Gray Code Enumeration of Plane Straight-Line Graphs*

O. Aichholzer†, F. Aurenhammer‡, C. Huemer§, B. Vogtenhuber¶

Abstract

We develop Gray code enumeration schemes for geometric straight-line graphs in the plane. The considered graph classes include plane graphs, connected plane graphs, and plane spanning trees. Previous results were restricted to the case where the underlying vertex set is in convex position.

1 Introduction

Let $E = \{e_1, \dots, e_m\}$ be an ordered set. For the purposes of this paper, E will consist of the $m = \binom{n}{2}$ line segments spanned by a set S of n points in the plane, in lexicographical order. Consider a collection $\mathcal{A}$ of subsets of E. For instance, think of $\mathcal{A}$ being the class of all plane spanning trees of S. We associate each member $A_i \in \mathcal{A}$ with its containment vector b_i with respect to E. That is, b_i is a binary string of length m whose j^{th} bit is 1 if $e_j \in A_i$ and 0, otherwise. A (combinatorial) *Gray code* for the class $\mathcal{A}$ is an ordering $A_1, \dots, A_t$ of $\mathcal{A}$ such that b_{i+1} differs from b_i by a transposition, for $i = 1, \dots, t-1$. For example (and as one of the results of this paper), for plane spanning trees a Gray code exists such that successive trees differ by a single edge move. Depending on the class we will consider, a transposition will be an exchange of two different bits (as for the spanning tree class) or/and a change of a single bit. Combinatorial Gray codes generalize the classical binary reflected Gray code scheme [16] for listing m-bit binary numbers so that successive numbers differ in exactly one bit position. See [17] for a survey article. We also refer to [3] for various results concerning edge moves in spanning trees.

Any Gray code for a given class $\mathcal{A}$ provides a complete enumeration scheme for $\mathcal{A}$ by means of constant-size operations. Listing all the objects of a given class is a fundamental problem in combinatorics and, in particular, in computational geometry. Not every enumeration scheme constitutes a Gray code, however, as a small difference between consecutive objects will not be guaranteed, in general. For instance, the popular reverse search enumeration technique [6] lacks this property. When interpreting $\{A_1, \dots, A_t\}$ as the set of nodes of an abstract graph that connects two nodes A_i and A_j whenever b_i and b_j are a single transposition apart, any Gray code for the class $\mathcal{A}$ corresponds to a Hamiltonian path in this transposition graph.

To see a simple example, let $\mathcal{A} = \mathcal{G}$, the class of all possible straight-line graphs on a given point set S. Each of the 2^m subsets of E defines a member of $\mathcal{G}$, for $m = \binom{n}{2}$. If we define a transposition to be a

*Research supported by the FWF Joint Research Project Industrial Geometry S9205-N12 and Projects MCYT-FEDER BFM2003-00368 and Gen. Cat 2005SGR00692

†Institute for Software Technology, University of Technology, Graz, Austria, oaich@ist.tugraz.at
‡Institute for Theoretical Computer Science, University of Technology, Graz, Austria, auren@igi.tugraz.at
§Departament de Matematica Aplicada, Universitat Politecnica de Catalunya, Barcelona, Spain, Huemer.Clemens@upc.edu
¶Institute for Software Technology, University of Technology, Graz, Austria, lambda@fsmat.at

change of a single bit (i.e., the insertion or removal of a single edge) then the transposition graph for $\mathcal{G}$ is the hypercube in m dimensions. The binary reflected $\binom{n}{2}$-bit Gray code (see Appendix) corresponds to a Hamiltonian cycle in this hypercube, and constitutes a combinatorial Gray code for $\mathcal{G}$.

While general enumeration schemes for geometric objects have been studied quite extensively, see e.g. [6, 2, 8, 7], respective results for Gray codes are sparse. In particular, all the known results for straight-line graphs concern the special case where the underlying set S of points is in convex position. Under this restriction, [5], [15], and [10], respectively, treat the classes of plane (i.e., crossing-free) straight-line graphs, of plane spanning paths, and of plane perfect matchings, [9, 13] study the class of triangulations and plane spanning trees, and [11, 12] the class of diagonal partitions. Many of these Gray code constructions are based on the same idea, namely, on a hierarchy of graphs structured by increasing point set cardinality. In this paper, we extend this approach to general point sets S, which becomes possible when combining it with classical combinatorial Gray codes. We construct Gray codes for the class $\mathcal{PG}$ of all plane straight-line graphs, the class $\mathcal{CPG}$ of all plane and connected straight-line graphs, and the class $\mathcal{ST}$ of all plane spanning trees, for a given point set. For the class $\mathcal{CPG}$ no results existed even for the convex case.

The respective challenging question for triangulations remains open. The only known result is that for triangulations on $n = 6$ points in nonconvex position a Gray code does not always exist; see Section 6.

