Efficient generation of elimination trees and graph associahedra*

Jean Cardinal† Arturo Merino‡ Torsten Mütze§

Abstract

An elimination tree for a connected graph G is a rooted tree on the vertices of G obtained by choosing a root x and recursing on the connected components of $G - x$ to produce the subtrees of x. Elimination trees appear in many guises in computer science and discrete mathematics, and they encode many interesting combinatorial objects, such as bitstrings, permutations and binary trees. We apply the recent Hartung-Hoang-Mütze-Williams combinatorial generation framework to elimination trees, and prove that all elimination trees for a chordal graph G can be generated by tree rotations using a simple greedy algorithm. This yields a short proof for the existence of Hamilton paths on graph associahedra of chordal graphs. Graph associahedra are a general class of high-dimensional polytopes introduced by Carr, Devadoss, and Postnikov, whose vertices correspond to elimination trees and whose edges correspond to tree rotations. As special cases of our results, we recover several classical Gray codes for bitstrings, permutations and binary trees, and we obtain a new Gray code for partial permutations. Our algorithm for generating all elimination trees for a chordal graph G can be implemented in time $\mathcal{O}(m+n)$ per generated elimination tree, where m and n are the number of edges and vertices of G, respectively. If G is a tree, we improve this to a loopless algorithm running in time $\mathcal{O}(1)$ per generated elimination tree. We also prove that our algorithm produces a Hamilton cycle on the graph associahedron of G, rather than just Hamilton path, if the graph G is chordal and 2-connected. Moreover, our algorithm characterizes chordality, i.e., it computes a Hamilton path on the graph associahedron of G if and only if G is chordal.

1 Introduction

Many recent developments in theoretical computer science and combinatorics are closely intertwined. Specifically, many combinatorial questions are motivated by applications to algorithm design, data structures, or network analysis. Conversely, most fundamental computational problems involve finite classes of combinatorial objects, such as relations, graphs, or words, and their analysis is a major drive for the development of combinatorial insights. There are four recurring fundamental algorithmic tasks that we wish to perform with such objects, namely to *count* them or to *sample* one of them at random, to *search* for an object that maximizes some objective function (combinatorial optimization), or to produce an exhaustive *list* of all the objects. A great deal of literature is devoted to all of these problems, and in this paper we focus on the last and most fine-grained of these tasks, namely *combinatorial generation*.

1.1 Combinatorial generation. The complexity of a combinatorial generation algorithm is typically measured as the time it takes to produce the next object in the list from the previous one. Clearly, the best we can hope for is that each object is produced in constant time. For this to be possible, any two consecutive objects should not differ much, so that the algorithm can perform the required modification in constant time. Such a listing of objects subject to some closeness condition is referred to as a *Gray code* [Sav97, Rus16]. For some applications a cyclic Gray code is desirable, i.e., the last object in the list and the first one also satisfy the closeness condition.

For example, the classical *binary reflected Gray code* [Gra53] is a listing of all bitstrings of length n such that each string differs from the previous one in a single bit, and this listing is cyclic.

*The full version of the paper can be accessed at `https://arxiv.org/abs/2106.16204`

†Computer Science Department, Université Libre de Bruxelles (ULB), Belgium.

‡Department of Mathematics, TU Berlin, Germany.

§Department of Computer Science, University of Warwick, United Kingdom

Arturo Merino was supported by ANID Becas Chile 2019-72200522. Torsten Mütze is also affiliated with the Faculty of Mathematics and Physics, Charles University Prague, Czech Republic, and he was supported by Czech Science Foundation grant GA 19-08554S. Arturo Merino and Torsten Mütze were also supported by German Science Foundation grant 413902284.

Another example is the problem of listing all permutations of length n such that every permutation is obtained from the previous one by an adjacent transposition, i.e., by swapping two neighboring entries of the permutation. This is achieved by the well-known *Steinhaus-Johnson-Trotter algorithm* [Tro62, Joh63, Ste64], which guarantees a cyclic listing; see Table 1.

A third classical example is the Gray code by Lucas, Roelants van Baronaigien, and Ruskey [LRR93] which generates all n-vertex binary trees by rotations, albeit non-cyclically. Binary trees are in bijection with many other Catalan objects such as triangulations of a convex polygon, well-formed parenthesis expressions, Dyck paths, etc. [Sta15]. In triangulations of a convex polygon, the rotation operation maps to another simple operation, known as a flip, which removes the diagonal of a convex quadrilateral formed by two triangles and replaces it by the other diagonal.

