561.1		
10,000	29,517	9209	3193	–	–	–	–	–	–	24,259.5	3611.4	–	–
	± 2192	± 92	± 726							± 4412.9	± 549.6		

Averages over 15 instances for each size; standard deviations on the second line of each row

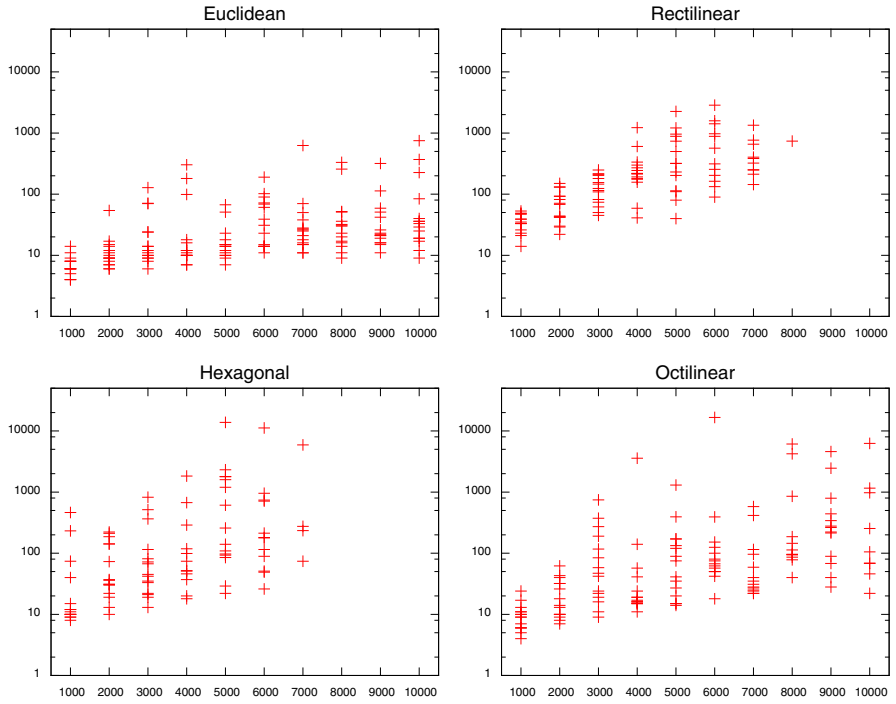

Fig. 3 Number of LPs solved in branch-and-cut algorithm for randomly generated problem instances

4.3.2 OR-Library instances

The OR-Library instances show a very similar behavior as the randomly generated problem instances (see Tables 6, 7, 8, 9); this is not unexpected as they were generated in a similar manner as the randomly generated instances (uniformly in a square).

4.3.3 TSPLIB instances

For the TSPLIB instances the FST counts vary significantly, depending on the distribution of the terminals (see Tables 10, 11, 12, 13). For the Euclidean metric, problem instances where (large subsets of) terminals are aligned in regular grids are particularly difficult, e.g. `pcb442` and `d1291` for which the number of generated FSTs is significantly larger than the average for randomly generated problem instances. Problem instance `p654`, which has its points partly aligned in regular grids and partly in clusters, is the smallest unsolved problem for the Euclidean and hexagonal metrics, and it was also somewhat hard for the octilinear metric (but completely trivial for the rectilinear metric).

As mentioned above, the time limit was increased for the Euclidean metric on the TSPLIB instances in order to identify the practical limit of the algorithm. For FST generation on the Euclidean metric, we allowed around 7 days of computing time; however, in some cases the memory requirements (sometimes in excess of 64 Gb)

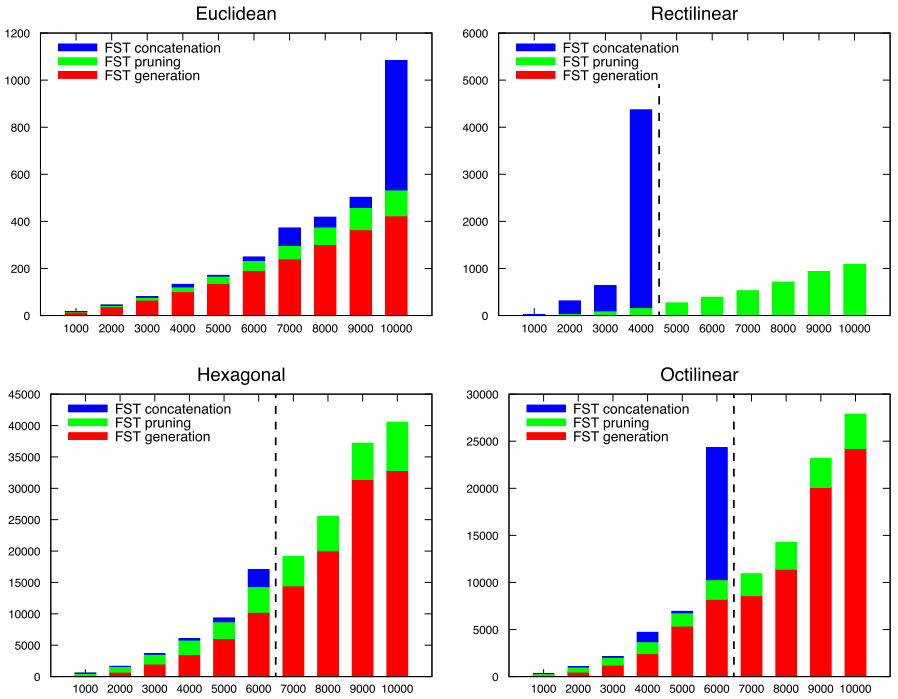

Fig. 4 Running times for randomly generated problem instances. Averages taken over 15 problem instances for each size. The scaling on the y-axis differs across the four figures. The FST concatenation problem was not solved for all of the larger rectilinear, hexagonal and octilinear problem instances—hence no average running times are shown for the larger problem instances. The vertical dashed line indicates the limit for which the FST concatenation problem could not be solved for all problem instances

rather than computing time was the practical limit for some of the problem instances, e.g. `u2319` and `fl3795`.

FST concatenation for the TSPLIB instances is, across all metrics, also significantly harder than for randomly generated instances—mainly due to the increased number of FSTs generated. Nevertheless, the largest LP-relaxation gap is 0.084%, and the largest number of branch-and-bound nodes is 661.

When compared to the 2000-study, a large fraction of the unsolved TSPLIB instances have now been solved. In the 2000-study 13 TSPLIB instances were unsolved for the *Euclidean* metric; among these FST generation was successful for 11 instances, and FST concatenation successful for 8 instances in the current study. Among the TSPLIB instances in the 2000-study, `fl3795` and `pla7397` are the only two instances for which Euclidean FST generation did not complete.

