A Note On Universal Point Sets for Planar Graphs

Manfred Scheucher* Hendrik Schrezenmaier* Raphael Steiner*

Abstract

We investigate which planar point sets allow simultaneous straight-line embeddings of all planar graphs on a fixed number of vertices. We first show that $(1.293 - o(1))n$ points are required to find a straight-line drawing of each n-vertex planar graph (vertices are drawn as the given points); this improves the previous best constant 1.235 by Kurowski (2004).

Our second main result is based on exhaustive computer search: We show that no set of 11 points exists, on which all planar 11-vertex graphs can be simultaneously drawn plane straight-line. This strengthens the result by Cardinal, Hoffmann, and Kusters (2015), that all planar graphs on $n \leq 10$ vertices can be simultaneously drawn on particular "universal" sets of n points while there are no universal sets for $n \geq 15$. Moreover, we provide a set of 23 planar 11-vertex graphs which cannot be simultaneously drawn on any set of 11 points. This, in fact, is another step towards a (negative) answer of the question, whether every two planar graphs can be drawn simultaneously – a question raised by Brass, Cenek, Duncan, Efrat, Erten, Ismailescu, Kobourov, Lubiw, and Mitchell (2007).

1 Introduction

A point set S in the Euclidean plane is called *n-universal* for a family $\mathcal{G}$ of planar n-vertex graphs if every graph G from $\mathcal{G}$ admits a plane straight-line embedding such that the vertices are drawn as points from S. A point set, which is n-universal for the family of all planar graphs, is simply called *n-universal*. We denote by $f_p(n)$ the size of a minimal n-universal set (for planar graphs), and by $f_s(n)$ the size of a minimal n-universal set for stacked triangulations, where stacked triangulations (a.k.a. planar 3-trees) are defined as follows:

Definition 1 (Stacked Triangulations)**.** *Starting from a triangle, one may obtain any stacked triangulation by repeatedly inserting a new vertex inside a face (including the outer face) of the actual triangulation and making it adjacent to all the three vertices contained in the face.*

Figures 2 and 3 show examples of stacked triangulations on 11 vertices.

De Fraysseix, Pach, and Pollack [DFPP90] showed that every planar n-vertex graph admits a straight-line embedding on an $(2n-4) \times (n-2)$ grid – even if the combinatorial embedding (including the choice of the outer face) is prescribed. Moreover, the graphs are only embedded on a triangular subset of the grid. Hence, $f_p(n) \leq n^2 - O(n)$. This bound was further improved to the currently best known bound $f_p(n) \leq \frac{n^2}{4} - O(n)$ [BCDE14] (see also [Sch90, Bra08]). Also various subclasses of planar graphs have been studied intensively: Any stacked triangulation on n vertices (with a fixed outer cell) can be drawn on a particular set of $f_s(n) \leq O(n^{3/2} \log n)$ points [FT15]. For outerplanar graphs, it is known that any set of n points in general position is

*Institut für Mathematik, Technische Universität Berlin, Germany,
{scheucher,schrezen,steiner}@math.tu-berlin.de

n-universal [PGMP91, CU96]. For 2-outerplanar graphs and for simply nested graphs an upper bound of $O(n \log n)$ is known [ABDB$^+$18].

Concerning the lower bound on $f_p(n)$ and $f_s(n)$, respectively, the relation $n \leq f_s(n) \leq f_p(n)$ obviously holds for any $n \in \mathbb{N}$. The first non-trivial lower bound on the size of n-universal sets was also given by de Fraysseix, Pach, and Pollack [DFPP90], who showed a lower bound of $f_p(n) \geq n + (1 - o(1))\sqrt{n}$. Chrobak and Karloff [CK89] further improved the lower bound to $(1.098 - o(1))n$, and the multiplicative constant was later on improved to 1.235 by Kurowski [Kur04]. In fact, Kurowski's lower bound even applies to $f_s(n)$.

Cardinal, Hoffmann, and Kusters [CHK15] showed that n-universal sets of size n exist for every $n \leq 10$, whereas for $n \geq 15$ no such set exists – not even for stacked triangulations:

$$f_p(n) = f_s(n) = n \text{ for } n \leq 10 \qquad \text{and} \qquad f_p(n) \geq f_s(n) > n \text{ for } n \geq 15.$$

Moreover, they found a collection of 7,393 planar graphs on $n = 35$ vertices which cannot be simultaneously drawn straight-line on a common set of n points. We call such a collection of graphs a *conflict collection*. This was a first big step towards an answer to the question by Brass and others [BCD$^+$07], which can be reformulated as follows:

Question 1. *Is there a conflict collection of size 2?*

2 Outline

Our first result is the following theorem, which further improves the lower bound on $f_s(n)$. We present its proof in Section 3.

Theorem 1. *It holds that $f_s(n) \geq (\alpha - o(1))n$, where $\alpha = 1.293\ldots$ is the unique real-valued solution of the equation $\frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}} = 2$.*

In Section 4 we present our second result, which is another step towards a (negative) answer of Question 1 and strengthens the results from [CHK15]. Its proof is based on exhaustive computer search.

Theorem 2 (Computer-assisted)**.** *There is a conflict collection consisting of 23 stacked triangulations on 11 vertices. Furthermore, there is no conflict collection consisting of 16 triangulations on 11 vertices.*

Corollary 3. *There is no 11-universal set of size 11 – even for stacked triangulations. Hence, $f_p(11) \geq f_s(11) \geq 12$.*

Last but not least, since all known proofs for lower bounds make use of separating triangles, we also started the investigation of 4-connected triangulations. In Section 5 we present some n-universal sets of size n for 4-connected planar graphs for all $n \leq 17$.

3 Proof of Theorem 1

To prove the theorem, we use a refined counting argument based on a construction of a set of labeled stacked triangulations that was already introduced in [CHK15]. There it was used to disprove the existence of n-universal sets of $n \geq 15$ points for the family of stacked triangulations.

Definition 2 (Labeled Stacked Triangulations, cf. [CHK15, Section 3]). *For every integer $n \geq 4$, we define the family $\mathcal{T}_n$ of labeled stacked triangulations on the set of vertices $V_n := \{v_1, ..., v_n\}$ inductively as follows:*

- *$\mathcal{T}_4$ consists only of the complete graph K_4 with labels $v_1, \ldots, v_4$.*
- *If T is a labeled graph in $\mathcal{T}_{n-1}$ with $n \geq 5$, and $v_iv_jv_k$ defines a face of T, then the graph obtained from T by stacking the new vertex v_n to $v_iv_jv_k$ (i.e., connecting it to v_i, v_j, and v_k) is a member of $\mathcal{T}_n$.*

It is important to notice that, when speaking of $\mathcal{T}_n$, we distinguish between elements if they are distinct as *labeled graphs*, even if their underlying graphs are isomorphic. The essential ingredient we will need from [CHK15] is the following.

Lemma 4 (cf. [CHK15, Lemmas 1 and 2]).

(i) For any $n \geq 4$, the family $\mathcal{T}_n$ contains exactly $2^{n-4}(n-3)!$ labeled stacked triangulations.

(ii) Let $P_n = \{p_1, \ldots, p_n\}$ be a set of $n \geq 4$ labeled points in the plane. Then for any bijection $\pi : V_n \to P_n$, there is at most one $T \in \mathcal{T}_n$ such that the embedding of T, which maps each vertex v_i to point $\pi(v_i)$, defines a straight-line-embedding of T.

We need the following simple consequence of the above:

Corollary 5. *Let $P = \{p_1, \ldots, p_m\}$ be a set of $m \geq n \geq 4$ labeled points in the plane. Then for any injection $\pi : V_n \to P$, there is at most one $T \in \mathcal{T}_n$ such that the embedding of T, which maps each vertex v_i to point $\pi(v_i)$, defines a straight-line-embedding of T.*

Proof. Let $T_1, T_2 \in \mathcal{T}_n$ be two stacked triangulations such that π describes a plane straight-line embedding of both. Since π is an injection, this means that π defines a straight-line embedding of both T_1, T_2 on the sub-point set $Q := \pi(V_n)$ of P of size n. Applying Lemma 4(ii) to the bijection $\pi : V_n \to Q$ and T_1, T_2, we deduce $T_1 = T_2$. This proves the claim. □

We are now ready to prove Theorem 1.