Combinatorial generation algorithms have been devised for many other classes of objects [Ehr73, Kay76, CRS+00], including objects derived from graphs and orders, such as spanning trees of a graph [Smi97], maximal cliques or independent sets of a graph [TIAS77], perfect matchings of bipartite graphs [FM94], perfect elimination orderings of chordal graphs [CIRS03], and linear extensions [VR81, PR94] or ideals of partial orders [Ste86].

A standard reference on combinatorial generation is Volume 4A of Knuth's series 'The Art of Computer Programming' [Knu11]. Generic methods have been proposed, such as the reverse-search technique of Avis and Fukuda [AF96], the ECO framework of Barcucci, Del Lungo, Pergola, and Pinzani [BDPP99], the antimatroid formulation of Pruesse and Ruskey [PR93], Li and Sawada's reflectable languages [LS09], the bubble language framework of Ruskey, Sawada, and Williams [RSW12], and Williams' greedy algorithm [Wil13].

$n=1$	$n=2$	$n=3$	$n=4$
1	12	123	1234
			1243
			1423
			4123
		132	4132
			1432
			1342
			1324
		312	3124
			3142
			3412
			4312
	21	321	4321
			3421
			3241
			3214
		231	2314
			2341
			2431
			4231
		213	4213
			2413
			2143
			2134

Table 1: The Steinhaus-Johnson-Trotter Gray code for permutations.

1.2 Flip graphs and polytopes. Given any class of combinatorial objects and a 'local change' operation between them, the corresponding *flip graph* has as vertices the combinatorial objects, and its edges connect pairs of objects that differ by the prescribed change operation. Partial orders and lattices are often lurking, such as the Boolean lattice for bitstrings, the weak Bruhat order on permutations, and the Tamari lattice for Catalan families. Moreover, flip graphs can often be realized as the skeletons of polytopes, and combinatorial generation for such classes of objects therefore amounts to computing Hamilton paths or cycles on this polytope. The polytopes associated with the three aforementioned examples of bitstrings, permutations, and binary trees are the hypercube, the permutahedron, and the associahedron, respectively; the latter two are shown in Figure 1. The associahedron, in particular, has a rich history and literature, connecting computer science, combinatorics, algebra, and topology [STT88, Lod04, HL07, Pou14].

1.3 Elimination trees. In this work, we focus on the generation of elimination trees, which are trees on n vertices that are obtained from a fixed graph G on n vertices, and which capture all ways of removing the vertices of G one after the other. For any graph G and any set of vertices X we write $G - X$ for the graph obtained by removing every vertex of X from G. For a singleton $X = \{x\}$ we simply write $G - \{x\} =: G - x$.

Given a connected graph $G = (V, E)$, an *elimination tree* for G is a rooted tree with vertex set V, composed of a root $x \in V$ that has as children elimination trees for each connected component of $G - x$. This definition is illustrated in Figure 2. An *elimination forest* for a graph G is a set of elimination trees, one for each connected component of G. We write $\mathcal{E}(G)$ for the set of all elimination forests for G.