For the *rectilinear* metric, FST generation was trivial for all TSPLIB instances in the 2000-study. FST concatenation left 7 unsolved problems in he 2000-study; among these 4 instances have been solved (`pcb1173`, `pcb3038`, `rl5934`, `pla7397`), while 3 instances remain unsolved using GeoSteiner (`fl1400`, `fl3795`, `fnl4461`); however, it should be noted that these 3 instances have previously been solved by Polzin and Vahdati Daneshmand [20] by reducing the FST concatenation problem to

Table 6 Euclidean metric. OR-Library instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
500 (1)	1321	370	213	286	2.74	3.42	0.000	1	7	6.43	1.11	0.04	7.58
500 (2)	1312	394	181	285	2.75	3.51	0.000	1	13	7.07	0.90	0.10	8.07
500 (3)	1321	400	187	288	2.73	3.37	0.000	1	4	6.44	1.00	0.04	7.48
500 (4)	1241	373	201	291	2.71	3.50	0.000	1	4	6.82	0.89	0.05	7.76
500 (5)	1182	356	238	300	2.66	2.87	0.000	1	7	4.91	0.70	0.06	5.67
500 (6)	1308	385	201	287	2.74	3.37	0.000	1	4	7.69	1.01	0.04	8.74
500 (7)	1231	382	199	290	2.72	3.38	0.000	1	6	5.77	0.81	0.06	6.64
500 (8)	1212	369	211	304	2.64	3.17	0.000	1	5	5.78	0.76	0.04	6.58
500 (9)	1279	383	197	294	2.70	3.38	0.000	1	5	6.11	0.89	0.04	7.04
500 (10)	1325	379	182	284	2.76	3.60	0.000	1	4	6.08	1.06	0.05	7.19
500 (11)	1277	383	208	297	2.68	3.25	0.000	1	2	8.25	0.86	0.03	9.14
500 (12)	1218	374	203	302	2.65	3.21	0.000	1	4	6.46	0.76	0.03	7.25
500 (13)	1172	348	232	288	2.73	3.37	0.000	1	4	4.76	0.70	0.04	5.50
500 (14)	1395	414	160	285	2.75	3.27	0.000	1	33	7.83	1.11	0.18	9.12
500 (15)	1268	421	175	294	2.70	3.22	0.000	1	5	6.70	0.95	0.06	7.71

Table 6 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000 (1)	2435	721	402	580	2.72	3.45	0.000	1	3	14.06	1.98	0.08	16.12
1000 (2)	2533	767	397	587	2.70	3.40	0.000	1	7	13.54	2.07	0.11	15.72
1000 (3)	2424	759	450	609	2.64	3.17	0.000	1	5	17.26	1.96	0.11	19.33
1000 (4)	2676	762	374	572	2.75	3.30	0.000	1	4	14.75	2.75	0.11	17.61
1000 (5)	2413	729	458	604	2.65	3.10	0.000	1	3	14.17	1.81	0.07	16.05
1000 (6)	2525	774	394	582	2.72	3.23	0.000	1	5	14.86	2.19	0.11	17.16
1000 (7)	2454	738	416	587	2.70	3.26	0.000	1	6	14.51	2.11	0.13	16.75
1000 (8)	2533	773	376	586	2.70	3.42	0.000	1	12	15.20	2.07	0.18	17.45
1000 (9)	2603	777	368	576	2.73	3.37	0.000	1	5	16.68	2.36	0.12	19.16
1000 (10)	2472	723	424	598	2.67	3.36	0.000	1	5	13.27	1.78	0.09	15.14
1000 (11)	2532	763	395	583	2.71	3.14	0.000	1	4	14.44	2.22	0.07	16.73
1000 (12)	2616	774	382	570	2.75	3.58	0.000	1	6	16.18	2.41	0.13	18.72
1000 (13)	2488	717	456	591	2.69	3.19	0.000	1	4	14.20	2.02	0.07	16.29
1000 (14)	2522	804	366	587	2.70	3.48	0.000	1	5	15.07	2.16	0.12	17.35
1000 (15)	2531	736	400	576	2.73	3.24	0.000	1	4	15.63	2.15	0.07	17.85
10,000 (1)	25,125	7436	4125	5905	2.69	3.29	0.000	1	134	416.73	106.94	31.99	555.66

Table 7 Rectilinear metric. OR-Library instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
500 (1)	1926	626	67	256	2.95	11.52	0.000	1	30	0.01	5.49	0.43	5.93
500 (2)	2248	722	54	245	3.04	12.80	0.000	1	16	0.02	7.01	0.28	7.31
500 (3)	2155	681	50	248	3.01	12.40	0.000	1	50	0.02	5.83	0.50	6.35
500 (4)	1891	693	62	267	2.87	11.26	0.000	1	15	0.01	4.32	0.23	4.56
500 (5)	1868	583	67	257	2.94	11.08	0.000	1	41	0.01	4.04	7.54	11.59
500 (6)	2069	682	58	262	2.90	11.69	0.000	1	14	0.01	5.62	0.30	5.93
500 (7)	1953	604	81	248	3.01	11.74	0.000	1	15	0.01	4.46	0.23	4.70
500 (8)	2025	721	55	257	2.94	11.50	0.000	1	38	0.02	5.07	1.15	6.24
500 (9)	2010	668	49	263	2.90	11.15	0.000	1	41	0.01	5.21	4.71	9.93
500 (10)	1951	619	68	258	2.93	11.53	0.000	1	12	0.01	4.42	0.16	4.59
500 (11)	2065	652	68	257	2.94	11.67	0.000	1	20	0.01	5.76	0.33	6.10
500 (12)	1912	698	57	260	2.92	11.21	0.000	1	17	0.02	5.91	0.34	6.27
500 (13)	1879	657	62	259	2.93	11.66	0.000	1	14	0.02	4.11	0.20	4.33
500 (14)	2122	766	40	248	3.01	12.02	0.000	1	37	0.02	5.67	0.69	6.38
500 (15)	1953	685	63	261	2.91	11.22	0.000	1	22	0.02	5.03	0.40	5.45

Table 7 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000 (1)	4164	1478	83	520	2.92	11.84	0.000	1	57	0.04	15.93	2.63	18.60
1000 (2)	4009	1335	110	499	3.00	11.43	0.000	1	29	0.04	12.53	4.74	17.31
1000 (3)	4020	1414	111	521	2.92	11.16	0.000	1	12	0.04	14.80	0.59	15.43
1000 (4)	4090	1446	92	503	2.99	11.61	0.000	1	28	0.04	14.18	1.16	15.38
1000 (5)	4013	1388	128	524	2.91	11.34	0.000	1	75	0.04	15.68	23.36	39.08
1000 (6)	4324	1475	115	507	2.97	11.57	0.000	1	107	0.04	17.98	35.65	53.67
1000 (7)	3996	1325	110	524	2.91	11.33	0.000	1	26	0.04	12.98	0.83	13.85
1000 (8)	4290	1461	93	521	2.92	11.80	0.000	1	74	0.04	17.32	11.49	28.85
1000 (9)	4479	1499	99	504	2.98	12.10	0.000	1	26	0.05	19.27	1.25	20.57
1000 (10)	3992	1280	101	507	2.97	11.81	0.000	1	109	0.04	12.44	435.76	448.24
1000 (11)	4020	1426	100	515	2.94	11.36	0.000	1	54	0.04	15.06	2.73	17.83
1000 (12)	4687	1579	78	493	3.03	12.71	0.001	1	27	0.05	26.72	6.90	33.67
1000 (13)	3873	1274	132	521	2.92	11.43	0.000	1	18	0.04	13.19	0.57	13.80
1000 (14)	4324	1478	95	519	2.92	11.74	0.000	1	159	0.05	17.73	71.14	88.92
1000 (15)	4145	1437	105	514	2.94	11.58	0.000	1	39	0.05	15.59	2.75	18.39
10,000 (1)	40,878	13,906	935	5174	2.93	11.63	0.002	19	293	2.10	1183.42	738.66	1924.18