Proof of Theorem 1. Let $n \geq 4$ be arbitrary and $m := f_s(n) \geq n$. There exists an n-universal point set $P = \{p_1, \ldots, p_m\}$ for all stacked triangulations, hence for every $T \in \mathcal{T}_n$ there exists a straight-line embedding of T on P, with (injective) vertex-mapping $\pi : V_n \to P$. By Corollary 5, we know that no two stacked triangulations from $\mathcal{T}_n$ (each of which has the same vertex set) yield the same injection π. Consequently, by Lemma 4(i), we have

$$2^{n-4}(n-3)! = |\mathcal{T}_n| \leq \frac{m!}{(m-n)!},$$

which means

$$\frac{1}{16n(n-1)(n-2)} 2^n \leq \binom{m}{n} = \binom{f_s(n)}{n}.$$

Let $\beta(n) := \frac{f_s(n)}{n}$. Using the fact that (Stirling-approximation)

$$\binom{f_s(n)}{n} \sim \underbrace{\sqrt{\frac{f_s(n)}{2\pi n(f_s(n)-n)}}}_{\leq 1} \frac{f_s(n)^{f_s(n)}}{n^n (f_s(n)-n)^{f_s(n)-n}} \leq \left(\frac{\beta(n)^{\beta(n)}}{(\beta(n)-1)^{\beta(n)-1}} \right)^n,$$

we deduce (taking logarithms) that:

$$(1-o(1))n \leq \log_2 \left(\frac{\beta(n)^{\beta(n)}}{(\beta(n)-1)^{\beta(n)-1}} \right) n \Longleftrightarrow 2-o(1) \leq \frac{\beta(n)^{\beta(n)}}{(\beta(n)-1)^{\beta(n)-1}}.$$

Consequently, $\beta(n) \geq (1-o(1))\alpha$, where α is the unique solution to $\frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}} = 2$. This proves $f_s(n) = n \cdot \beta(n) \geq (1-o(1))\alpha n$, which is the claim. □

4 Proof of Theorem 2 and Corollary 3

In the following, we outline the strategy which we have used to find a conflict collection of 23 stacked 11-vertex triangulations. A reader who is mainly interested in verifying our computational results might want to jump directly to Section 4.5.

One fundamental observation which we use throughout this section is the following: if an n-universal point set has collinear points, then by perturbation one can obtain another n-universal point set *in general position*, i.e., with no collinear points. Hence, in the following we only consider point sets in general position. Also it is not hard too see that, if two point sets are *combinatorially equivalent*, i.e., there is a bijection such that the corresponding triples of points induce the same orientations, then both sets allow precisely the same straight-line drawings. Hence, in the following we further restrict our considerations to *(non-degenerated) order types*, i.e., the set of equivalence classes of point sets (in general position).

4.1 Enumeration of Order Types

The database of all order types of up to $n = 11$ points was developed by Aurenhammer, Aichholzer, and Krasser [AAK02, AK06] (see also Krasser's dissertation [Kra03]). The file for all order types of up to $n = 10$ points (each represented by a point set) is available online, while the file for $n = 11$ requires almost 100GB of storage and is available on demand [Aic]. Their algorithm starts with an *abstract order type* on $k-1$ points (which only encodes the triple orientations of a point set), computes its dual pseudoline arrangement, and inserts a k-th pseudoline in all possible ways. Due to geometrical constraints, there are in fact abstract order types enumerated which do not have a realization as a point set. However, since every order type is in fact also an abstract order type, it is sufficient for our purposes to test all abstract order types – independent from realizability.

For means of redundancy and to provide a fully checkable and autonomous proof, we have implemented an alternative algorithm to enumerate all abstract order types based on the following idea: Given a set of points $s_1, \ldots, s_n$ with $s_i = (x_i, y_i)$ sorted left to right[1], and let

$$\chi_{ijk} := \operatorname{sgn} \det \begin{pmatrix} 1 & 1 & 1 \\ x_i & x_j & x_k \\ y_i & y_j & y_k \end{pmatrix} \in \{-1, 0, +1\}$$

denote the induced triple orientations, then the *signotope axioms* assert that, for every 4-tuple s_i, s_j, s_k, s_l with $i < j < k < l$, the sequence

$$\chi_{ijk},\ \chi_{ijl},\ \chi_{ikl},\ \chi_{jkl}$$

[1] in the dual line arrangement the lines are sorted by increasing slope

(index-triples are in lexicographic order) changes its sign at most once. For more information on the signotope axioms we refer to Felsner and Weil [FW01] (see also [BFK15]).

Given an abstract order type on $k-1$ points, we insert a k-th point in all possible ways, such that the signotope axioms are preserved. With our C++ implementation, we managed to verify the numbers of abstract order types from [AAK02, AK06, Kra03]. In fact, the enumeration of all 2,343,203,071 abstract order types of up to $n = 11$ points (cf. OEIS/A6247) can be done within about 20 hours on a single CPU.

4.2 Enumeration of Planar Graphs

To enumerate all non-isomorphic maximal planar graphs on 11 vertices (i.e, triangulations), we have used the plantri graph generator (version 4.5) [BM99]. It is worth to note that also the nauty graph generator [MP14] can be used for the enumeration because the number of all (not necessarily planar) graphs on 11 vertices is not too large and the database can be filtered for planar graphs in reasonable time – negligible compared to the CPU time which we have used for later computations. For various computations on graphs, such as filtering stacked triangulations or to produce graphs for this paper, we have used SageMath [S$^+$18a][2].

4.3 Deciding Universality using a SAT Solver

For a given point set S and a planar graph $G = (V, E)$ we model a propositional formula in conjunctive normal form (CNF) which has a solution if and only if G can be embedded on S – in fact, the variables encode a straight-line drawing.

To model the CNF, we have used

- the variables M_{vp} to describe, whether vertex v is mapped to point p, and
- the variables A_{pq} to describe, whether the straight-line segment pq between the two points p and q is "active" in a drawing.

It is not hard to use a CNF to assert that such a vertex-to-point mapping is bijective. Also it is easy to assert that, if two adjacent vertices u and v are mapped to points p and q, then the straight-line segment pq is active. For each pair of crossing straight-line segments pq and rs (dependent on the order type of the point set) at least one of the two segments is not allowed to be active.

Implementation detail: We have implemented a C++ routine which, given a point set and a graph as input, creates an instance of the above described model and then uses the solver MiniSat 2.2.0 [ES03] to decide whether the graph admits a straight-line embedding.

4.4 Finding Conflict Collections – A Quantitive Approach

Before we actually tested whether a set of 11 points is 11-universal or not, we discovered a few necessary criteria for the point set, which can be checked much more efficiently. These considerations allowed a significant reduction of the total computation times.

[2] We recommend the Sage Reference Manual on Graph Theory [S$^+$18b] and its collection of excellent examples.

Phase 1: First of all, an 11-universal point set – if one exists – trivially has a triangular convex hull. Secondly, the planar graph depicted in Figure 1 asserts an 11-universal set S to have a certain structure. If the embedding is as on the left of Figure 1, then one of the two degree 3 vertices is drawn as extremal point of S, i.e., lies on the boundary of the convex hull $\text{conv}(S)$ of S. After the removal of this particular point, the remaining 10 points have 4 convex layers of sizes 3, 3, 3, and 1, respectively. If the embedding is as on the right of Figure 1, then either one or two points of the blue triangle are drawn as extremal points of S (recall the triangular convex hull of S). And again, the points inside the blue triangle and outside the blue triangle have convex layers of sizes 3, 3, 1, and 3, 1, respectively. Altogether, only 293,114,696 of the 2,343,203,071 abstract order types on 11 points fulfill the two conditions.