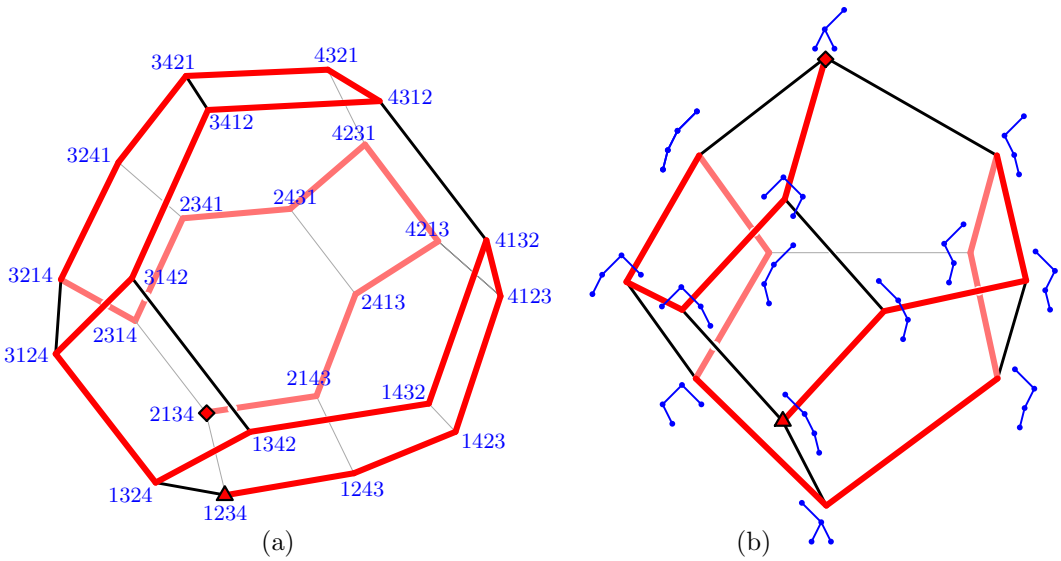

Figure 1: The three-dimensional permutahedron (left) and associahedron (right), with the Steinhaus-Johnson-Trotter Hamilton path and the Lucas-Roelants van Baronaigien-Ruskey Hamilton path, respectively (bold edges). The starting and end vertices are marked by a triangle or a diamond, respectively.

We emphasize that an elimination tree is unordered, i.e., there is no ordering associated with the children of a vertex in the tree. Similarly, there is no ordering among the elimination trees in an elimination forest. It is useful to think of an elimination tree for a graph G as the outcome of the process of removing vertices in some *elimination ordering*, which is a permutation that specifies the order of removed vertices; see Figure 2 (c): We first remove the root x from G, then proceed to remove the next vertex in the ordering from the connected component of $G - x$ it belongs to. In general, one elimination tree corresponds to several distinct elimination orderings. Specifically, these are all the linear extensions of the partial order whose cover graph is the elimination tree turned upside down.

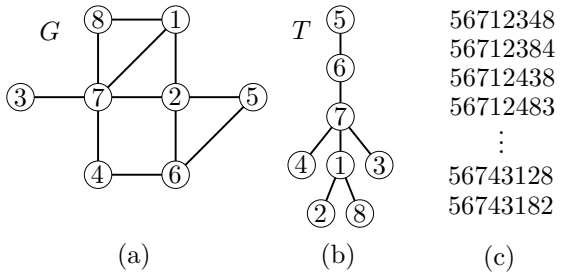

Figure 2: (a) A connected graph G; (b) an elimination tree T for G; (c) elimination orderings yielding the same tree T.

1.4 Applications and related notions. Elimination trees are also found under the guise of vertex rankings and centered colorings, and elimination forests are also known as G-forests [BM21], spines [MP15], and when defined in the more general context of building sets, as $\mathcal{B}$-forests [Pos09]. They have been studied extensively in various contexts, including data structures, combinatorial optimization, graph theory, and polyhedral combinatorics.

For example, Liu and coauthors [Liu88, Liu89, Liu90, EL05, EL07] used elimination trees in efficient parallel algorithms for matrix factorization. Elimination trees are also met in the context of VLSI design [Lei80, SDG92], and for parallel scheduling in modular products manufacturing [IRV88b, IRV91, NW89]. In the context of scheduling, one is typically interested in finding an elimination tree of minimum height, which determines the number of parallel steps in the schedule. This problem, known to be NP-hard in general, has drawn a lot of attention in the last thirty years [Sch93, AH94, DKKM94, DKKM99, BGHK95, BDJ$^+$98]. Computing optimal elimination trees for trees G is possible in linear time [IRV88a, Sch89].

A central notion in graph theory is the *tree-depth* of a graph, which is yet another name for the minimum

height of an elimination tree [NO12, RRSS14, FGP15, KR18]. In particular, tree-depth and elimination trees can be defined via the following other well-known objects. A *ranking* of the vertices of a graph G is a labeling of its vertices with integers from $\{1, 2, \ldots, k\}$ such that any path between two vertices with the same label contains a vertex with a larger label. A *centered coloring* of a graph G is a vertex coloring such that for any connected subgraph H, some color appears exactly once in H. It is not difficult to show that the minimum k for which there exists a vertex ranking of G is equal to the minimum number of colors in a centered coloring of G, which is in turn equal to the tree-depth of G. For a connected graph G, the elimination tree corresponding to a vertex ranking or a centered coloring can be constructed by iteratively picking respectively the largest label or the unique color as the root x of the tree, and recursing on the connected components of $G - x$.