1.1 A hierarchy for plane graphs

Let $S = \{p_1, \ldots, p_n\}$ be the underlying set of points in the plane. Without loss of generality, let S be given in sorted order of x-coordinates. For simplicity, we also assume that no three points in S are collinear. Then, for $1 \leq k \leq n-1$, the point p_{k+1} lies outside the convex hull of $\{p_1, \ldots, p_k\}$. This property will turn out useful in the subsequent constructions.

Let now $\mathcal{A}$ be one of the classes $\mathcal{PG}$, $\mathcal{CPG}$, or $\mathcal{ST}$. We will define a hierarchy (a tree structure) $H_{\mathcal{A}}(S)$ for $\mathcal{A}$ and S such that the k^{th} level of the hierarchy consists of all the members of $\mathcal{A}$ on top of the first k points in S, for $k = 1, \ldots, n$. That is, each member in $H_{\mathcal{A}}(S)$ except the root (at level 1) has a unique parent, and each member in $H_{\mathcal{A}}(S)$ which is not a leaf has a unique set of children.

The Gray code construction for $\mathcal{A}$ is done recursively. It hinges on an appropriate rule for defining the parent of a given member, as well as on a consistent rule for enumerating its children. Designing the enumeration rule for the children is the crucial part, as it has to yield a Gray code for the children which fits the (previously constructed) Gray code for the parents. In particular, $H_{\mathcal{A}}(S)$ has to be an ordered tree such that the ordering at each of its levels is a Gray code.

2 The class $\mathcal{PG}$

For the class $\mathcal{PG}$ of all plane straight-line graphs, things are surprisingly simple. Let, for this class, a transposition be the change of a single bit, that is, the insertion or removal of a single edge. Let G' be at level $k+1$ of the hierarchy $H_{\mathcal{PG}}(S)$, for $k \geq 1$. That is, G' is some plane straight-line graph whose vertex set is $\{p_1, \ldots, p_{k+1}\}$.

The parent of G' is obtained by removing from G' the vertex p_{k+1} and all its incident edges. This gives a unique graph G at level k of $H_{\mathcal{PG}}(S)$. We say that a vertex p_j of G is *visible* (from p_{k+1}) if the line segment $p_{k+1}p_j$ does not cross any edge of G. As we are interested only in plane graphs, all the children of G are obtained by adding the vertex p_{k+1} and connecting p_{k+1} to subsets of visible points in all possible ways (including the empty set).

Let v_G be the number of visible vertices of G. We can use the cyclic binary v_G-bit Gray code $B(v_G)$ (see Appendix) for encoding all the possible subsets of edges incident to p_{k+1}. A transposition in this code

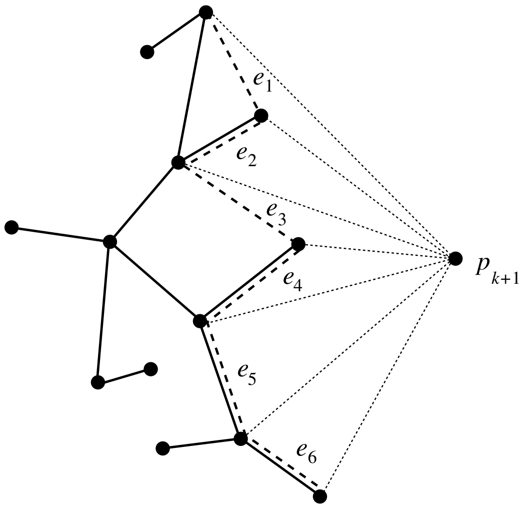

Figure 1: The chain of a spanning tree

then means adding or removing a single edge to a visible vertex. To specify the first and the last subset (i.e., child of G), we take $\emptyset$ for the first child and the singleton set $\{p_{k+1}p_j\}$ for the last child, where $p_{k+1}p_j$ is the edge tangent to the convex hull of G from above. (Vertex p_j is visible for any choice of the graph G. In particular, p_j is the first visible vertex in counter-clockwise order.) Accordingly, the first string in the code $B(v_G)$ is $00\ldots0$ and the last string is $10\ldots0$.

To construct a Gray code for level $k+1$ of $H_{\mathcal{PG}}(S)$, let G and F be adjacent at level k of $H_{\mathcal{PG}}(S)$. Attaching $00\ldots0$ to the string for G and F, respectively, leaves the strings one transposition apart. The same is true for $10\ldots0$. That is, the first child of G is adjacent to the first child of F, and the last child of G is adjacent to the last child of F. So we can run $B(v_G)$ followed by $\overline{B(v_F)}$ (the code $B(v_F)$ in reverse order), and so on. The construction of the desired Gray code is now obvious. In addition, we can use the fact that the number of plane straight-line graphs on $n \geq 2$ points is even (because each convex hull edge appears in exactly half of the graphs), which by induction implies that the Gray code is cyclic. In the language of graph theory, our result reads:

Theorem 1 *The transposition graph for $\mathcal{PG}$ contains a Hamiltonian cycle.*

It would be interesting to know the average number of visible vertices for a level-k member of $H_{\mathcal{PG}}(S)$. This information could be used to give bounds on the number of plane straight-line graphs.