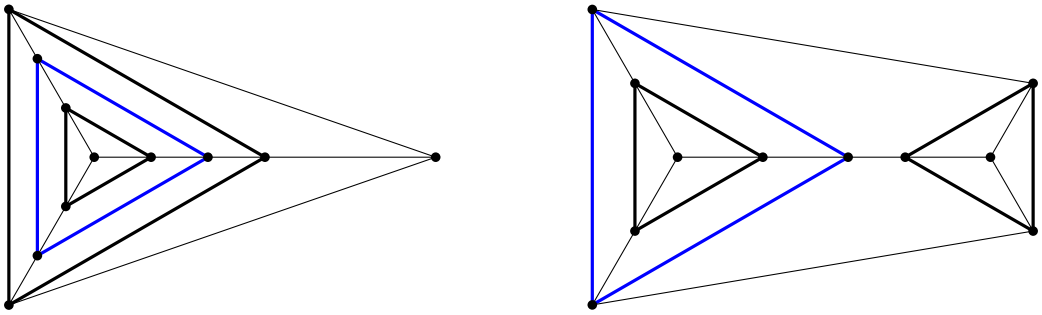

Figure 1: The two embeddings of a graph, which forces the point set to have a certain structure. Each of the vertices of the blue triangle connects to one of the vertices of the two copies of K_4.

There exist stacked triangulations on 11 points in which every face is incident to a degree-3-vertex; see for example G_{11} in Figure 2. Independent from the embedding of such a graph, there is a degree-3-vertex on the outer face, and hence all inner points lie inside a triangle spanned by an interior point and two extremal points.

Phase 2: For each of the 293,114,696 abstract order types on 11 points which fulfill the conditions above, we have tested the embeddability of all maximal planar graphs on n vertices separately using a SAT-solver based approach [ES04]. In fact, as soon as one graph was not embeddable, the remaining graphs needed not to be checked. To speed up the computations we have used a priority queue: a graph which does not admit an embedding gets increased priority for other point sets to be tested first.

To keep the conflict collection as small as possible, we first filtered out all point sets which do not allow a simultaneous embedding of all planar graphs on 11 vertices with maximum degree 10. Only 278,530 of the 293,114,696 abstract order types remained (computation time about 100 CPU days). It is worth to note that there are 82 maximal planar graphs on 11 vertices with maximum degree 10 (cf. OEIS/A207), and that each of these graphs is a stacked triangulation.

At this point one can check with only a few CPU hours that the remaining 278,530 abstract order types are not 11-universal. Moreover, since some stacked triangulations on 11 vertices (e.g. G_8 from Figure 2) contain the graph from Figure 1 as a subgraph, the statement even applies to stacked triangulations. Consequently, the family of all 434 stacked triangulations on 11 vertices (cf. OEIS/A27610) is a conflict collection, and Corollary 3 follows directly.

Phase 3: To find a smaller conflict collection, we tested for each of the 434 stacked triangulations and each of the 278,530 remaining abstract order types, whether an embedding is possible (additional 35 CPU days). We used this binary information to formulate an integer

program searching for a minimal set of triangulations, without simultaneous embedding. Using the Gurobi solver (version 8.0.0) [Gur18], we managed to find a collection $\mathcal{G}$ of 11 stacked triangulations which cannot be embedded simultaneously[3]; see Listing 1 and Figure 2.

In fact, the Gurobi IP solver showed optimality and thus no conflict collection of size less than 11 can exist for $n = 11$. Since we asserted in Phases 1 and 2 that

(1) the graph in Figure 1,

(2) a triangulation where every face is incident to a vertex of degree 3, and

(3) all 82 triangulations with maximum degree 10

occur in the conflict collection, this yields a conflict collection of size $95 = 1 + 1 + 82 + 11$. In fact, since this subset of 11 stacked triangulations contains triangulations fulfilling properties (1) and (2) (see e.g. graphs G_8 and G_{11} in Figure 2), we indeed have a conflict collection of size 93.

Phase 4: Recall that a minimal conflict collection not necessarily needs to fulfill the properties (1)–(3). Hence we again repeat the strategy from Phase 2, except that we test for the embeddability of the 11 stacked triangulations from the collection $\mathcal{G}$ obtained in Phase 3 instead of the 82 maximal planar graphs on 11 vertices with maximum degree 10.

After another 230 days of CPU time, our program had filtered out 17,533 of the 293,114,696 abstract order types (obtained in Phase 1) which allow a simultaneous embedding of the 11 stacked triangulations from $\mathcal{G}$.

Phase 5: As the reader might already guess, we proceed as in Phase 3: we tested for each of the 434 stacked triangulations and each of the 17,533 order types from Phase 4, whether an embedding is possible (only 2 CPU days). Using the Gurobi solver, we managed to find a collection $\mathcal{H}$ of 12 stacked triangulations, which cannot be simultaneously embedded on those order types; see Listing 1 and Figure 2.

Together with the 11 stacked triangulations from $\mathcal{G}$ we obtain a conflict collection of size 23, and the first part of Theorem 2 follows.

Phase 6: As the solution of the integer programming instance from Phase 5 found by Gurobi was optimal, any conflict collection of stacked triangulations must have size at least 12. To further improve this lower bound, we have repeated our computations for the union of the two sets of point sets obtained in Phase 3 and Phase 5, respectively. Using Gurobi, we could optimally solve the instance and hence any conflict collection consisting of 11-vertex stacked triangulations has size at least 17.

Surprisingly, when replacing the 434 stacked triangulations by the collection of all 1,249 triangulations (cf. OEIS/A109), the Gurobi solver also showed that any conflict collection of (arbitrary) 11-vertex triangulations has size at least 17. The computation took about 1 day using 20 threads in parallel.

For means of redundancy, we have verified all lower bounds obtained by Gurobi also using CPLEX (version 12.8.0.0) [IBM18], which performed similar to Gurobi (with 20 threads in parallel).

This completes the proof of the second part of Theorem 2.

[3] Indeed it was a funny coincidence that this set has cardinality 11, which is also the number of vertices.