Table 8 Hexagonal metric. OR-Library instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
500 (1)	1280	639	113	304	2.64	4.71	0.000	1	5	28.55	109.91	0.07	138.53
500 (2)	1403	744	101	299	2.67	4.40	0.000	1	6	34.01	128.62	0.11	162.74
500 (3)	1364	700	102	295	2.69	4.56	0.000	1	5	25.03	101.54	0.08	126.65
500 (4)	1364	658	130	311	2.60	5.01	0.000	1	11	30.99	147.93	0.10	179.02
500 (5)	1218	656	117	309	2.61	4.12	0.000	1	8	20.44	58.74	0.08	79.26
500 (6)	1303	652	121	307	2.63	4.33	0.000	1	71	34.11	154.02	0.43	188.56
500 (7)	1292	703	92	320	2.56	4.74	0.000	1	6	23.76	92.77	0.09	116.62
500 (8)	1383	691	100	313	2.59	4.24	0.000	1	7	26.36	181.59	0.08	208.03
500 (9)	1254	650	129	311	2.60	4.99	0.000	1	6	32.65	94.35	0.08	127.08
500 (10)	1316	677	88	305	2.64	4.39	0.000	1	4	38.03	151.32	0.08	189.43
500 (11)	1353	680	93	309	2.61	4.69	0.000	1	9	26.81	191.68	0.13	218.62
500 (12)	1201	683	99	324	2.54	4.12	0.000	1	11	18.92	98.35	0.12	117.39
500 (13)	1209	672	91	306	2.63	4.48	0.000	1	5	18.44	50.69	0.09	69.22
500 (14)	1432	708	81	288	2.73	4.43	0.000	1	24	35.82	154.47	0.16	190.45
500 (15)	1420	697	103	312	2.60	4.51	0.000	1	6	31.84	183.58	0.11	215.53

Table 8 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000 (1)	2799	1377	200	588	2.70	4.58	0.000	1	12	161.22	410.31	0.28	571.81
1000 (2)	2760	1349	225	601	2.66	4.72	0.000	1	16	169.85	276.52	0.32	446.69
1000 (3)	2456	1346	240	635	2.57	3.75	0.000	1	33	127.66	303.06	0.48	431.20
1000 (4)	2691	1365	186	590	2.69	4.33	0.000	1	16	125.46	257.62	0.36	383.44
1000 (5)	2597	1390	195	612	2.63	4.43	0.000	1	17	121.39	266.45	0.36	388.20
1000 (6)	2806	1464	200	612	2.63	4.45	0.000	1	12	195.17	385.22	0.36	580.75
1000 (7)	2425	1265	259	641	2.56	4.26	0.000	1	7	99.50	241.33	0.23	341.06
1000 (8)	2844	1427	169	600	2.66	4.81	0.000	1	3106	193.73	495.99	86.65	776.37
1000 (9)	3041	1494	172	604	2.65	4.54	0.000	1	22	219.68	685.77	0.46	905.91
1000 (10)	2776	1400	187	602	2.66	4.55	0.000	1	36	101.18	293.27	0.63	395.08
1000 (11)	2706	1326	210	632	2.58	4.49	0.000	1	7	151.22	484.06	0.20	635.48
1000 (12)	2832	1421	201	608	2.64	5.02	0.000	1	11	172.82	328.58	0.35	501.75
1000 (13)	2497	1270	215	617	2.62	4.20	0.000	1	10	116.71	251.36	0.23	368.30
1000 (14)	2861	1402	192	616	2.62	4.65	0.000	1	10	211.47	392.57	0.30	604.34
1000 (15)	2725	1333	190	620	2.61	4.75	0.000	1	12	207.58	434.25	0.31	642.14
10,000 (1)	26,249	13,785	1965	6174	2.62	4.52	0.000	1	59	27,936.87	5149.89	33.92	33,120.68

Table 9 Octilinear metric. OR-Library instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
500 (1)	1602	425	185	283	2.76	4.67	0.000	1	4	22.06	143.24	0.06	165.36
500 (2)	1657	490	139	281	2.78	5.15	0.000	1	12	27.41	93.65	0.10	121.16
500 (3)	1666	511	116	284	2.76	4.84	0.000	1	6	25.39	93.46	0.07	118.92
500 (4)	1548	465	157	294	2.70	5.08	0.000	1	4	23.86	105.96	0.06	129.88
500 (5)	1391	469	166	302	2.65	4.17	0.000	1	7	18.96	55.68	0.06	74.70
500 (6)	1547	446	161	294	2.70	4.62	0.000	1	11	23.10	128.59	0.08	151.77
500 (7)	1450	462	158	296	2.69	4.62	0.000	1	4	19.98	71.90	0.04	91.92
500 (8)	1418	441	160	301	2.66	4.56	0.000	1	6	21.05	72.98	0.06	94.09
500 (9)	1426	447	166	300	2.66	4.48	0.000	1	56	21.59	72.66	0.35	94.60
500 (10)	1472	451	155	293	2.70	4.72	0.000	1	8	21.22	82.42	0.07	103.71
500 (11)	1535	463	141	300	2.66	4.56	0.000	1	7	25.51	139.05	0.06	164.62
500 (12)	1328	420	181	301	2.66	4.42	0.000	1	4	19.70	73.55	0.04	93.29
500 (13)	1430	422	191	295	2.69	4.66	0.000	1	15	18.61	55.73	0.16	74.50
500 (14)	1623	501	128	288	2.73	4.59	0.000	1	5	25.61	115.57	0.06	141.24
500 (15)	1549	493	143	300	2.66	4.63	0.000	1	44	21.60	111.09	9.69	142.38

Table 9 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
1000 (1)	2832	882	340	615	2.62	4.66	0.000	1	7	99.67	225.55	0.29	325.51
1000 (2)	2973	954	286	586	2.70	4.81	0.000	1	9	112.00	148.82	0.19	261.01
1000 (3)	2961	952	324	605	2.65	4.32	0.000	1	18	109.03	357.76	0.24	467.03
1000 (4)	3141	934	297	576	2.73	4.82	0.000	1	7	117.16	200.84	0.13	318.13
1000 (5)	2962	886	368	608	2.64	4.31	0.000	1	8	107.23	212.93	0.14	320.30
1000 (6)	3104	914	314	588	2.70	4.65	0.000	1	30	111.17	181.13	0.36	292.66
1000 (7)	2869	902	305	580	2.72	4.55	0.000	1	21	98.74	222.80	0.58	322.12
1000 (8)	3079	923	279	576	2.73	4.89	0.000	1	122	121.31	219.21	540.03	880.55
1000 (9)	3048	965	288	585	2.71	4.64	0.000	1	19	118.86	235.96	0.35	355.17
1000 (10)	2954	898	315	598	2.67	4.74	0.000	1	8	104.18	183.66	0.17	288.01
1000 (11)	3029	942	273	587	2.70	4.54	0.000	1	6	113.05	211.09	0.15	324.29
1000 (12)	3321	920	312	571	2.75	4.94	0.000	1	8	131.34	260.39	0.17	391.90
1000 (13)	2826	849	356	583	2.71	4.43	0.000	1	5	93.60	238.36	0.10	332.06
1000 (14)	3257	948	292	586	2.70	4.88	0.000	1	9	130.79	245.74	0.16	376.69
1000 (15)	2988	916	290	574	2.74	4.56	0.000	1	5	110.72	212.12	0.15	322.99
10,000 (1)	29,855	9091	3058	5886	2.70	4.58	0.000	1	134	25,932.21	3441.30	30.10	29,403.61