Elimination trees also occur naturally in the problem of searching in a tree or a graph [BAFN99, OP06, MOW08, EZKS16], with applications to fault detection and database integrity checking. Recently, an online search model on trees was defined based on elimination trees [BCI+20], which generalizes [BK20] the classical splay tree data structure of Sleator and Tarjan.

1.5 Encoded combinatorial objects. In the context of combinatorial generation, elimination trees are interesting, as they encode several familiar combinatorial objects:

- When G is the complete graph on $[n]$, then its elimination trees are paths, which can be interpreted as *permutations* of $[n]$: Read off the vertex labels from the root to the leaf in the elimination tree; see Figure 3 (a).
- When G is the path with vertices labeled $1, \ldots, n$ between the end vertices, then its elimination trees are all n-vertex *binary trees*: The distinction between left and right child in the binary tree is induced by the smaller and larger vertex labels; see Figure 3 (b).
- When G is a star with 1 as the center and with leaves $2, \ldots, n$, then its elimination trees are brooms: a path composed of elements from a subset of $[n] \setminus \{1\}$, followed by a subtree of height one rooted in 1. By reading off the labels from the handle of the broom starting at the root and ending at the parent of 1, and subtracting 1 from those labels, we obtain a linearly ordered subset of $[n-1]$, which is known as a *partial permutation*; see Figure 3 (c). We see that elimination trees for stars are in one-to-one correspondence with partial permutations.
- The graph G may also be disconnected. In particular, if G is a disjoint union of n edges $\{i, n+i\}$ for $i = 1, \ldots, n$, then its elimination forests consist of n disjoint one-edge trees, which are either rooted in i or $n+i$ for all $i = 1, \ldots, n$. We can thus interpret the elimination forest as a *bitstring* of length n, where the ith bit is 0 if i is root, and the ith bit is 1 if $n+i$ is root; see Figure 3 (d).

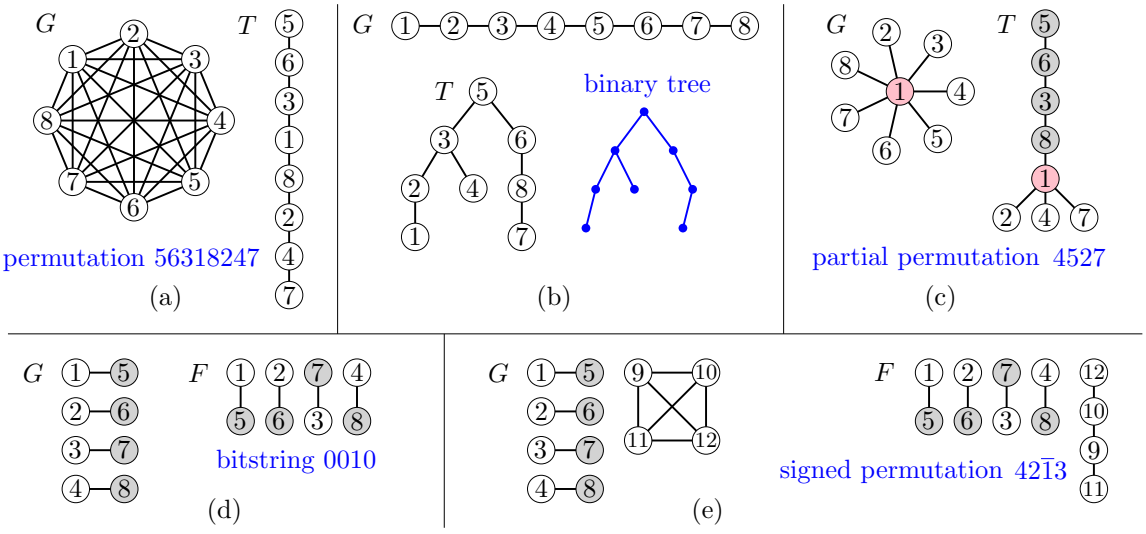

Figure 3: Combinatorial objects encoded by elimination trees for suitable graphs G.