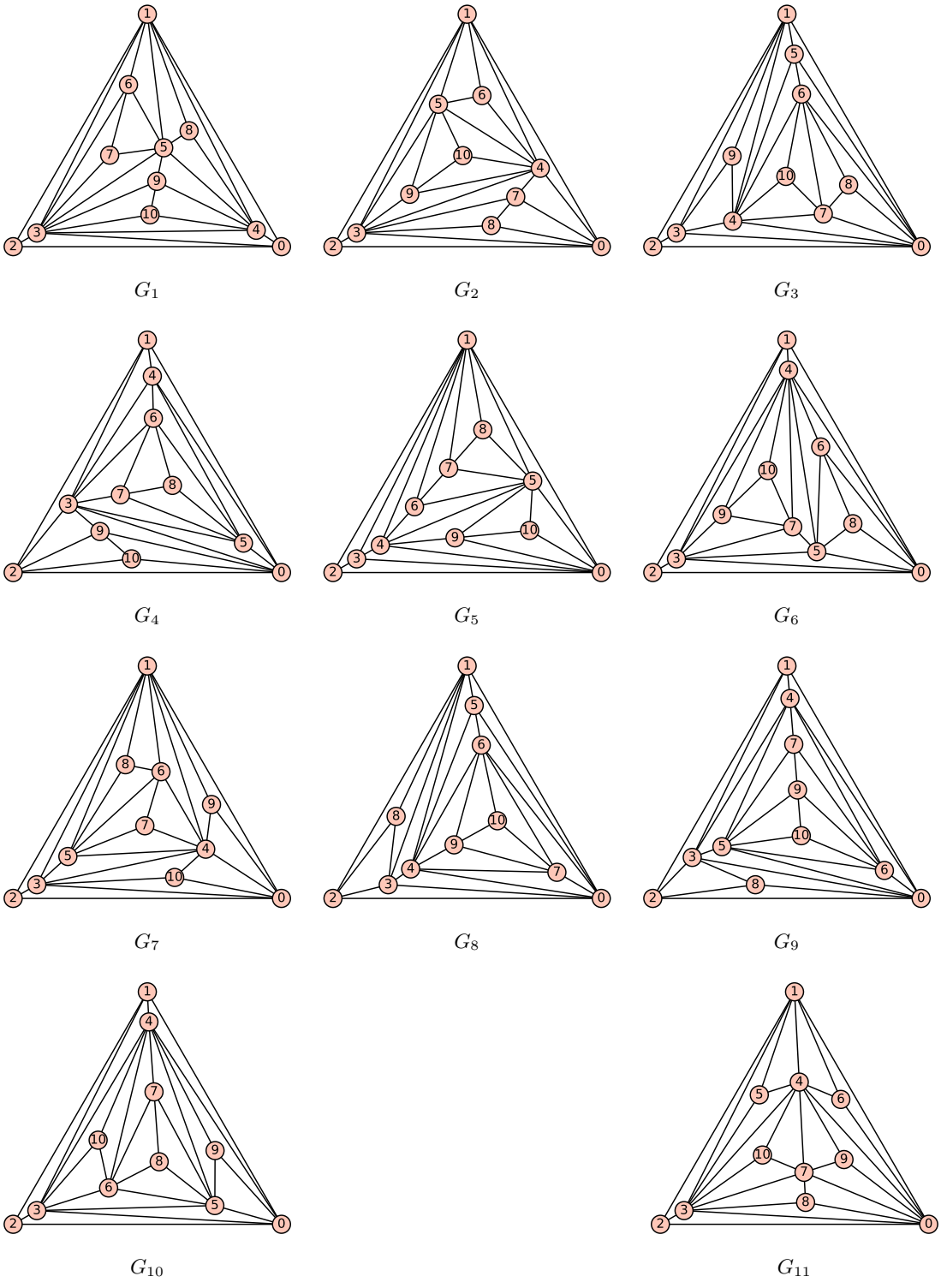

Figure 2: The 11 stacked triangulations from the conflict collection $\mathcal{G}$ obtained in Phase 3.

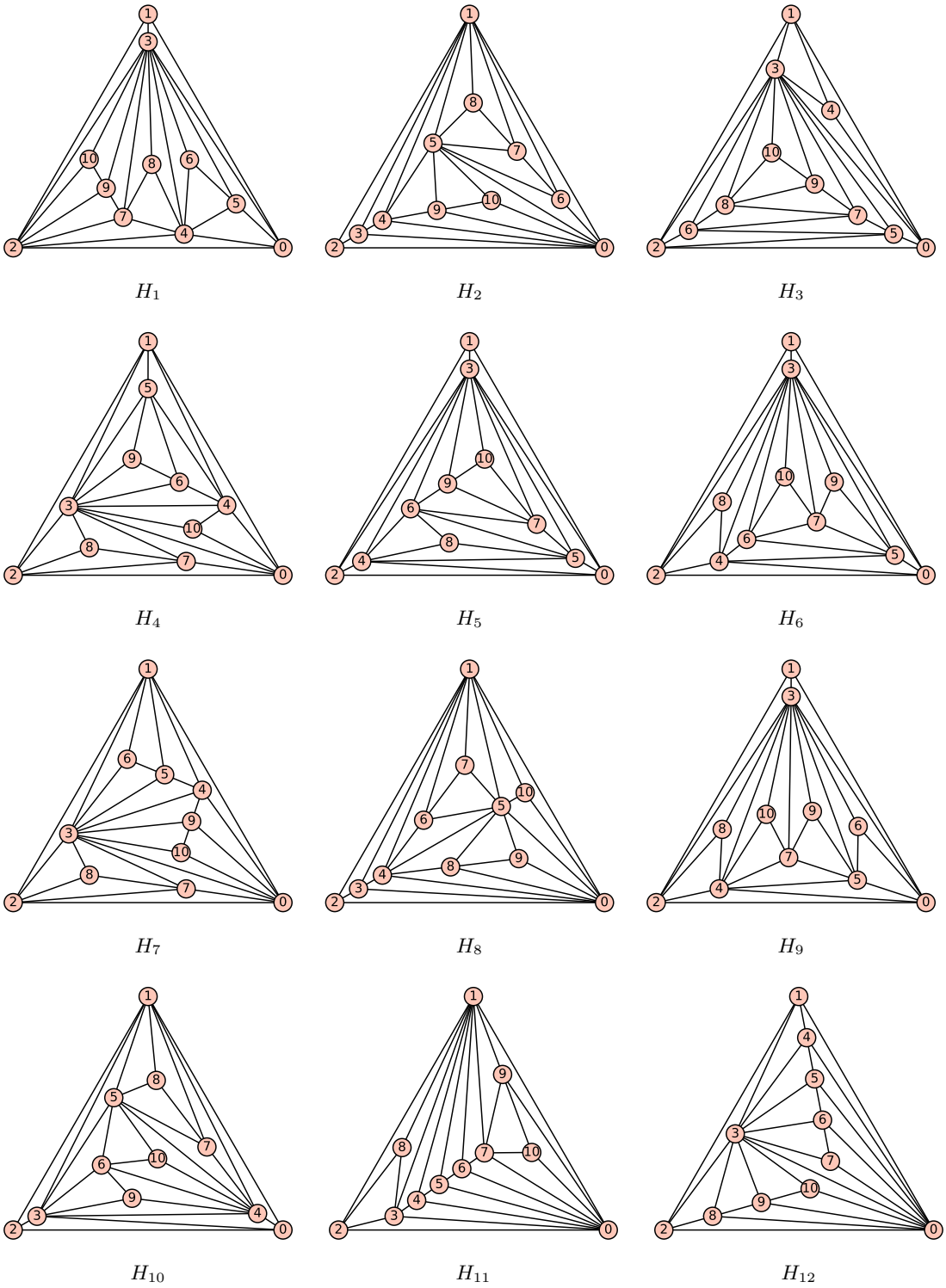

Figure 3: The 12 stacked triangulations from the conflict collection $\mathcal{H}$ obtained in Phase 5.

4.5 How to Verify our Results?

To verify the computational results which are essential for the proof of the first part of Theorem 2, one can enumerate all order types on 11 points and test the conflict collection of 23 triangulations (`data/triangulations/n11_conflicting23.txt`). Starting with the unique order type on 3 points (`data/order_types/n3_order_types.bin`), it takes about 1 CPU day to enumerate all order types on 11 points. By falsifying simultaneous embeddability of the 23 graphs (this computation takes about 200 CPU days, but can be run parallelized), the first part of Theorem 2 is then verified.

For the second part of the Theorem 2, one can filter the order types, which allow a simultaneous embedding of the triangulations from Phase 2 and 4, and then – using CPLEX or Gurobi – compute the minimum size of a conflicting collection among all 11-vertex triangulations and 11-vertex stacked triangulations, respectively. To save some computation time, we provide the filtered list in the files `data/triangulations/n11_after_phase2.bin.zip` and `n11_after_phase4.bin.zip`. The list of all (stacked) triangulations is provided in the files `n11_all_triangulations.txt` and `n11_all_stacked_triangulations.txt`.

A more detailed description is provided in Appendix A. The source codes of our programs and relevant data are available on the companion website [Sch].

5 Universal Sets for 4-Connected Graphs

For $n \leq 10$, examples of n-universal sets of n points for planar n-vertex graphs were already given in [CHK15]. To provide n-universal sets for 4-connected planar graphs for $n = 11, \ldots, 17$, we slightly adopted our framework. Again, we enumerated 4-connected planar triangulations using the plantri graph generator, and using our C++ implementation, tested for universality. Our idea to find the proposed point sets for $n = 11, \ldots, 17$ was to start with an $(n-1)$-universal set of $n-1$ points and insert an n-th point in all possible ways (cf. Section 4.1). The abstract order types obtained in this way – if they turned out to be universal – were then realized as point sets using the framework *pyotlib*[4]. The obtained sets are given in Listing 3.