3 Plane spanning trees

Our strategy for constructing Gray codes also works for the class $\mathcal{ST}$ of plane spanning trees. A transposition will now be an exchange of two different bits, because the addition (or removal) of a single edge destroys the tree property.

Consider a member T' at level $k+1$ of the hierarchy $H_{\mathcal{ST}}(S)$, for $k \geq 1$. That is, T' is some plane spanning tree on $\{p_1, \ldots, p_{k+1}\}$. Defining an appropriate parent of T' is less trivial now.

For an arbitrary plane straight-line graph G on $\{p_1, \ldots, p_k\}$, let $q_1, \ldots, q_v$ be the visible vertices of G (as seen from p_{k+1}) in counter-clockwise order. We define the *chain for G*, $C(G)$, as the ordered set of line segments $q_i q_{i+1}$, for $1 \leq i \leq v-1$. Observe that for every edge $e \in C(G) \setminus G$, the set of visible vertices of $G \cup \{e\}$ is the same as for G.

Parent rule: Let G be the graph obtained by removing from T' the vertex p_{k+1} and all its incident edges. Let G consist of $r \geq 1$ components. We add the first $r-1$ edges of the chain $C(G)$ that connect these components, and define the resulting tree as the parent of T'.

This rule yields a unique parent for T'. Notice that this parent is well-defined (i.e., belongs to level k of $H_{\mathcal{ST}}(S)$) because G can always be connected to a tree using edges of $C(G)$, and no such edge crosses any edge of G. From the definition of the parent we get the definition for the children, as follows.

Let T be a tree at level k of the hierarchy $H_{\mathcal{ST}}(S)$. For two edges $e, e' \in C(T)$, write $e' \prec e$ if e' occurs before e in the total order on the chain $C(T)$. We define $E(T)$ as the set of all edges $e \in T \cap C(T)$ such that

(1) removal of e does not make a non-visible vertex of T visible, and

(2) no edge $e' \in C(T) \setminus T$ with $e' \prec e$ gives a cycle in $T \cup \{e'\}$ that contains e.

See Figure 1. The spanning tree T is drawn with bold lines. Its chain $C(T)$ consists of six edges $e_1, \ldots, e_6$. Edges e_1 and e_3 are not part of T and thus do not belong to $E(T)$. Also, we have $e_4, e_5 \notin E(T)$ because these edges reveal visible points. Finally, $e_2 \notin E(T)$ as the edge $e_1 \prec e_2$ closes a cycle in T that contains e_2. This gives $E(T) = \{e_6\}$.

Children rule: The children of T are obtained by removing, for each possible subset of $E(T)$, its edges from T and connecting each resulting component to the vertex p_{k+1} using a single edge to some visible vertex, in all possible ways.

It is clear that all the graphs constructed from T in this way are plane spanning trees on $\{p_1, \ldots, p_{k+1}\}$. To see that each member of level $k+1$ of the hierarchy $H_{\mathcal{ST}}(S)$ is generated exactly once by the children rule (provided all the members of level k have so), we need the lemma below. Let us color the edges of $E(T)$ *green*, and the edges that connect to p_{k+1} *red*.

Lemma 1 *The parent rule and the children rule are consistent.*

Proof. Child $\rightarrow$ parent: Let T' be some child constructed from T. Then T' contains $r \geq 1$ red edges. If $r = 1$ then no green edge has been removed from T for constructing T'. According to the parent rule, we now remove from T' the vertex p_{k+1}, which correctly gives us the single component T. Now assume $r \geq 2$. Then $r-1$ green edges have been removed from T to construct T', leaving r components. Let K_i and K_{i+1} be two of these components, such that there is a single (removed) green edge e between them. There is no edge $e' \in C(T) \setminus T$ with $e' \prec e$ and which gives a cycle in $T \cup \{e'\}$ containing e. Thus, when removing from T' the vertex p_{k+1} and joining components of the resulting graph G according to the parent rule, the edge e is indeed the first edge of $C(G)$ that connects K_i and K_{i+1}. (Otherwise we would have chosen e' for removal instead of e.) Thus we correctly get T from T' again.

Parent $\rightarrow$ child: Let T be the parent constructed from T'. T is obtained by removing p_{k+1} and joining the r components of the resulting graph G with edges of $C(G)$. Let e be such an edge. Then e is the first edge of $C(G)$ that connects the respective two components. Therefore, no edge $e' \in C(T) \setminus T$ with $e' \prec e$ gives a cycle in $T \cup \{e'\}$ that contains e. Moreover, because $e \in C(G) \setminus G$, we know that $G \cup \{e\}$ and G have the same set of visible vertices. In conclusion, e is indeed a green edge. Thus, by the children rule, T' will be constructed as one of the children of T. $\square$

We are now ready to prove:

Theorem 2 *The transposition graph for $\mathcal{ST}$ contains a Hamiltonian path.*