Table 10 Euclidean metric. TSPLIB instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
d198	852	264	54	103	2.91	2.91	0.000	1	5	21.92	1.92	0.03	23.87
lin318	1229	442	42	208	2.52	4.77	0.000	1	7	15.21	1.72	0.07	17.00
fl417	3656	2207	94	180	3.31	3.34	0.000	1	6	209.87	55.44	0.25	265.56
pcb442	8690	1771	126	261	2.69	4.08	0.042	13	108	38,917.65	1499.18	2.54	40,419.37
att532	1382	404	217	310	2.71	3.36	0.000	1	19	13.64	1.04	0.14	14.82
ali535	1271	416	262	338	2.58	2.81	0.000	1	11	19.00	1.14	0.06	20.20
u574	1426	450	230	344	2.67	3.15	0.000	1	10	13.03	1.10	0.09	14.22
rat575	1651	503	170	323	2.78	3.62	0.000	1	9	15.97	1.51	0.14	17.62
p654	29,943	14,458	120	–	–	–	–	–	–	882,034.39	21,822.87	–	–
d657	1654	494	277	390	2.68	3.00	0.000	1	6	52.12	1.83	0.07	54.02
gr666	1763	497	261	378	2.76	3.15	0.000	1	8	56.49	1.79	0.09	58.37
u724	1926	515	304	409	2.77	3.53	0.000	1	4	13.23	1.59	0.05	14.87
rat783	2153	658	232	448	2.75	3.52	0.000	1	7	19.32	1.95	0.12	21.39
dsj1000	2457	719	466	596	2.68	2.88	0.000	1	11	71.97	3.09	0.14	75.20
pr1002	2512	721	448	579	2.73	3.05	0.000	1	4	46.74	2.69	0.09	49.52
u1060	3690	1470	431	597	2.77	3.25	0.005	37	255	110.76	16.41	19.31	146.48
vm1084	2244	915	633	786	2.38	2.75	0.000	1	5	10.76	1.47	0.09	12.32
pcb1173	2941	994	613	802	2.46	3.18	0.000	1	11	43.66	5.04	0.21	48.91
d1291	33,100	3204	82	996	2.30	2.78	0.027	49	1040	353,162.66	3092.52	177.48	356,432.66
rl1304	2529	1204	869	1058	2.23	1.68	0.000	1	6	15.86	2.10	0.15	18.11
rl1323	2500	1204	883	1076	2.23	1.65	0.000	1	4	23.09	1.91	0.08	25.08
nrw1379	4105	1255	345	752	2.83	4.16	0.000	1	9	43.83	5.88	0.39	50.10
fl1400	49,916	35,847	238	–	–	–	–	–	–	239,677.98	32,854.93	–	–

Table 10 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
u1432	30,549	16,316	80	–	–	–	–	–	–	133,924.47	28,046.71	–	–
fl1577	19,644	8981	384	924	2.71	2.82	0.000	1	37	848,027.36	36,532.47	17.35	884,577.18
d1655	13,835	5003	514	1078	2.53	3.41	0.019	273	44,894	75,914.23	2236.44	5809.52	83,960.19
vm1748	3643	1527	943	1258	2.39	2.81	0.000	1	6	23.10	3.43	0.23	26.76
u1817	53,911	7344	61	1146	2.58	3.70	0.002	15	106	477,437.54	13,437.49	109.59	490,984.62
rl1889	3651	1683	1248	1485	2.27	2.02	0.000	1	4	29.53	4.06	0.19	33.78
d2103	–	–	–	–	–	–	–	–	–	–	–	–	–
u2152	39,603	11,027	149	1298	2.66	3.85	0.005	14	271	44,102.01	25,832.77	59.50	69,994.28
u2319	–	–	–	–	–	–	–	–	–	–	–	–	–
pr2392	6373	2016	987	1476	2.62	3.61	0.000	1	6	67.20	10.56	0.40	78.16
pcb3038	11,636	4376	914	1879	2.62	3.69	0.004	13	163	720.95	97.82	85.85	904.62
fl3795	–	–	–	–	–	–	–	–	–	–	–	–	–
fnl4461	13,323	4068	1091	2457	2.82	4.13	0.000	1	159	301.81	46.13	44.95	392.89
rl5915	18,995	7034	3479	4979	2.19	2.22	0.000	1	25	3025.80	136.86	4.57	3167.23
rl5934	18,163	7239	3129	4825	2.23	2.13	0.001	3	232	1152.83	146.80	661.40	1961.03
pla7397	–	–	–	–	–	–	–	–	–	–	–	–	–
rl11849	46,742	16,542	5440	–	–	–	–	–	–	34,250.04	2835.90	–	–
usa13509	33,327	10,810	5586	8287	2.63	3.29	0.000	1	7421	3539.85	240.38	691,445.94	695,226.17
brd14051	40,831	12,305	3986	7864	2.79	4.00	0.000	1	467	36,234.57	665.05	3660.68	40,560.30
d15112	43,467	12,714	4465	–	–	–	–	–	–	6740.17	483.48	–	–
d18512	54,211	16,447	5028	10,323	2.79	4.04	0.000	1	1336	12,045.25	834.17	122,217.99	135,097.41
pla33810	–	–	–	–	–	–	–	–	–	–	–	–	–
pla85900	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 11 Rectilinear metric. TSPLIB instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
d198	265	200	143	173	2.14	3.66	0.000	1	3	0.00	0.06	0.00	0.06
lin318	963	492	53	227	2.40	8.90	0.000	1	60	0.01	1.49	2694.71	2696.21
fl417	1178	698	127	238	2.75	11.94	0.000	1	75	0.01	1.75	62.01	63.77
pcb442	558	422	353	394	2.12	3.99	0.000	1	3	0.01	0.11	0.03	0.15
att532	2230	751	78	273	2.95	11.44	0.000	1	222	0.01	6.97	83.95	90.93
ali535	1788	575	157	308	2.73	9.77	0.000	1	47	0.01	3.67	0.81	4.49
u574	1470	586	176	375	2.53	8.94	0.000	1	10	0.01	1.61	0.13	1.75
rat575	3326	1217	17	282	3.04	13.11	0.000	1	16	0.03	13.43	0.72	14.18
p654	930	668	504	586	2.11	5.89	0.000	1	4	0.03	0.40	0.05	0.48
d657	2259	841	121	379	2.73	10.57	0.000	1	118	0.02	4.78	4.61	9.41
gr666	2804	930	95	349	2.91	11.03	0.000	1	210	0.02	10.06	47.52	57.60
u724	1658	813	250	506	2.43	9.63	0.000	1	29	0.02	1.57	0.50	2.09
rat783	3870	1432	39	418	2.87	12.65	0.000	1	61	0.04	15.85	1.51	17.40
dsj1000	4086	1254	136	503	2.99	11.09	0.000	1	25	0.04	15.13	1.10	16.27
pr1002	2147	922	459	702	2.43	8.63	0.000	1	11	0.04	1.81	0.86	2.71
u1060	2765	1236	327	675	2.57	11.35	0.000	1	145	0.04	4.48	774.91	779.43
vm1084	2313	1137	612	838	2.29	8.35	0.000	1	15	0.05	2.69	0.34	3.08
pcb1173	2886	941	408	688	2.70	6.20	0.000	1	12	0.04	3.58	0.40	4.02
d1291	1379	1334	1164	1250	2.03	1.80	0.000	1	4	0.06	0.41	0.06	0.53
rl1304	1926	1254	1007	1141	2.14	5.04	0.000	1	7	0.05	1.07	0.14	1.26
rl1323	1899	1287	956	1157	2.14	5.45	0.000	1	7	0.05	1.12	0.13	1.30
nrw1379	8202	2864	55	661	3.08	13.02	0.000	1	84	0.09	90.45	25.81	116.35
fl1400	5830	3667	166	–	–	–	–	–	–	0.10	19.90	–	–