It is also worth to note that the numbers of 4-connected triangulations for $n = 11, \ldots, 20$ are 25; 87; 313; 1,357; 6,244; 30,926; 158,428; 836,749; 4,504,607; 24,649,284 (cf. OEIS/A7021). Hence, even if a universal point set is known, it is getting more and more time consuming to verify n-universality as n gets larger (at least using our SAT solver approach).

6 Discussion

In Section 3, we provided an improved lower bound for $f_p(n)$ and $f_s(n)$. However, the best known general upper bounds remain far from linear.

In Section 4, we have applied the ideas from Phases 2 and 3 twice (cf. Phases 4 and 5) to reduce the size of a conflict collection. One could further proceed with this strategy to find even smaller conflict collections (if such exist). Also one could simply test whether all elements from the conflict collection are indeed necessary, or whether certain elements can be removed. Note that, to compute a minimal conflict collection for $n = 11$, one could theoretically check which

[4] The "**py**thon **o**rder **t**ype **lib**rary" was initiated during the Bachelor's studies of the first author [Sch14] and provides many features to work with (abstract) order types such as local search techniques, realization or proving non-realizability of abstract order types, coordinate minimization and "beautification" for nicer visualizations. For more information, please consult the author.

graphs admit an embedding on which point set and then find a minimal set cover as described in Phase 3 (Section 4). In practice, however, formulating such a minimal set cover instance (as integer program) is not reasonable because testing the embeddability of every graph in every point set would be an extremely time consuming task. (Recall that we used a priority queue to speed up our computation, so only a few pairs were actually tested. Also recall that, to generate the set cover instances, we only looked at a comparably small number of order types.) And even if such an instance was formulated, due to its size, the IP/set cover might not be solvable optimally in reasonable time.

Besides the computations for $n = 11$ points, we also adopted our program to find all n-universal order types on n points for every $n \leq 10$, and hence could verify the results from [CHK15, Table 1]. To be precise, we found 5,956 9-universal abstract order types on $n = 9$ points, whereas only 5,955 are realizable as point sets. It is worth to note that there is exactly one non-realizable abstract order type on 9 points in the projective plane, which is dual to the simple non-Pappus arrangement, and that all abstract order types on $n \leq 8$ points are realizable. Besides the already known 2,072 realizable order types on 10 points, no further non-realizable 10-universal abstract order types were found. For more details on realizability see for example [Kra03] or [FG18].

Unfortunately, we do not have an argument for subsets/supersets of n-universal point sets, and thus the question for $n = 12, 13, 14$ remains open. However, based on computational evidence (see also [CHK15, Table 1]), we strongly conjecture that no n-universal set of n points exists for $n \geq 11$.

As mentioned in the introduction of this paper, various graph classes have been studied for this problem. Even though our contribution on 4-connected planar graphs in Section 5 is rather small, it gives some evidence that comparably less points are needed to embed 4-connected planar graphs. In fact, we would not be surprised if n-universal sets of n points exist for 4-connected planar graphs.

Acknowledgements

Manfred Scheucher was supported by DFG Grant FE 340/12-1. Hendrik Schrezenmaier was supported by DFG Grant FE-340/11-1. Raphael Steiner was supported by DFG-GRK 2434.

References

[AAK02] O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating Order Types for Small Point Sets with Applications. *Order*, 19(3):265–281, 2002.

[ABDB+18] P. Angelini, T. Bruckdorfer, G. Di Battista, M. Kaufmann, T. Mchedlidze, V. Roselli, and C. Squarcella. Small universal point sets for k-outerplanar graphs. *Discrete & Computational Geometry*, pages 1–41, 2018.

[Aic] O. Aichholzer. Enumerating Order Types for Small Point Sets with Applications. `http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/ordertypes/`.

[AK06] O. Aichholzer and H. Krasser. Abstract Order Type Extension and New Results on the Rectilinear Crossing Number. *Computational Geometry: Theory and Applications*, 36(1):2–15, 2006.

[BCD+07] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. P. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007.

[BCDE14] M. J. Bannister, Z. Cheng, W. E. Devanny, and D. Eppstein. Superpatterns and Universal Point Sets. *Journal of Graph Algorithms and Applications*, 18(2):177–209, 2014.

[BFK15] M. Balko, R. Fulek, and J. Kynčl. Crossing Numbers and Combinatorial Characterization of Monotone Drawings of K_n. *Discrete & Computational Geometry*, 53(1):107–143, 2015.

[BM99] G. Brinkmann and B. D. McKay. Fast generation of some classes of planar graphs. *Electronic Notes in Discrete Mathematics*, 3:28–31, 1999.

[Bra08] F. J. Brandenburg. Drawing planar graphs on $\frac{8}{9}n^2$ area. *Electronic Notes in Discrete Mathematics*, 31:37–40, 2008.

[CHK15] J. Cardinal, M. Hoffmann, and V. Kusters. On Universal Point Sets for Planar Graphs. *Journal of Graph Algorithms and Applications*, 19(1):529–547, 2015.

[CK89] M. Chrobak and H. J. Karloff. A Lower Bound on the Size of Universal Sets for Planar Graphs. *ACM SIGACT News*, 20(4):83–86, 1989.

[CU96] N. Castañeda and J. Urrutia. Straight Line Embeddings of Planar Graphs on Point Sets. In *Proceedings of the 8th Canadian Conference on Computational Geometry (CCCG'96)*, pages 312–318, 1996.
`http://www.cccg.ca/proceedings/1996/cccg1996_0052.pdf`.

[DFPP90] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[ES03] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of Theory and Applications of Satisfiability Testing - SAT 2003*, pages 502–518, 2003.

[ES04] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003*, pages 502–518. Springer, 2004.

[FG18] S. Felsner and J. E. Goodman. Pseudoline Arrangements. In Toth, O'Rourke, and Goodman, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 3 edition, 2018.

[FT15] R. Fulek and C. D. Tóth. Universal point sets for planar three-trees. *Journal of Discrete Algorithms*, 30:101–112, 2015.

[FW01] S. Felsner and H. Weil. Sweeps, Arrangements and Signotopes. *Discrete Applied Mathematics*, 109(1):67–94, 2001.

[GP83] J. E. Goodman and R. Pollack. Multidimensional Sorting. *SIAM Journal on Computing*, 12(3):484–507, 1983.

[Gur18] Gurobi Optimization, LLC. Gurobi Optimizer, 2018.
`http://www.gurobi.com`.

[IBM18] IBM ILOG CPLEX Optimization Studio, 2018. http://www.ibm.com/products/ilog-cplex-optimization-studio/.

[Kra03] H. Krasser. *Order Types of Point Sets in the Plane*. PhD thesis, Institute for Theoretical Computer Science, Graz University of Technology, Austria, 2003.

[Kur04] M. Kurowski. A $1.235n$ lower bound on the number of points needed to draw all n-vertex planar graphs. *Information Processing Letters*, 92(2):95–98, 2004.

[MP14] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[PGMP91] J. Pach, P. Gritzmann, B. Mohar, and R. Pollack. Embedding a planar triangulation with vertices at specified points. *American Mathematical Monthly*, 98:165–166, 1991.

[S+18a] W. A. Stein et al. *Sage Mathematics Software (Version 8.1)*. The Sage Development Team, 2018. http://www.sagemath.org.

[S+18b] W. A. Stein et al. *Sage Reference Manual: Graph Theory (Release 8.1)*, 2018. http://doc.sagemath.org/pdf/en/reference/number_fields/number_fields.pdf.

[Sch] M. Scheucher. Webpage: Source Codes and Data for Universal Point Sets. http://page.math.tu-berlin.de/~scheuch/supplemental/universal_sets.

[Sch90] W. Schnyder. Embedding Planar Graphs on the Grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148. Society for Industrial and Applied Mathematics, 1990.