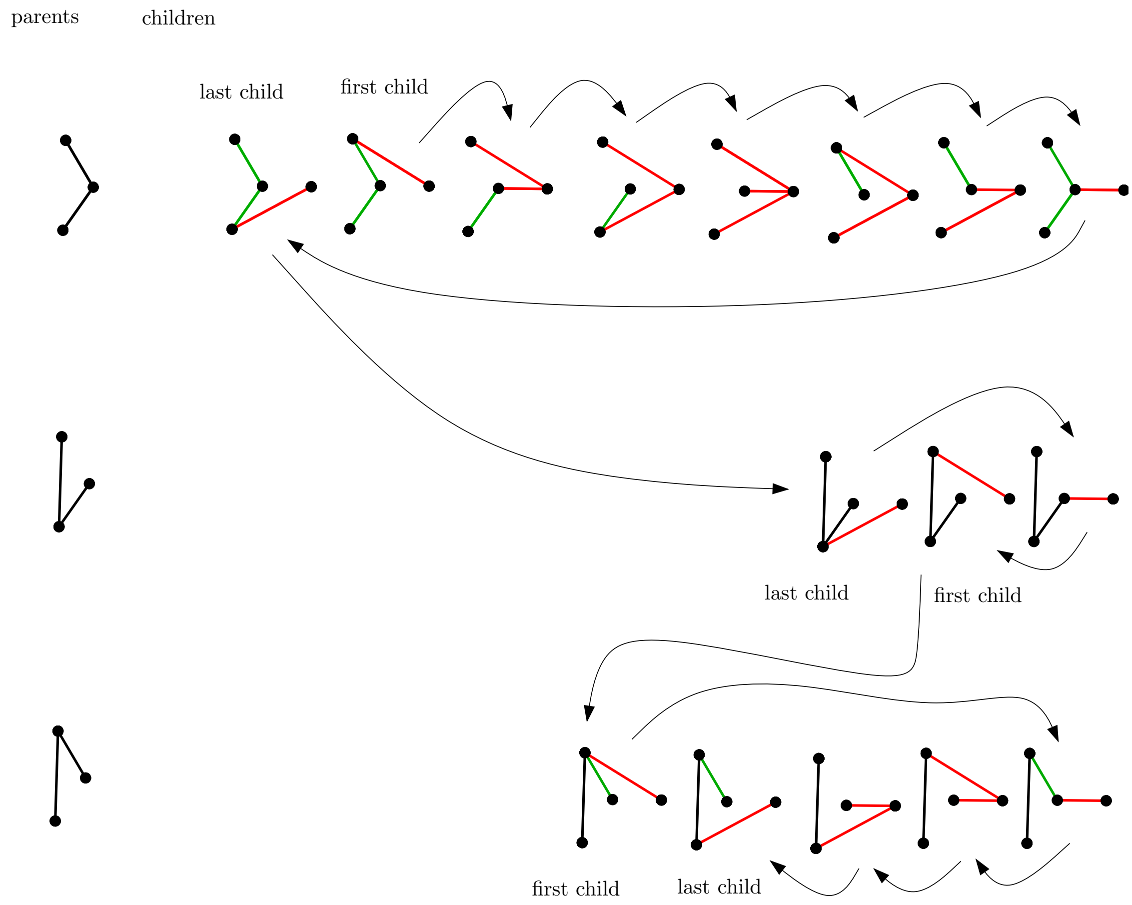

Figure 2: Gray code construction for spanning trees

Proof. Let T be a tree at level k. Define as the first child T^f (respectively, as the last child T^ℓ) of T the tree obtained when adding to T the upper (respectively, lower) tangent from p_{k+1} to the convex hull of T. Note that T^f and T^ℓ are well-defined children of T. We claim (and prove below) that the transposition graph for $\mathcal{ST}$ contains a path from T^f to T^ℓ that contains all the children of T. The theorem follows because, for two neighboring trees T and U at level k, their first children T^f and U^f, as well as their last children T^ℓ and U^ℓ, are a single transposition apart.

To encode all the children of T, we have to consider each subset $Y \subseteq E(T)$ of green edges and, for fixed Y, all allowed distributions of red edges. Let $g = |E(T)|$. Similar to Section 2, the binary reflected g-bit Gray code $B(g)$ (let us call it the *green* code) is used for encoding all possible subsets of $E(T)$. Note that $E(T) = \emptyset$ is possible. Now consider a fixed subset $Y \subseteq E(T)$. Let $T \setminus Y$ have $r \geq 1$ components. Given an arbitrary visible vertex q_j in each component K_j, $1 \leq j \leq r$, there exists a Gray code $R(r)$ (the *red* code) that starts with the positions of the edges $q_1 p_{k+1}, \ldots, q_r p_{k+1}$ and that encodes all allowed positions of the red edges; see the Appendix.

Between every two transpositions in the green code we work off the red code. Care has to be taken when switching between the codes. If a green edge is added (i.e., two components are joined) then some red edge has to be removed. All other red edges stay at their positions, which are the starting positions for the subsequent red code. Similarly, if a green edge is removed (i.e., a component is split into two) then some red edge has to be added. Again, all other red edges stay at their positions which are the starting positions for the next red code.