Table 11 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
u1432	1431	1431	1431	1431	2.00	0.00	0.000	1	1	0.06	0.21	0.03	0.30
fl1577	3820	2416	601	1191	2.32	10.59	0.000	1	11	0.08	5.82	1.29	7.19
d1655	2129	1677	1314	1508	2.10	3.57	0.000	1	16	0.07	1.14	0.32	1.53
vm1748	3912	1940	945	1328	2.32	8.93	0.000	1	30	0.11	6.88	3.15	10.14
u1817	1839	1820	1787	1805	2.01	0.36	0.000	1	3	0.10	0.37	0.05	0.52
rl1889	2867	1836	1355	1612	2.17	5.49	0.000	1	6	0.09	2.18	0.31	2.58
d2103	2240	2104	1964	2046	2.03	0.61	0.000	1	3	0.14	0.89	0.08	1.11
u2152	2171	2160	2123	2142	2.00	0.22	0.000	1	4	0.13	0.48	0.06	0.67
u2319	2318	2318	2318	2318	2.00	0.00	0.000	1	1	0.17	0.54	0.02	0.73
pr2392	4443	2357	1204	1738	2.38	7.75	0.000	1	14	0.13	4.11	1.01	5.25
pcb3038	9048	3138	706	1704	2.78	6.20	0.000	1	4764	0.19	30.73	16,448.87	16,479.79
fl3795	7768	5986	2727	–	–	–	–	–	–	0.41	17.96	–	–
fnl4461	27,818	9473	163	–	–	–	–	–	–	0.54	762.73	–	–
rl5915	7462	5841	4989	5425	2.09	3.46	0.000	1	21	0.72	7.49	4.55	12.76
rl5934	8052	5898	4762	5353	2.11	3.15	0.000	1	24	0.70	9.65	2.85	13.20
pla7397	10,452	7799	5608	6779	2.09	3.88	0.000	1	99	1.35	19.52	11.61	32.48
rl11849	16,582	11,758	9069	10,432	2.14	3.86	0.000	1	321	2.76	49.10	143.68	195.54
usa13509	53,025	18,585	1804	–	–	–	–	–	–	2.94	1969.08	–	–
brd14051	77,236	27,581	700	–	–	–	–	–	–	3.65	7491.68	–	–
d15112	80,581	28,224	784	–	–	–	–	–	–	4.84	7039.53	–	–
d18512	105,887	37,497	823	–	–	–	–	–	–	6.14	15,157.44	–	–
pla33810	47,821	35,328	20,269	27,432	2.23	2.07	0.000	3	4420	22.10	404.54	5644.07	6070.71
pla85900	130,469	100,238	56,143	–	–	–	–	–	–	156.53	7869.10	–	–

Table 12 Hexagonal metric. TSPLIB instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
d198	544	419	26	133	2.48	3.58	0.000	1	55	1.38	50.51	1.08	52.97
lin318	1415	547	58	214	2.48	4.61	0.000	1	7	61.00	253.12	0.11	314.23
fl417	1734	1067	35	201	3.07	4.94	0.000	1	89	74.45	574.01	1.31	649.77
pcb442	5520	1227	122	296	2.49	5.35	0.084	11	36	850.51	15,382.63	0.88	16,234.02
att532	1521	699	111	327	2.62	4.83	0.000	1	62	34.79	257.00	0.72	292.51
ali535	1437	773	122	366	2.46	3.68	0.000	1	85	72.68	383.60	0.69	456.97
u574	1338	843	125	368	2.56	3.40	0.000	1	15	13.01	165.38	0.17	178.56
rat575	2143	1034	60	323	2.78	5.27	0.000	1	5	173.76	323.16	0.15	497.07
p654	–	–	–	–	–	–	–	–	–	–	–	–	–
d657	2002	1049	106	391	2.68	4.35	0.000	1	26	52.23	790.81	0.40	843.44
gr666	2054	959	106	401	2.66	4.55	0.000	1	20	54.66	770.20	0.50	825.36
u724	1717	1205	120	484	2.49	3.44	0.000	1	11	32.39	117.40	0.23	150.02
rat783	2593	1323	91	460	2.70	4.95	0.000	1	22	165.36	451.75	0.45	617.56
dsj1000	2750	1338	243	610	2.64	3.97	0.000	1	16	130.02	1440.26	0.42	1570.70
pr1002	2695	1506	193	645	2.55	4.28	0.000	1	30	58.51	1039.10	0.57	1098.18
u1060	3022	1790	196	–	–	–	–	–	–	107.22	1402.49	–	–
vm1084	1933	1496	411	845	2.28	2.52	0.000	1	6	26.65	98.04	0.25	124.94
pcb1173	3043	1555	400	787	2.49	4.72	0.000	1	8	279.14	2751.92	0.29	3031.35
d1291	5425	3986	38	1034	2.25	3.65	0.000	1	21	837.83	4527.74	1.69	5367.26
rl1304	1905	1750	770	1165	2.12	0.73	0.000	1	18	13.77	112.27	0.55	126.59
rl1323	2053	1763	714	1154	2.15	1.44	0.000	1	7	19.70	225.99	0.31	246.00
nrw1379	6051	2740	93	756	2.82	5.89	0.000	1	174	3368.13	2971.76	19.13	6359.02
fl1400	15,416	7589	74	–	–	–	–	–	–	17,090.64	35,574.42	–	–

Table 12 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
u1432	–	–	–	–	–	–	–	–	–	–	–	–	–
fl1577	7229	5509	324	1228	2.28	3.42	0.000	1	11	17,501.08	26,926.86	3.91	44,431.85
d1655	4232	2882	494	1235	2.34	4.71	0.031	43	17,616	92.42	3462.87	13,426.57	16,981.86
vm1748	3301	2412	709	1353	2.29	2.96	0.000	1	29	105.07	381.21	1.43	487.71
u1817	11,066	5393	61	1379	2.32	3.68	0.003	3	12,853	12,711.96	68,964.49	15,653.42	97,329.87
rl1889	2847	2415	1005	1598	2.18	1.71	0.000	1	9	33.54	130.51	0.63	164.68
d2103	7312	5381	66	1985	2.06	0.95	0.000	1	8	326.33	6170.10	1.45	6497.88
u2152	12,094	5861	212	1578	2.36	4.23	0.000	1	324	36,195.36	66,704.89	26.97	102,927.22
u2319	–	–	–	–	–	–	–	–	–	–	–	–	–
pr2392	6313	3676	521	1519	2.57	4.99	0.000	1	23	189.22	742.03	2.56	933.81
pcb3038	12,553	5529	470	–	–	–	–	–	–	48,677.24	44,133.20	–	–
fl3795	–	–	–	–	–	–	–	–	–	–	–	–	–
fnl4461	17,124	8029	389	–	–	–	–	–	–	17,103.32	12,753.33	–	–
rl5915	10,090	8679	3418	5337	2.11	1.42	0.000	1	29	620.76	1412.03	5.38	2038.17
rl5934	10,154	8767	3083	5256	2.13	1.37	0.000	1	15	709.07	2469.14	5.13	3183.34
pla7397	–	–	–	–	–	–	–	–	–	–	–	–	–
rl11849	–	–	–	–	–	–	–	–	–	–	–	–	–
usa13509	–	–	–	–	–	–	–	–	–	–	–	–	–
brd14051	–	–	–	–	–	–	–	–	–	–	–	–	–
d15112	–	–	–	–	–	–	–	–	–	–	–	–	–
d18512	–	–	–	–	–	–	–	–	–	–	–	–	–
pla33810	–	–	–	–	–	–	–	–	–	–	–	–	–
pla85900	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 13 Octilinear metric. TSPLIB instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
d198	277	239	90	153	2.29	1.67	0.000	1	4	0.28	12.78	0.02	13.08
lin318	1320	532	35	210	2.51	5.83	0.000	1	73	7.82	152.74	0.80	161.36
fl417	1357	653	125	217	2.92	2.97	0.000	1	8	6.69	256.11	0.22	263.02
pcb442	680	529	228	318	2.39	2.88	0.035	5	33	1.26	31.88	0.69	33.83
att532	1706	496	158	313	2.70	4.68	0.000	1	11	21.43	258.74	0.09	280.26
ali535	1390	493	228	359	2.49	4.05	0.000	1	209	16.61	288.51	1.74	306.86
u574	1298	536	237	377	2.52	3.82	0.000	1	7	10.96	140.89	0.16	152.01
rat575	1784	670	117	343	2.67	4.97	0.000	1	15	28.73	79.14	1.14	109.01
p654	1466	1014	136	328	2.99	3.41	0.050	419	2685	5.29	222.32	3204.50	3432.11
d657	1575	660	254	431	2.52	3.92	0.000	1	11	17.78	454.49	0.11	472.38
gr666	2138	626	207	382	2.74	4.62	0.000	1	18	34.35	734.36	0.19	768.90
u724	1389	723	307	513	2.41	3.48	0.000	1	6	12.63	47.23	0.17	60.03
rat783	2331	843	178	491	2.59	4.98	0.000	1	6	44.93	146.73	0.12	191.78
dsj1000	2887	824	394	583	2.71	4.16	0.000	1	5	97.30	1247.34	0.14	1344.78
pr1002	1967	998	451	688	2.45	3.40	0.000	1	4	20.83	375.04	0.18	396.05
u1060	3179	1254	466	693	2.53	3.26	0.000	1	1292	50.00	1786.55	91,427.58	93,264.13
vm1084	1933	1045	674	838	2.29	3.41	0.000	1	12	28.25	85.37	0.27	113.89
pcb1173	2586	1119	551	798	2.47	3.15	0.000	1	7	51.11	740.75	0.17	792.03
d1291	2776	2097	233	–	–	–	–	–	–	18.07	1282.39	–	–
rl1304	1869	1339	906	1123	2.16	1.93	0.000	1	6	16.91	151.40	0.16	168.47
rl1323	1775	1324	948	1151	2.15	1.83	0.000	1	4	17.82	159.96	0.15	177.93
nrw1379	5209	1577	266	786	2.75	5.52	0.000	1	59	289.63	688.16	4.67	982.46
fl1400	16,860	6697	184	–	–	–	–	–	–	752.03	34,119.33	–	–