[Sch14] M. Scheucher. *On Order Types, Projective Classes, and Realizations*. Bachelor's thesis, Graz University of Technology, Austria, 2014. http://www.math.tu-berlin.de/~scheuch/publ/bachelors_thesis_tm_2014.pdf.

```
G_1  = [(0,1),(0,2),(0,3),(0,4),(1,2),(1,3),(1,4),(1,5),(1,6),
        (1,8),(2,3),(3,4),(3,5),(3,6),(3,7),(3,9),(3,10),(4,5),
        (4,8),(4,9),(4,10),(5,6),(5,7),(5,8),(5,9),(6,7),(9,10)]

G_2  = [(0,1),(0,2),(0,3),(0,4),(0,7),(0,8),(1,2),(1,3),(1,4),
        (1,5),(1,6),(2,3),(3,4),(3,5),(3,7),(3,8),(3,9),(4,5),
        (4,6),(4,7),(4,9),(4,10),(5,6),(5,9),(5,10),(7,8),(9,10)]

G_3  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(1,2),
        (1,3),(1,4),(1,5),(1,9),(2,3),(3,4),(3,9),(4,5),(4,6),
        (4,7),(4,9),(4,10),(5,6),(6,7),(6,8),(6,10),(7,8),(7,10)]

G_4  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,9),(0,10),(1,2),(1,3),
        (1,4),(2,3),(2,9),(2,10),(3,4),(3,5),(3,6),(3,7),(3,9),
        (4,5),(4,6),(5,6),(5,7),(5,8),(6,7),(6,8),(7,8),(9,10)]

G_5  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,9),(0,10),(1,2),(1,3),
        (1,4),(1,5),(1,6),(1,7),(1,8),(2,3),(3,4),(4,5),(4,6),
        (4,9),(5,6),(5,7),(5,8),(5,9),(5,10),(6,7),(7,8),(9,10)]

G_6  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,8),(1,2),(1,3),
        (1,4),(2,3),(3,4),(3,5),(3,7),(3,9),(4,5),(4,6),(4,7),
        (4,9),(4,10),(5,6),(5,7),(5,8),(6,8),(7,9),(7,10),(9,10)]

G_7  = [(0,1),(0,2),(0,3),(0,4),(0,9),(0,10),(1,2),(1,3),(1,4),
        (1,5),(1,6),(1,8),(1,9),(2,3),(3,4),(3,5),(3,10),(4,5),
        (4,6),(4,7),(4,9),(4,10),(5,6),(5,7),(5,8),(6,7),(6,8)]

G_8  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(1,2),(1,3),
        (1,4),(1,5),(1,8),(2,3),(2,8),(3,4),(3,8),(4,5),(4,6),
        (4,7),(4,9),(5,6),(6,7),(6,9),(6,10),(7,9),(7,10),(9,10)]

G_9  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,8),(1,2),(1,3),
        (1,4),(2,3),(2,8),(3,4),(3,5),(3,8),(4,5),(4,6),(4,7),
        (5,6),(5,7),(5,9),(5,10),(6,7),(6,9),(6,10),(7,9),(9,10)]

G_10 = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,9),(1,2),(1,3),(1,4),
        (2,3),(3,4),(3,5),(3,6),(3,10),(4,5),(4,6),(4,7),(4,9),
        (4,10),(5,6),(5,7),(5,8),(5,9),(6,7),(6,8),(6,10),(7,8)]

G_11 = [(0,1),(0,2),(0,3),(0,4),(0,6),(0,7),(0,8),(0,9),(1,2),
        (1,3),(1,4),(1,5),(1,6),(2,3),(3,4),(3,5),(3,7),(3,8),
        (3,10),(4,5),(4,6),(4,7),(4,9),(4,10),(7,8),(7,9),(7,10)]
```

Listing 1: Edge-lists of the 11 stacked triangulations from collection $\mathcal{G}$ obtained in Phase 3.

```
H_1  = [(0,1),(0,2),(0,3),(0,4),(0,5),(1,2),(1,3),(2,3),(2,4),
        (2,7),(2,9),(2,10),(3,4),(3,5),(3,6),(3,7),(3,8),(3,9),
        (3,10),(4,5),(4,6),(4,7),(4,8),(5,6),(7,8),(7,9),(9,10)]

H_2  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,9),(0,10),(1,2),
        (1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(2,3),(3,4),(4,5),
        (4,9),(5,6),(5,7),(5,8),(5,9),(5,10),(6,7),(7,8),(9,10)]

H_3  = [(0,1),(0,2),(0,3),(0,4),(0,5),(1,2),(1,3),(1,4),(2,3),
        (2,5),(2,6),(3,4),(3,5),(3,6),(3,7),(3,8),(3,9),(3,10),
        (5,6),(5,7),(6,7),(6,8),(7,8),(7,9),(8,9),(8,10),(9,10)]

H_4  = [(0,1),(0,2),(0,3),(0,4),(0,7),(0,10),(1,2),(1,3),(1,4),
        (1,5),(2,3),(2,7),(2,8),(3,4),(3,5),(3,6),(3,7),(3,8),
        (3,9),(3,10),(4,5),(4,6),(4,10),(5,6),(5,9),(6,9),(7,8)]

H_5  = [(0,1),(0,2),(0,3),(0,4),(0,5),(1,2),(1,3),(2,3),(2,4),
        (3,4),(3,5),(3,6),(3,7),(3,9),(3,10),(4,5),(4,6),(4,8),
        (5,6),(5,7),(5,8),(6,7),(6,8),(6,9),(7,9),(7,10),(9,10)]

H_6  = [(0,1),(0,2),(0,3),(0,4),(0,5),(1,2),(1,3),(2,3),(2,4),
        (2,8),(3,4),(3,5),(3,6),(3,7),(3,8),(3,9),(3,10),(4,5),
        (4,6),(4,8),(5,6),(5,7),(5,9),(6,7),(6,10),(7,9),(7,10)]

H_7  = [(0,1),(0,2),(0,3),(0,4),(0,7),(0,9),(0,10),(1,2),(1,3),
        (1,4),(1,5),(1,6),(2,3),(2,7),(2,8),(3,4),(3,5),(3,6),
        (3,7),(3,8),(3,9),(3,10),(4,5),(4,9),(5,6),(7,8),(9,10)]

H_8  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,8),(0,9),(0,10),(1,2),
        (1,3),(1,4),(1,5),(1,6),(1,7),(1,10),(2,3),(3,4),(4,5),
        (4,6),(4,8),(5,6),(5,7),(5,8),(5,9),(5,10),(6,7),(8,9)]

H_9  = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(1,2),(1,3),(2,3),
        (2,4),(2,8),(3,4),(3,5),(3,6),(3,7),(3,8),(3,9),(3,10),
        (4,5),(4,7),(4,8),(4,10),(5,6),(5,7),(5,9),(7,9),(7,10)]

H_10 = [(0,1),(0,2),(0,3),(0,4),(1,2),(1,3),(1,4),(1,5),(1,7),
        (1,8),(2,3),(3,4),(3,5),(3,6),(3,9),(4,5),(4,6),(4,7),
        (4,9),(4,10),(5,6),(5,7),(5,8),(5,10),(6,9),(6,10),(7,8)]

H_11 = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,9),(0,10),
        (1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9),(2,3),
        (2,8),(3,4),(3,8),(4,5),(5,6),(6,7),(7,9),(7,10),(9,10)]

H_12 = [(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(0,9),
        (0,10),(1,2),(1,3),(1,4),(2,3),(2,8),(3,4),(3,5),(3,6),
        (3,7),(3,8),(3,9),(3,10),(4,5),(5,6),(6,7),(8,9),(9,10)]
```

Listing 2: Edge-lists of the 12 stacked triangulations from collection $\mathcal{H}$ obtained in Phase 5.

```
P_11 = [(0,22),(24,25),(24,24),(13,21),(16,16),
        (13,17),(5,19),(3,20),(22,7),(23,6),(25,0)]

P_12 = [(36,50),(49,0),(48,1),(39,11),(38,16),(37,28),
        (37,27),(36,30),(33,38),(32,41),(30,49),(0,45)]

P_13 = [(99,102),(92,0),(89,4),(58,46),(56,56),(59,65),(60,66),
        (55,75),(48,92),(46,97),(76,101),(45,101),(0,102)]

P_14 = [(306,358),(358,0),(354,2),(313,306),(304,302),
        (251,280),(243,270),(232,256),(174,177),(170,177),
        (159,163),(131,135),(122,125),(0,176)]

P_15 = [(0,0),(693,591),(692,481),(3,2),(261,161),(304,155),
        (451,123),(492,132),(532,141),(650,172),(650,168),
        (672,173),(684,174),(689,175),(693,20)]

P_16 = [(438,447),(447,0),(446,1),(343,122),(344,126),(379,259),
        (378,266),(376,277),(360,360),(355,360),(356,381),
        (436,446),(358,424),(357,433),(42,389),(0,383)]

P_17 = [(980,0),(0,735),(4,736),(301,810),(306,805),(596,695),
        (384,716),(415,709),(424,707),(974,10),(612,666),(975,6),
        (754,635),(834,609),(884,597),(890,962),(890,977)]
```

Listing 3: Universal set of n points for 4-connected planar graphs for $n = 11, \ldots, 17$.