When combining the green and the red code we obtain a Gray code for all the children of T. However, this code will not end with the last child T^{ℓ} of T, in general. To complete the construction, we proceed as follows.

The green code is started at the subset $Y = \emptyset$ of $E(T)$. (Recall that T^f and T^{ℓ} arise as children for this subset.) As removing $\emptyset$ from T leaves the single component $T = K_1$, the corresponding red code is just $R(1)$. We start $R(1)$ with a red edge e whose choice depends on the number v_1 of visible vertices of K_1. If $v_1 > 2$ then e is chosen to be not a convex hull tangent else e is chosen to be the upper tangent (which corresponds to T^f). Now, before running $R(1)$, we perform one transposition in the green code and then run the combined code until having returned to $\emptyset$. We re-enter $R(1)$ with an edge e' different from e, which is guaranteed when the appropriate red edge is removed when inserting the green edge which yields the component K_1. Finally, $R(1)$ is traversed from e' to e in any order but so that T^f and T^{ℓ} are adjacent. For $v_1 > 2$ this is possible because any two strings in $R(1)$ differ by a single transposition. On the other hand, for $v_1 = 2$ the code $R(1)$ only contains the two strings for e and e', which then correspond to T^f and T^{ℓ}.

A cyclic code for the children of T is obtained in which T^f and T^{ℓ} are adjacent. Cutting this code between these two trees gives the desired result. □

If the Hamiltonian path in Theorem 2 both starts and ends with a first child (or with a last child), then we get a Hamiltonian cycle. Constructing such a cycle in the general case is left to further research. Figure 2 illustrates our Gray code construction for a set of 4 points in nonconvex position. The inserted point $p_{k+1} = p_4$ corresponds to the rightmost vertex.

4 Connected plane graphs

The Gray code construction for the class $\mathcal{ST}$ can be modified to yield a Gray code for the class $\mathcal{CPG}$ of all connected plane straight-line graphs. Allowed transpositions will now be an exchange of two different bits or a change of a single bit. Disallowing either of these types makes the transposition graph for $\mathcal{CPG}$ non-Hamiltonian.

The parent rule needs no modification. Concerning the children rule, let G be a graph at level k of the hierarchy $H_{\mathcal{CPG}}(S)$. We slightly redefine the set $E(G)$, namely, as the set of all edges $e \in G \cap C(G)$ such that

(1) removal of e disconnects G but makes no non-visible vertex of G visible, and

(2) no edge $e' \in C(G) \setminus G$ with $e' \prec e$ gives a cycle in $G \cup \{e'\}$ that contains e.

Children rule: The children of G are obtained by removing, for each possible subset of $E(G)$, its edges from G and connecting each resulting component to the vertex p_{k+1} with a (nonempty) set of edges to visible vertices, in all possible ways.

The proof of consistency of the parent rule and the children rule is similar to Lemma 1; we leave it to the interested reader. As before, let us color the edges in the set $E(G)$ green and the edges that connect to p_{k+1} red.

Theorem 3 *The transposition graph for $\mathcal{CPG}$ contains a Hamiltonian path.*

Proof. We mimic the proof for spanning trees (Theorem 2). The green code stays unchanged, whereas the red code has to be modified. Let G be the level-k member of the hierarchy $H_{\mathcal{CPG}}(S)$ whose children are to be encoded, and let $Y \subseteq E(G)$ be the current subset of green edges. For each component K_j of $G \setminus Y$ we

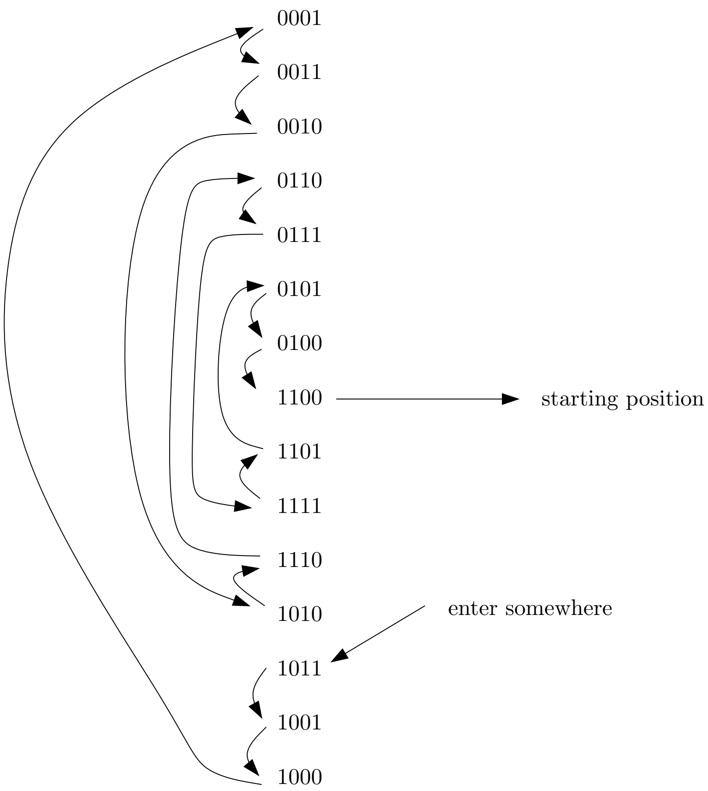

Figure 3: Traversing the code $M(1)$

have to encode all possible nonempty subsets of visible vertices (i.e., of red edges). A modified red code M achieving this is given in the Appendix. This code exists for every fixed starting position of red edges.