Table 13 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
u1432	2402	2029	342	863	2.66	3.34	0.043	661	8178	8.55	58.83	1423.10	1490.48
fl1577	4356	2635	494	1209	2.30	1.62	0.000	1	102	71.09	3847.04	6.15	3924.28
d1655	2749	2195	664	1244	2.33	2.45	0.019	83	1403	23.49	1600.79	873.80	2498.08
vm1748	3306	1695	1019	1335	2.31	3.51	0.000	1	8	92.25	264.12	0.79	357.16
u1817	3715	2938	161	–	–	–	–	–	–	38.69	680.79	–	–
rl1889	2789	1892	1303	1610	2.17	2.29	0.000	1	8	48.84	232.01	0.43	281.28
d2103	4447	3474	177	1919	2.10	0.83	0.000	1	13	55.68	3794.06	1.48	3851.22
u2152	3971	3375	260	1548	2.39	2.22	0.000	1	166	43.68	179.66	22.80	246.14
u2319	5975	4176	92	–	–	–	–	–	–	34.58	190.78	–	–
pr2392	4866	2654	1134	1688	2.42	3.69	0.000	1	9	119.54	240.68	1.30	361.52
pcb3038	10,898	4015	844	1900	2.60	3.57	0.003	19	151	1004.02	12,879.92	46.90	13,930.84
fl3795	–	–	–	–	–	–	–	–	–	–	–	–	–
fnl4461	17,014	5184	727	2464	2.81	5.45	0.000	1	458	7400.69	4035.67	556.04	11,992.40
rl5915	9400	6924	3798	5224	2.13	1.95	0.000	1	23	635.37	1618.47	5.19	2259.03
rl5934	9902	7043	3439	5172	2.15	1.80	0.000	1	177	858.25	3597.12	35.64	4491.01
pla7397	17,429	11,966	1766	–	–	–	–	–	–	2456.84	55,923.99	–	–
rl11849	21,470	14,421	6298	9468	2.25	2.30	0.000	3	137	6010.50	23,773.68	257.75	30,041.93
usa13509	39,254	12,801	4533	–	–	–	–	–	–	41,525.14	56,671.60	–	–
brd14051	–	–	–	–	–	–	–	–	–	–	–	–	–
d15112	–	–	–	–	–	–	–	–	–	–	–	–	–
d18512	–	–	–	–	–	–	–	–	–	–	–	–	–
pla33810	–	–	–	–	–	–	–	–	–	–	–	–	–
pla85900	–	–	–	–	–	–	–	–	–	–	–	–	–

a Steiner tree problem in a graph (rather than to a minimum spanning tree problem in a hypergraph).

4.3.4 Large-scale randomly generated instances for the Euclidean metric

It appears from the experimental results on the randomly generated instances that the Euclidean metric is significantly "easier" than the other metrics. In order to test the practical limit of GeoSteiner for the Euclidean metric, we ran GeoSteiner on a series of 45 large-scale randomly generated instances with up to 100,000 terminals. We allowed up to 7 days of computing time for each of the three phases (FST generation, FST pruning and FST concatenation) for these problem instances. The 100k instances provoked a bug that turned out to be caused by use of a 16-bit type. This type was replaced with a 32-bit type for the 100k instances. It is unknown whether this change would produce a measurable difference if all of the other experiments were re-run using the 32-bit type. We think measurable differences would be unlikely.

Table 14 shows the results of our experiments on the large-scale instances. All 15 instances with 25k terminals, 8 out of 15 instances with 50k terminals and 3 out of 15 instances with 100k terminals have been solved. FST generation and FST pruning completed for all 45 instances, and the running time was quite stable and predictable. FST generation took less than one hour for 25k terminals and less than one day for 100k terminals.

The bottleneck for the large-scale instances is the FST concatenation problem. The branch-and-cut algorithm basically spends all its time solving the LP-relaxation problem in the root node. Figure 5 shows the UB/LB-gap traces for each of the 45 instances. It follows that the algorithm very quickly obtains a gap of less than 0.01%, and then stays at a gap of around 0.001% before suddenly dropping down just before the instance is solved. All the unsolved 50k and 100k instances have a gap that is less than 0.001% when the algorithm times out.

4.4 Performance across code bases and configurations

The effect of the algorithmic enhancements is illustrated by comparing the branch-and-bound code on three different code bases: the code used in the 2000-study (GeoSteiner 2000), the publicly available GeoSteiner 3.1 code and the GeoSteiner 4.0 code. Also, we show the effect of using FST pruning in all three cases.

The tests are made on the 15 *estein* instances with 1000 terminals. These problem instances were difficult to solve under the rectilinear metric using GeoSteiner 2000, and therefore they nicely illustrate the improvements made. The average running time in the 2000-study on these 15 instances was around 2,000 seconds for the Euclidean metric and more than 100,000 seconds for the rectilinear metric. We estimate that the combined effects of platform improvements (including hardware, compiler, operating system, and CPLEX) to be about 15–20 times faster than the platform used in the 2000-study.

Table 15 presents the results of running the old and new code on the 15 estein instances (on the *same* modern machine). These results serve two purposes: (1) demon-