A Detailed Description of Tools

In following we give a detailed description of the tools which are required to verify the proof of Theorem 2. Moreover, we exemplify how the tools can be used. Even though our C++ code is platform-independent, we assume that the reader uses a Unix/Linux operating system and only give usage examples for this particular setup. (We ran our experiments in Fedora 27 and openSUSE 15.)

A.1 Enumerating Abstract Order Types

extend_order_type We provide a C++ program `extend_order_type` which reads all abstract order types on a fixed number of points n from the input file, extends it in all possible ways, and writes all so-obtained abstract order types on $n+1$ points to an output file (without duplicates). The program (see the folder `cprogram/scripts/extend_order_type/`) can be built using qmake[5] and make. The following bash command builds the program:

```
$ qmake && make
```

The File Format Concerning the file format, we have to explain a little more theory: An abstract order type can be encoded by its triple orientations

$$\Lambda_{i,j,k} \in \{-1, 0, +1\} \text{ for each } 1 \leq i, j, k \leq n.$$

Since encoding this "big lambda matrix" uses a cubic amount of bits, it is more efficient to encode the "small lambda matrix", which has the following entries:

$$\lambda_{i,j} := |\{k \in \{1, \ldots, n\} \setminus \{i, j\} \colon \Lambda_{i,j,k} > 0\}| \text{ for each } 1 \leq i, j, k \leq n.$$

Diagonal elements $\lambda_{i,i}$ are omitted (or can be set to zero). This data structure was first introduced by Goodman and Pollack [GP83].

Small lambda matrices of (non-degenerated) abstract order types fulfill $\lambda_{i,j} + \lambda_{j,i} = n - 2$ (i.e., for each two fixed points i, j, any other point either lies on the left or on the right side of the directed line through i and j), hence only entries $\lambda_{i,j}$ with $1 \leq i < j \leq n$ need to be stored. Moreover, the first point can be assumed to lie on the boundary of the convex hull, and other points can be assumed to be sorted around the first point. Such a labeling of points is called "natural labeling" and yields $\Lambda_{1,i,j} = +$ for $1 < i < j \leq n$ and $\lambda_{1,j} = j - 1$ for all $1 < j \leq n$. Consequently, elements from the first row of the small lambda matrix need not to be stored.

Note that the lexicographically minimal small lambda matrix (over all labelings) – which we compute to distinguish different order types – is also naturally labeled.

Altogether, we encoded the entries $\lambda_{i,j}$ for $1 < i < j \leq n$ as 8-bit (1-byte) integers in lexicographic order, i.e,

$$\lambda_{2,3}, \lambda_{2,4}, \ldots, \lambda_{2,n}, \lambda_{3,4}, \lambda_{3,5}, \ldots, \lambda_{n-1,n}.$$

For more information we again refer to the articles by Aurenhammer, Aichholzer, and Krasser [AAK02, AK06], and the dissertation of Krasser [Kra03].

[5] In face, no Qt-specific features are used. Alternatively to qmake one could also use cmake or just use a stand-alone Makefile

Usage of the Program The program `extend_order_type` can be used as follows:

```
./extend_order_type [n] [order types file] [parts] [from part] [to part]
```

The following describes the parameters.

- "n" is the number of points in the abstract order types from the input file,
- "order types file" is the path to the input file,
- "parts" is the number of threads in total,
- "from part" is the id of the first thread to be run, and
- "to part" is the id of the first thread not to be run.

The difference "to part"-"from part" is precisely the number of threads to be started on the local machine. As an example, to start a computation on 4 machines with 4 threads each (16 threads in total), one can simply run one of the following commands on each of the machines:

```
./extend_order_type [n] [order types file] 16 0 4
./extend_order_type [n] [order types file] 16 4 8
./extend_order_type [n] [order types file] 16 8 12
./extend_order_type [n] [order types file] 16 12 16
```

We also provide our python script `create_jobs.py`, which we used to automatically create job files for the parallel computations on the cluster.

Complete Enumeration To generate all abstract order types, we start with a binary file `n3_order_types.bin` with content "0x00" (one byte) – this encodes the unique order type on 3 points, which is described by the small lambda matrix

```
-  0  1
1  -  0*
0  1  -
```

The entry $\lambda_{23} = 0$, which is marked with a star (*), is the one entry which is actually encoded as "0x00" in the file. This file is also available in the folder `data/order_types/`.

The following command now enumerates all abstract order types on 4 points:

```
$ xxd n3_order_types.bin
00000000: 00                                       .
$ ./extend_order_type 3 n3_order_types.bin 1 0 1
n: 3
starting threads: 1
[0/1] started
[0/1] n 3       ct 0    extensions 0
[0/1] finished
all threads done.
total solutions: 2/1
$ xxd n3_order_types.bin.ext0_1.bin
00000000: 0001 0001 0001
```

Note that the command xxd displays a hex dump of the given file. The generated output file `n3_order_types.bin.ext0_1.bin` contains 6 bytes in total, encoding the two order types on 4 points (with 3 bytes each). The first three bytes encode the order type of 4 points in convex position, which has the following small lambda matrix:

```
-  0  1  2
2  -  0* 1*
1  2  -  0*
0  1  2  -
```

The remaining three bytes encode the other order type of 4 points, which has a triangular convex hull and one interior point. Its small lambda matrix is the following:

```
-  0  1  2
2  -  1* 0*
1  1  -  1*
0  2  1  -
```

When renaming the file `n3_order_types.bin.ext0_1.bin` to `n4_order_types.bin`, one can now analogously enumerate all order types of 5, 6, ... points with

```
./extend_order_type 4 n4_order_types.bin 1 0 1
./extend_order_type 5 n4_order_types.bin 1 0 1
...
```

A.2 Enumerating Triangulations

Plantri Having the graph generator plantri installed (available from `https://users.cecs.anu.edu.au/~bdm/plantri/`), it can be run with parameters "[number of points] -g", to enumerate all triangulations on the specified number of points in graph6 format. With the additional parameter "-c4", only 4-connected triangulations are enumerated. As an example, following command enumerates all 4-connected triangulations on 8 vertices:

```
$ ./plantri 8 -g -c4
./plantri 8 -g -c4
G|tJH{
G|thXs
2 triangulations written to stdout; cpu=0.00 sec
```

To store graphs in files, one can simply pipe the standard output to the desired file:

```
$ ./plantri 8 -g -c4 > n8c4.g6
./plantri 8 -g -c4
2 triangulations written to stdout; cpu=0.00 sec
```

For more information on plantri, we refer to `https://users.cecs.anu.edu.au/~bdm/plantri/plantri-guide.txt`, and for more information on the graph6 format, we refer to `https://users.cecs.anu.edu.au/~bdm/data/formats.html`.

Filter Triangulations Having mathematics software system SageMath installed (available from `http://www.sagemath.org/download.html`, see also `http://doc.sagemath.org/pdf/en/installation/installation.pdf`), we used the SageMath scripts `filter_3tree.sage` and `filter_maxdeg.sage` to filter stacked triangulations and triangulations with maximum degree $|V| - 1$, respectively.