When switching between the codes, if a green edge is removed (i.e., a component is split into two) then some red edge is added to keep the graph connected. All other red edges are held fixed. Conversely, if a green edge is added (i.e., two components are joined) then either all red edges are held fixed or one of them is removed. This gives the starting positions of the red code for the subsequent subset of green edges. When combining the green and the red code we proceed as follows.

The green code is started with removing from G the subset $Y = \emptyset$. The modified red code for the resulting single component $K_1 = G$ is just $M(1)$, that is, basically a binary reflected Gray code. Let $b_1, \ldots, b_k$ be the strings in $M(1)$. We have $k \geq 3$ and k odd. Recall that $b_1 = G^\ell$ and $b_k = G^f$, and put $m = \frac{k+1}{2}$. The string $b_m = 110\ldots0$ is the string with lowest index whose leading bit is 1. We take b_m as the starting position for $M(1)$. Now, before running $M(1)$, we do one transposition in the green code, and then run the combined code until the subset $\emptyset$ is reached again. Let b_r be the string at which $M(1)$ is re-entered. That is, b_r describes the final positions of the red edges for the last subset Y in the green code. (Note that $Y = \{e\}$ is a singleton set; adding e forms the component $K_1 = G$.) To complete the code construction, we need a traversal of $M(1)$ that starts with b_r and ends with b_m, where b_1 and b_k are consecutive, and where any two consecutive strings are a single transposition apart.

Case $r > m$. See Figure 3. We start the traversal with the strings $b_r \ldots b_k b_1 \ldots b_{k-r}$ and continue with the strings $b_{k-r+1} b_{r-1} b_{r-2} b_{k-r+2} b_{k-r+3} b_{r-3} \ldots$ and so on, until b_m is reached. Observe that b_1 and b_k differ by an exchange of two bits, and that b_{r-i} and b_{k-r+i} for fixed i differ only in the leading bit. (If r is even then the string $b_{m-1} = 010\dots0$ is left out in this sequence. But b_{m-1} can always be put between two suitable strings.)

Case $r < m$. The traversal is symmetric, namely the strings $b_r \dots b_1 b_k \dots b_{k-r}$ followed by the strings $b_{k-r-1} b_{r+1} b_{r+2} b_{k-r-2} b_{k-r-3} b_{r+3} \dots$ and so on, until b_m.

Case $r = m$. In this case, we do not re-enter $M(1)$ at $b_r = b_m$, but at its neighbor $b_{m-1} = 010\dots0$. This corresponds to removing some red edge when the green edge e (defined above) is added. That is, an exchange of two bits takes place. The desired traversal is $b_{m-1} b_{m-2} \dots b_1 b_k b_{k-1} \dots b_m$.

In conclusion, a cyclic code for the children of G is obtained where the first child G^f and the last child G^ℓ are adjacent. □

5 Algorithmic issues

To perform a Gray code enumeration of a given graph class $\mathcal{A}$, the hierarchy $H_\mathcal{A}(S)$ is traversed in preorder and its leaves are reported. For $\mathcal{A} \in \{\mathcal{PG}, \mathcal{ST}, \mathcal{CPG}\}$, each non-leaf member of $H_\mathcal{A}(S)$ has at least two children. Consequently, the time complexity is dominated by computing the leaves. When computing the children of a given parent G, the main tasks are calculating the chain $C(G)$ and the set $E(G)$ of green edges. (These tasks are void for the class $\mathcal{PG}$.) Both tasks can be accomplished in time linear in the size of G, as is shown below.

Let L be any connected plane straight-line graph. The *faces* of L are maximal connected subsets of the complement of L (in a geometric sense) in its convex hull. To *walk* around a face F of L means traversing those edges of L, and of the convex hull of L, that bound F. Edges that bound F from both sides are traversed twice.

Now, let the parent G live on the points $\{p_1, \dots, p_k\}$. For computing $C(G)$, we define the graph L as the union of G with the two tangent edges from p_{k+1} to G. There is a unique face F of L incident to p_{k+1}. Starting from p_{k+1}, we walk around F and update visible vertices. That is, we check their containment in the currently spanned angle as seen from p_{k+1} using a stack.

For calculating $E(G)$, we define $L = G \cup C(G)$. We color the edges $e \in C(G)$ as *black* or *white* depending on whether $e \in G$ or not. Let us first handle the case for the class $\mathcal{ST}$ where G is a plane spanning tree.