Table 14 Euclidean metric. Large-scale randomly generated instances

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
25k (1)	63,292	18,708	10,144	14,674	2.70	3.29	0.000	1	205	2379.09	800.68	341.45	3521.22
25k (2)	63,298	18,877	9957	14,746	2.70	3.30	0.000	1	77	2406.77	727.75	121.12	3255.64
25k (3)	62,700	19,076	9670	14,681	2.70	3.26	0.000	1	120	2328.46	783.39	196.88	3308.73
25k (4)	63,469	19,020	9797	14,730	2.70	3.31	0.000	1	126	2401.79	749.28	202.80	3353.87
25k (5)	62,993	18,597	9987	14,704	2.70	3.30	0.000	1	27	2301.07	710.96	41.09	3053.12
25k (6)	63,502	18,969	9721	14,708	2.70	3.30	0.000	1	332	2336.73	732.54	1043.54	4112.81
25k (7)	64,190	19,290	9465	14,687	2.70	3.32	0.000	3	80	2521.96	872.09	130.20	3524.25
25k (8)	63,251	18,902	9734	14,622	2.71	3.28	0.000	1	83	2473.16	788.44	127.60	3389.20
25k (9)	63,698	19,066	9660	14,620	2.71	3.30	0.000	1	199	2405.71	800.08	670.23	3876.02
25k (10)	62,592	18,838	10,055	14,732	2.70	3.25	0.000	1	83	2401.15	750.70	149.94	3301.79
25k (11)	63,081	18,885	9856	14,718	2.70	3.32	0.000	1	162	2447.10	762.18	221.50	3430.78
25k (12)	63,011	18,858	9961	14,706	2.70	3.30	0.000	1	567	2315.33	706.05	6250.64	9272.02
25k (13)	62,903	18,817	10,010	14,713	2.70	3.26	0.000	1	175	2339.86	743.58	323.11	3406.55
25k (14)	64,099	19,141	9482	14,579	2.71	3.33	0.000	1	75	2469.17	809.94	97.87	3376.98
25k (15)	63,593	18,856	9838	14,666	2.70	3.31	0.000	1	55	2415.08	726.07	85.72	3226.87
50k (1)	126,946	38,363	19,119	–	–	–	–	–	–	11,793.53	3293.71	–	–
50k (2)	125,875	37,855	19,836	29,529	2.69	3.28	0.000	1	519	11,614.54	3324.06	8965.89	23,904.49
50k (3)	126,871	37,751	19,729	29,308	2.71	3.26	0.000	1	57	11,602.68	3218.38	232.35	15,053.41
50k (4)	127,161	37,776	19,449	–	–	–	–	–	–	11,706.12	3249.14	–	–
50k (5)	126,797	38,309	19,204	–	–	–	–	–	–	11,811.63	3321.44	–	–
50k (6)	127,189	38,412	19,061	29,407	2.70	3.31	0.000	1	1012	11,769.66	3134.74	32,494.45	47,398.85
50k (7)	126,804	38,022	19,372	29,414	2.70	3.31	0.000	1	94	11,737.50	3070.84	423.30	15,231.64
50k (8)	127,503	38,061	19,443	29,355	2.70	3.30	0.000	1	295	11,803.59	3290.28	3519.58	18,613.45

Table 14 continued

Instance	FST counts			SMT properties			FST conc			CPU time			
	Gen	Prun	Req	NumF	SizeF	Red	Gap	Nds	LPs	Gen	Prun	Conc	Total
50k (9)	127,089	37,826	19,882	–	–	–	–	–	–	11,818.66	3194.24	–	–
50k (10)	126,534	37,974	19,426	29,401	2.70	3.30	0.000	1	395	11,676.56	3197.87	5771.43	20,645.86
50k (11)	12,6675	38,062	19,342	–	–	–	–	–	–	11,843.33	3087.62	–	–
50k (12)	126,432	38,118	19,571	–	–	–	–	–	–	11,657.30	3226.43	–	–
50k (13)	126,760	37,730	19,564	–	–	–	–	–	–	11,731.23	3225.79	–	–
50k (14)	126,821	38,068	19,263	29,273	2.71	3.30	0.000	1	1339	11,735.92	3544.57	123,029.24	138,309.73
50k (15)	127,274	38,241	19,338	29,351	2.70	3.30	0.000	1	1540	11,975.34	3308.85	57,238.76	72,522.95
100k (1)	253,124	75,848	39,253	–	–	–	–	–	–	77,228.56	15,317.49	–	–
100k (2)	253,233	75,935	38,878	–	–	–	–	–	–	76,102.94	14,395.28	–	–
100k (3)	253,841	75,915	38,581	–	–	–	–	–	–	79,185.89	15,610.50	–	–
100k (4)	254,068	76,236	39,178	–	–	–	–	–	–	81,504.20	15,044.78	–	–
100k (5)	255,139	76,149	38,407	–	–	–	–	–	–	58,308.26	15,875.59	–	–
100k (6)	255,071	76,015	38,823	–	–	–	–	–	–	81,790.92	13,798.18	–	–
100k (7)	253,663	75,754	38,973	–	–	–	–	–	–	57,312.35	15,495.87	–	–
100k (8)	253,699	75,524	39,237	58,645	2.71	3.30	0.000	1	215	80,427.88	16,383.35	3944.88	100,756.11
100k (9)	254,748	75,597	38,949	58,620	2.71	3.33	0.000	1	481	56,211.54	15,219.60	11,631.47	83,062.61
100k (10)	254,068	76,022	38,591	–	–	–	–	–	–	82,821.57	15,865.32	–	–
100k (11)	253,655	76,215	38,922	–	–	–	–	–	–	58,139.65	15,509.03	–	–
100k (12)	253,087	75,564	39,243	–	–	–	–	–	–	79,385.13	15,588.51	–	–
100k (13)	253,845	75,411	39,131	58,666	2.70	3.31	0.000	1	335	55,184.98	16,915.00	15,532.15	87,632.13
100k (14)	253,926	76,299	38,588	–	–	–	–	–	–	81,493.39	15,758.26	–	–
100k (15)	253,574	75,956	38,836	–	–	–	–	–	–	57,218.97	15,825.07	–	–

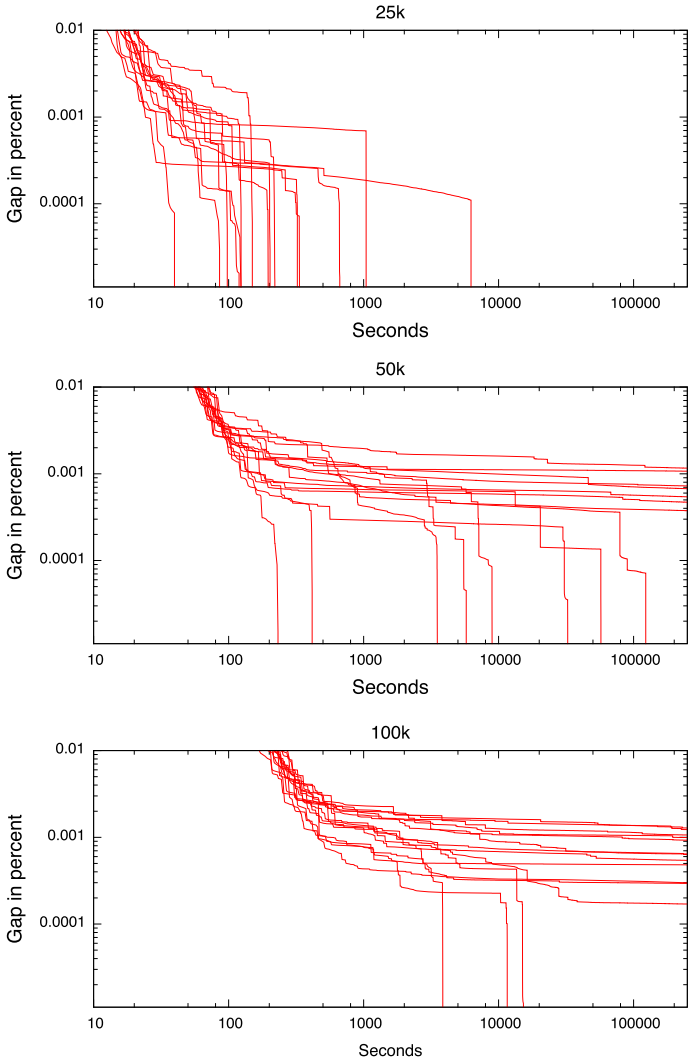

Fig. 5 Gap traces for FST concatenation on large-scale randomly generated problem instances (Euclidean metric). The gap is defined as the ratio between the current best upper bound and lower bound in percent (hence a ratio of 1.01 corresponds to a gap of 1 percent). Running time for FST concatenation on x-axis

strate the rough order of magnitude performance improvement between versions, and (2) provide a means to estimate the "platform improvements" between the 2000-study and the present study. Because of their relatively low variance in run times between instances of the same size, random instances are the best choice for both of these purposes. It follows from Table 15 that the algorithmic enhancements have resulted in two orders of magnitude speed-up for the rectilinear metric—and somewhat less for the Euclidean metric. FST pruning has a significant (positive) effect on the performance of the FST concatenation algorithm.