The script `filter_3tree_relabel.sage` is a slight modification of `filter_3tree.sage`, which relabels the vertices of the given graph in a way, such that the vertices 0, 1, and 2 span the initial triangle, and the k-th vertex is stacked into a triangular face of the subgraph induced by the vertices $0, 1, \ldots, k-1$.

Note that the triangulations enumerated by plantri do not necessarily fulfill this property. The triangulations shown in Figures 2 and 3 (see also Listings 1 and 2) were relabed using `filter_3tree_relabel.sage`. The respective files are available in `data/triangulations/`.

Edge-List Encoding We have chosen a different plain text format, which is easier to load in C++: We encode a graph by its edge list. For each graph, we write the start and end vertices of the edges $\{u_1, v_1\}, \ldots, \{u_m, v_m\}$ simply as "$u_1\ v_1\ u_2\ v_2\ \ldots u_m\ v_m$" in a line, followed by a line-break. The following example gives an illustration (continues with the 4-connected 8-vertex triangulations from before):

```
$ sage encode.sage n8c4.g6
0 1 0 2 0 3 0 4 1 2 1 4 1 5 1 6 2 3 2 6 2 7 3 4 3 7 4 5 4 7 5 6 5 7 6 7
0 1 0 2 0 3 0 4 1 2 1 4 1 5 2 3 2 5 2 6 2 7 3 4 3 7 4 5 4 6 4 7 5 6 6 7
```

This encoding can be performed using the Sage-script `encode.sage`.

Drawing Graphs Last but not least, we provide the script `draw.sage` which we used to automatically generate drawings for stacked triangulations; see Figures 2 and 3. The idea is to start with a Tutte embedding and then use global optimization heuristics (see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`) to optimize a certain quality function, which simultaneously maximizes edge lengths and vertex-edge distances.

A.3 Testing n-Universality

test_universal_sets We provide a C++ program `test_universal_sets` to find n-universal point sets. The program (see the folder `cprogram/scripts/test_universal_sets/`) can be built analogously to `extend_order_type` (using qmake and make), except that Minisat is required to be build as a library first.

Building the Minisat Library As described in the README file delivered with Minisat (cf. `cprogram/minisat-2.2.0`, can also be downloaded from `http://minisat.se/MiniSat.html`), one needs to set the MROOT variable. This can be done for example with the following command:

```
$ export MROOT=$PWD
```

In the `simp` folder from Minisat, one then can run

```
$ make lib_release.a
```

to build the library. Having the `lib_release.a` built, we are now ready to build our program `test_universal_sets`. Note that, if minisat-2.2.0 is not placed inside the basis directory (where the `otlib.pro` is located), one might need to slightly adopt the project file `test_universal_sets.pro` so that the minisat headers and library are found. In particular, the following two lines might need to be adopted:

```
INCLUDEPATH += $$OTLIBDIR/minisat-2.2.0
LIBS += $$OTLIBDIR/minisat-2.2.0/simp/lib_release.a
```

Usage of the Program

```
./test_universal_sets [n] [order types file] [graphs file] [phase]
                      [parts] [from part] [to part]
```

The parameters can be described as follows:

- "n" is the number of points in the abstract order types from the input file,
- "order types file" is the path to the input file for abstract order types,
- "graphs file" is the path to the input file for graphs,
- "phase" specifies the actions which should be performed (see below),
- "parts" is the number of threads in total,
- "from part" is the id of the first thread to be run, and
- "to part" is the id of the first thread not to be run.

Abstract order types are again encoded by their small lambda matrix as described above. For the graph file we have chosen the plain-text edge-list format described above. The phase parameter describes, what the program should test:

- If the program is ran with parameter "phase=1", then point sets are filtered out which do not fulfill the necessary conditions described in Phase 1 of Section 4.
- If the program is ran with parameter "phase=2", then for each point set all graphs from the list are tested for simultaneous embeddability. As described in Phase 2 of Section 4, we stop as soon as one graph is not embeddable, and we use a priority queue to speedup the computations.

  The parameter "phase=2" is also used to test Phase 4 of Section 4.
- If the program is ran with parameter "phase=3", then all pairs of order types and graphs are tested for embeddability. Unlike for the computations of Phase 2, we do not change the order of the list of graphs. For each given order type from the input file, a line of zeros and ones is written to a plain-text output file, where the j-th zero/one in the i-th line encodes whether the j-th graph can be embedded on the i-th point set. In the following we refer to this file as "stat"-file.

  The parameter "phase=3" is also used to test Phase 5 of Section 4.

Note that for Phase 6 of Section 4 one can simply concatenate the stat-files obtained in Phases 3 and 5, and run CPLEX/Gurobi – no additional computations with our C++ program are necessary.

How To Load Realizations from the Order Type Database To load files from the order type database [Aic], which provide point set realizations of all order types, one simply needs to change the line

```
// #define REALIZATIONS
```

to

```
#define REALIZATIONS
```

in the source file `test_universal_sets.cpp`. Note that in the binary files `otypes04.b08`, ..., `otypes08.b08` (available at [Aic]) each order type of $n = 4, \ldots, 8$ points is encoded by one of its realizing point sets: the points $(x_1, y_1), \ldots, (x_n, y_n)$ are encoded as "$x_1 y_1 \ldots x_n y_n$" using 1 byte per coordinate (values inbetween 0 and 255). For $n = 9$, $n = 10$, and $n = 11$, each coordinate is encoded using 2 bytes (values inbetween 0 and 65536).

A.4 Integer Programming

Gurobi Having Gurobi/Gurobipy installed (see Section 12 "Python Interface" from `http://www.gurobi.com/documentation/8.1/quickstart_linux.pdf`), we have used the Python script `test_min_cover.py` to create a (Mixed) Integer Linear Programming instance from a stat-file (created from our C++ program). The script parses the input file, writes the instance to an "lp" file, and then starts the Gurobi solver to find an (optimal) solution. An instance, which is stored in an lp-file, can also be read and solved on a different machine for example via the following command:

```
$ gurobi
...
gurobi> m=read("n11_phase5_statistic_stacked.txt.instance.lp")
gurobi> m.optimize()
Optimize a model with 17533 rows, 423 columns and 1031205 nonzeros
...
Explored 295 nodes (25169 simplex iterations) in 69.54 seconds
Thread count was 6 (of 6 available processors)

Solution count 5: 12 13 14 ... 50

Optimal solution found (tolerance 1.00e-04)
Best objective 1.200000000000e+01, best bound 1.200000000000e+01, gap 0.0000%
```

It is worth to note that, when solving an instance using Gurobi (also with CPLEX), the current upper and lower bound on the optimal value is printed to the console every few seconds. Moreover, when aborting the solving process (CTRL+C), the currently best solution is printed.

For more information on free academic Gurobi versions, checkout `http://www.gurobi.com/academia/academia-center`.

CPLEX An instance, which is stored in an lp-file, can be read and solved via the CPLEX Interactive Optimizer as exemplified in the following:

```
$ cplex
...
CPLEX> read n11_phase5_statistic_stacked.txt.instance.lp
```

```
Problem 'n11_phase5_statistic_stacked.txt.instance.lp' read.
Read time = 0.17 sec. (14.61 ticks)
CPLEX> optimize
...
Root node processing (before b&c):
  Real time             =    8.02 sec. (3713.90 ticks)
Parallel b&c, 6 threads:
  Real time             =   28.38 sec. (4730.46 ticks)
  Sync time (average)   =    3.49 sec.
  Wait time (average)   =    0.01 sec.
                          ------------
Total (root+branch&cut) =   36.40 sec. (8444.36 ticks)

Solution pool: 4 solutions saved.

MIP - Integer optimal solution:  Objective =  1.2000000000e+01
Solution time =   36.40 sec.  Iterations = 35476  Nodes = 358
Deterministic time = 8444.36 ticks  (232.00 ticks/sec)
```

For more information on free academic CPLEX versions, checkout `http://www.ibm.com/developerworks/community/blogs/jfp/entry/CPLEX_Is_Free_For_Students`