Initially, $E(G)$ contains all black edges. We walk around each face F of L incident to $C(G)$, except the face incident to p_{k+1}. To test property (1) (visibility change) we check containment of each vertex x of F in the visibility angles $\alpha(e)$ spanned by the black edges e of F. If $x \in \alpha(e)$ then we delete e from $E(G)$. To test property (2) (induced cycle) we observe that F has at most one white edge e'. (G would be disconnected, otherwise.) If e' exists and F contains no edge of the convex hull of G, then we delete from $E(G)$ all black edges e of F with $e > e'$. This leaves the correct set $E(G)$.

If $\mathcal{CPG}$ is the considered class then property (1) also requires a test for connectedness. But removal of *no* black edge e of F disconnects G if and only if all edges of F are part of G. So all black edges of F have to be deleted from $E(G)$ in the affirmative case.

We may (inductively) assume that G is given in sorted adjacency list representation. So the graphs L and all their faces can be computed from G in $O(k)$ time. For computing $C(G)$ and $E(G)$, each such face is considered at most once. We conclude:

Theorem 4 *Let $\mathcal{A}$ be any of the classes $\mathcal{PG}, \mathcal{ST}, \mathcal{CPG}$, for a given set S of n points in the plane. A Gray code enumeration of $\mathcal{A}$ can be performed in $O(n)$ time per member.*

For the class $\mathcal{ST}$ this result is superior to the result for (non-Gray code) enumeration in [6], where $O(n^3)$ time per tree is needed. Note that the average number of children of a member is only a constant, as there are $O(c^n)$ plane straight-line graphs on S, for c depending on the class but not on n; see [4]. Thus, on the average, $O(n)$ time has to be invested for constructing each child.

6 Discussion

We have shown that, for several classes of plane straight-line graphs, their transposition graph is Hamiltonian. To see some negative examples, consider the class of all triangulations of the point set S, and let a transposition be an edge flip [14]. The simple 6-point example in Figure 4 shows that the corresponding flip graph is not Hamiltonian, in general. However, it still may be true that this property is attained for sufficently large point sets.

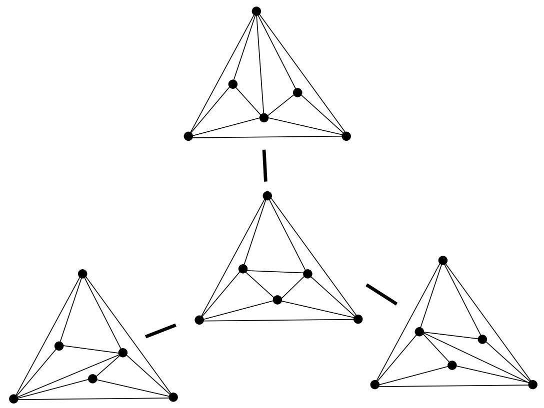

Figure 4: Transposition graph for triangulations

For pseudo-triangulations, see e.g. [1], the situation is unclear as well. If only edge-removing (respectively, edge-inserting) flips are allowed then the flip graph is not Hamiltonian. Figure 5 gives a counterexample. Our conjecture is that the flip graph becomes Hamiltonian when both edge-exchanging *and* edge-removing flips are admitted. The flip graph of minimum (or pointed) pseudo-triangulations is Hamiltonian for all sets of up to 5 points. (Only edge-exchanging flips are allowed in order to stay within this class.) We believe this to be true for general point sets. Our hierarchical approach possibly may be used to settle these problems. Let us address two more questions we find interesting.

With small adaptions, our enumeration scheme also works for point sets containing collinear points. Thus we can provide Gray codes for the respective graph classes on the grid.

The efficient generation of random spanning trees, plane graphs, or connected plane graphs is still an open problem. Progress can possibly be made by selecting random children for members in the respective classes.

7 Appendix

The binary reflected Gray code

The binary reflected Gray code $B(n)$ for n-bit numbers is defined recursively as follows. $B(1)$ is a list of two one-bit strings, namely $0, 1$. For $n \geq 2$, $B(n)$ is formed by taking the list for $B(n-1)$, prepending

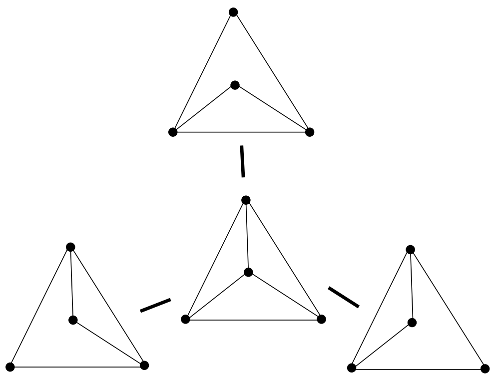

Figure 5: Graph for pseudo-triangulations and removing flips

the bit 0 to every string, and then taking the list for $B(n-1)$ in reversed order and prepending the bit 1 to every string. That is,

$$B(n) = \begin{cases} 0, 1 & \text{if } n = 1 \\ 0 \cdot B(n-1) \circ 1 \cdot \overline{B(n-1)} & \text{if } n \geq 2 \end{cases}$$

where $\cdot$ and $\circ$ denote character and list concatenation, respectively.