Table 15 Comparison of different code bases

Version	Metric	Pruning	Nds	LPs	Time
2000	Euclidean	No	1.3	620.1	322.74
3.1	Euclidean	No	1.0	480.1	209.61
4.0	Euclidean	No	1.0	41.7	11.73
2000	Euclidean	Yes	1.1	23.4	17.50
3.1	Euclidean	Yes	1.0	21.7	17.57
4.0	Euclidean	Yes	1.0	5.1	12.43
2000	Rectilinear	No	142.7	2857.5	7898.26
3.1	Rectilinear	No	4.7	1015.1	1844.88
4.0	Rectilinear	No	5.3	216.9	85.25
2000	Rectilinear	Yes	43.5	4405.5	2803.18
3.1	Rectilinear	Yes	2.9	104.1	69.26
4.0	Rectilinear	Yes	3.0	40.7	22.10

FST concatenation on the 15 estein instances with 1000 terminals. Average number of nodes, LPs and running times for FST concatenation under the Euclidean and rectilinear metric. FSTs are generated and pruned using GeoSteiner 4.0 for all FST concatenation algorithms. Local cuts were not activated in the GeoSteiner 4.0 code

5 Conclusions

In this paper we presented an updated computational study of the GeoSteiner software package. As a consequence of a number of algorithmic enhancements, the software package can now compute minimum Steiner trees for 5–10 times as many terminals when compared to the 2000-study. Highly structured and/or clustered terminal sets still remain a challenge under some metrics.

During the preparation of this paper, GeoSteiner, Inc. decided to release its proprietary software under an open source license as GeoSteiner 5.0. This version of the software is substantively identical to version 4.0: differences include the removal of license enforcement code, changes to the software configuration and build process, and other minor changes that should not affect any of the results presented in this paper.

Acknowledgements The authors would like to thank the referees for their comments that helped to improve this paper.

References

1. Althaus, E.: Berechnung optimaler Steinerbäume in der Ebene. Master's thesis, Max-Planck-Institut für Informatik in Saarbrücken, Universität des Saarlandes (1998)
2. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: TSP cuts which do not conform to the template paradigm. In: Jünger, M., Naddef, D. (eds.) Computational combinatorial optimization. Lecture Notes in Computer Science, vol. 2241. Springer, Berlin (2001)
3. Beasley, J.E.: OR-library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**, 1069–1072 (1990)
4. Brazil, M., Thomas, D.A., Weng, J.F., Zachariasen, M.: Canonical forms and algorithms for Steiner trees in uniform orientation metrics. Algorithmica **44**, 281–300 (2006)
5. Brazil, M., Zachariasen, M.: Steiner trees for fixed orientation metrics. J. Glob. Optim. **43**, 141–169 (2009)
6. Brazil, M., Zachariasen, M.: The uniform orientation Steiner tree problem is NP-hard. Int. J. Comput. Geom. **24**, 87–105 (2014)

7. Brazil, M., Zachariasen, M.: Optimal Interconnection Trees in the Plane: Theory, Algorithms and Applications. Springer, Berlin (2015)
8. Fößmeier, U., Kaufmann, M.: Solving rectilinear Steiner tree problems exactly in theory and practice. In: Burkard, R., Woeginger, G. (eds.) Algorithms ESA 97, Lecture Notes in Computer Science, vol. 1284, pp. 171–185. Springer, Berlin (1997)
9. Garey, M.R., Graham, R.L., Johnson, D.S.: The complexity of computing Steiner minimal trees. SIAM J. Appl. Math. **32**(4), 835–859 (1977)
10. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. SIAM J. Appl. Math. **32**(4), 826–834 (1977)
11. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. Appl. Math. **16**(1), 1–29 (1968)
12. Hanan, M.: On Steiner's problem with rectilinear distance. SIAM J. Appl. Math. **14**(2), 255–265 (1966)
13. Huang, T., Young, E.F.Y.: Obsteiner: an exact algorithm for the construction of rectilinear Steiner minimum trees in the presence of complex rectilinear obstacles. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(6), 882–893 (2013)
14. Hwang, F.K., Richards, D.S.: Steiner tree problems. Networks **22**, 55–89 (1992)
15. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. Annals of Discrete Mathematics, vol. 53. Elsevier, Amsterdam (1992)
16. Kahng, A.B., Mandoiu, I.I., Zelikovsky, A.Z.: Highly scalable algorithms for rectilinear and octilinear Steiner trees. In: Proceedings of the Asia and South Pacific Design Automation Conference, New York, pp. 827–833 (2003)
17. Mehlhorn, K., Näher, S.: LEDA: a library of efficient data types and algorithms. In: Proceedings of the Seventeenth International Colloquim on Automata, Languages and Programming, pp. 1–5 (1990)
18. Mehlhorn, K., Näher, S.: LEDA—A Platform for Combinatorial and Geometric Computing. Max-Planck-Institut für Informatik, Saarbrücken http://www.mpi-sb.mpg.de/LEDA/leda.html (1996)
19. Mehlhorn, K., Näher, S.: LEDA—A Platform for Combinatorial and Geometric Computing. Max-Planck-Institut für Informatik, Saarbrücken (1996). http://www.mpi-sb.mpg.de/LEDA/leda.html
20. Polzin, T., Daneshmand, S.V.: On Steiner trees and minimum spanning trees in hypergraphs. Oper. Res. Lett. **31**, 12–20 (2003)
21. Salowe, J.S., Warme, D.M.: Thirty-five-point rectilinear Steiner minimal trees in a day. Networks **25**(2), 69–87 (1995)
22. Smith, J.M., Lee, D.T., Liebman, J.S.: An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric. Networks **11**, 23–29 (1981)
23. Thomborson, C.D., Alpern, B., Carter, L.: Rectilinear Steiner tree minimization on a workstation. In: Dean, N., Shannon, G.E. (eds.) Computational Support for Discrete Mathematics, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 15, pp. 119–136. American Mathematical Society, Providence (1994)
24. Warme, D. M.: Spanning Trees in Hypergraphs with Applications to Steiner Trees. PhD thesis, University of Virginia, (1998)
25. Warme, D.M., Winter, P., Zachariasen, M.: Exact algorithms for plane Steiner tree problems: a computational study. In: Du, D.-Z., Smith, J.M., Rubinstein, J.H. (eds.) Advances in Steiner Trees, pp. 81–116. Kluwer Academic Publishers, Boston (2000)
26. Widmayer, P., Wu, Y.F., Wong, C.K.: On some distance problems in fixed orientations. SIAM J. Comput. **16**(4), 728–746 (1987)
27. Winter, P.: An algorithm for the Steiner problem in the Euclidean plane. Networks **15**, 323–345 (1985)
28. Winter, P., Zachariasen, M.: Euclidean Steiner minimum trees: an improved exact algorithm. Networks **30**, 149–166 (1997)
29. Zachariasen, M.: Rectilinear full Steiner tree generation. Networks **33**, 125–143 (1999)
30. Zachariasen, M., Rohe, A.: Rectilinear group Steiner trees and applications in VLSI design. Math. Program. **94**, 407–433 (2003)
31. Zachariasen, M., Winter, P.: Concatenation-based greedy heuristics for the Euclidean Steiner tree problem. Algorithmica **25**, 418–437 (1999)
32. Zachariasen, M., Winter, P.: Obstacle-avoiding Euclidean Steiner trees in the plane: an exact algorithm. In: Workshop on Algorithm Engineering and Experimentation (ALENEX), Lecture Notes in Computer Science, vol. 1619, pp. 282–295. Springer, Baltimore (1999)