The red Gray code

For a given set Y of green edges, let the graph $T \setminus Y$ consist of the components $K_1, \ldots, K_r$. Let component K_j have v_j visible vertices, and let $1, \ldots, v_j$ be any fixed ordering of these vertices. The red code $R(1)$ for the first component K_1 is given by $1, 2, \ldots, v_1$. That is, $R(1)$ consists of v_1 strings of length v_1 each having a single bit 1. Assume we are given a red code $R(s)$ for $K_1, \ldots, K_s$, for $s < r$. Then the red code $R(s+1)$ is given by

$$1 \cdot R(s) \circ 2 \cdot \overline{R(s)} \circ 3 \cdot R(s) \circ \ldots \circ v_{s+1} \cdot \overline{R(s)}$$

where the last sublist may be $v_{s+1} \cdot R(s)$, according to the parity of v_{s+1}.

The modified red Gray code

As above, let the graph $G \setminus Y$ consist of the components $K_1, \ldots, K_r$ where K_j has v_j visible vertices. Define B_j as the binary reflected v_j-bit Gray code $B(v_j)$, but without the string $00\ldots0$. (The set of red edges connecting K_j to p_{k+1} has to be nonempty.) B_j is still cyclic because an exchange of two bits is an allowed transposition. Now number the strings in B_j consecutively from 1 to $w_j = 2^{v_j} - 1$, starting from an arbitrary string. The modified red code $M(1)$ for the first component K_1 is given by $1, 2, \ldots, w_1$. The construction for $M(s+1)$ proceeds along the same scheme as for the red code R above.

References

[1] O. Aichholzer, F. Aurenhammer, P. Braß, H. Krasser. *Pseudo-triangulations from surfaces and a novel type of edge flip*. SIAM Journal on Computing 32 (2003), 1621-1653.

[2] O. Aichholzer, F. Aurenhammer, H. Krasser. *Enumerating order types for small point sets with applications*. Order 19 (2002), 265-281.

[3] O. Aichholzer, F. Aurenhammer, F. Hurtado. *Sequences of spanning trees and a fixed tree theorem*. Computational Geometry: Theory and Applications 21 (2002), 3-20.

[4] M. Ajtai, V. Chvátal, M.M. Newborn, E. Szemerédi. *Crossing-free subgraphs*. Annals of Discrete Mathematics 12 (1982), 9-12.

[5] R. Arenas, J. Gonzalez, A. Marquez, M. Puertas Gonzalez. *Grafo de Grafos Planos de un Poligono Convexo*. Jornadas de Matematica Discreta y Algoritmica 4 (2004), 31-38.

[6] D. Avis, K. Fukuda, *Reverse search for enumeration*, Discrete Applied Mathematics 65 (1996), 21-46.

[7] S. Bereg. *Enumerating pseudo-triangulations in the plane*. Computational Geometry: Theory and Applications 30 (2005), 207-222.

[8] S. Felsner. *On the number of arrangements of pseudolines*, Discrete & Computational Geometry 18 (1997), 257–267.

[9] M. C. Hernando, F. Hurtado, A. Marquez, M. Mora, M. Noy. *Geometric tree graphs of points in convex position*. Discrete Applied Mathematics 93 (1999), 51-66.

[10] M. C. Hernando, F. Hurtado, M. Noy. *Graph of non-crossing perfect matchings*. Graphs and Combinatorics 18 (2002), 517-532.

[11] C. Huemer, F. Hurtado, M. Noy, E. Omana-Pulido. *Gray codes for non-crossing partitions and dissections of a convex polygon*. In: Proc. X Encuentros de Geometria Computacional, Sevilla, 2003, 20-23.

[12] C. Huemer, F. Hurtado, J. Pfeifle. *Gray codes and polytopal complexes for dissections of a polygon into k-gons*. In: Proc. XI Encuentros de Geometria Computacional, Santander, 2005, 31-38.

[13] F. Hurtado, M. Noy. *Graph of triangulations of a convex polygon and tree of triangulations*. Computational Geometry: Theory and Applications 13 (1999), 179-188.

[14] F. Hurtado, M. Noy, J. Urrutia. *Flipping edges in triangulations*. Discrete & Computational Geometry 22 (1999), 333-346.

[15] E. Rivera-Campo, V. Urrutia-Galicia. *Hamilton cycles in the path graph of a set of points in convex position*. Computational Geometry: Theory and Applications 18 (2001), 65-72.

[16] F. Ruskey. *Simple combinatorial Gray codes constructed by reversing sublists*. Springer Lecture Notes in Computer Science 762 (1993), 201-208.

[17] C. Savage. *A survey of combinatorial Gray codes*. SIAM Review 39 (1997), 605-629.