- Combining the aforementioned encodings for permutations and bitstrings, we can take G as a disjoint union of n edges $\{i, n+i\}$ for $i = 1, \ldots, n$ and a complete graph on the n vertices $\{2n+1, \ldots, 3n\}$. The elimination forests for G can be interpreted as *signed permutations* of $[n]$: Read off the vertex labels of the path on $\{2n+1, \ldots, 3n\}$ from the root to the leaf in the corresponding elimination tree, subtracting $2n$ from those labels, and take the resulting entry i of the permutation with positive sign if i is root and with negative sign if $n+i$ is root; see Figure 3 (e).

The task of generating all elimination trees for a graph considered in this paper is thus a generalization of generating each of the aforementioned concrete classes of combinatorial objects.

1.6 Rotations and graph associahedra. Elimination trees can be locally modified by rotation operations, which generalize the binary tree rotations used in standard online binary search tree algorithms [AVL62, GS78, ST85]. In fact, rotations are one of the elementary, unit-cost operations in the online search model studied in [BCI+20, BK20].

Formally, rotations in elimination trees are defined as follows; see Figure 4. Let T be an elimination tree for a connected graph G and let j be a vertex from G, distinct from the root of T. Let i be the parent of j in T, and let H be the subgraph of G induced by the vertices in the subtree rooted at i. Then the *rotation of the edge* $\{i, j\}$ transforms T into another elimination tree T' for G in which:

- j becomes the parent of i, and the child of the parent of i in T (or the root if i is the root of T),
- the subtrees of i in T remain subtrees of i,
- a subtree S of j in T remains a subtree of j, unless the vertices of S belong to the same connected component of $H - j$ as i, in which case S becomes a subtree of i.

A rotation in an elimination forest for a disconnected graph is a rotation in one of its elimination trees. A rotation can be interpreted as the change in an elimination tree for G resulting from swapping i and j in the elimination ordering of the vertices.

Under the encodings discussed in Section 1.5, elimination tree rotations correspond to natural 'local change' operations on the corresponding combinatorial objects. Specifically, one can check that they translate to adjacent transpositions in permutations, classical rotations in binary trees, adjacent transpositions or deletions or insertions of a trailing element in partial permutations, flipping a single bit in bitstrings, or adjacent transpositions or sign changes in signed permutations, respectively.

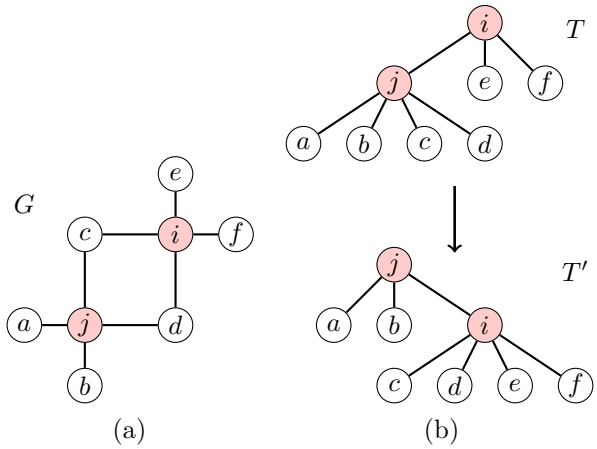

Figure 4: Elimination tree rotation. Part (a) shows two vertices i and j in a graph G; (b) shows the corresponding tree rotation.

It is well known that for any graph G, the flip graph of elimination forests for G under tree rotations is the skeleton of a polytope, referred to as the *graph associahedron* $\mathcal{A}(G)$ [CD06, Dev09, Pos09]. Graph associahedra are special cases of *generalized permutahedra* that have applications in algebra and physics [PRW08, AA17]. For G being a complete graph, a cycle, a path, a star, or a disjoint union of edges, $\mathcal{A}(G)$ is the permutahedron, the cyclohedron, the standard associahedron, the stellohedron, or the hypercube, respectively. Figure 5 shows the graph associahedra of all 4-vertex graphs.

We consider the problem of generating all elimination forests for a graph G by rotations, or equivalently, of computing Hamilton paths and cycles on the graph associahedron $\mathcal{A}(G)$. In previous work, Manneville and Pilaud [MP15] showed that for any graph G with at least two edges, $\mathcal{A}(G)$ has a Hamilton cycle. Their construction is an inductive gluing argument on $\mathcal{A}(G)$, which does not translate into an efficient algorithm for computing such

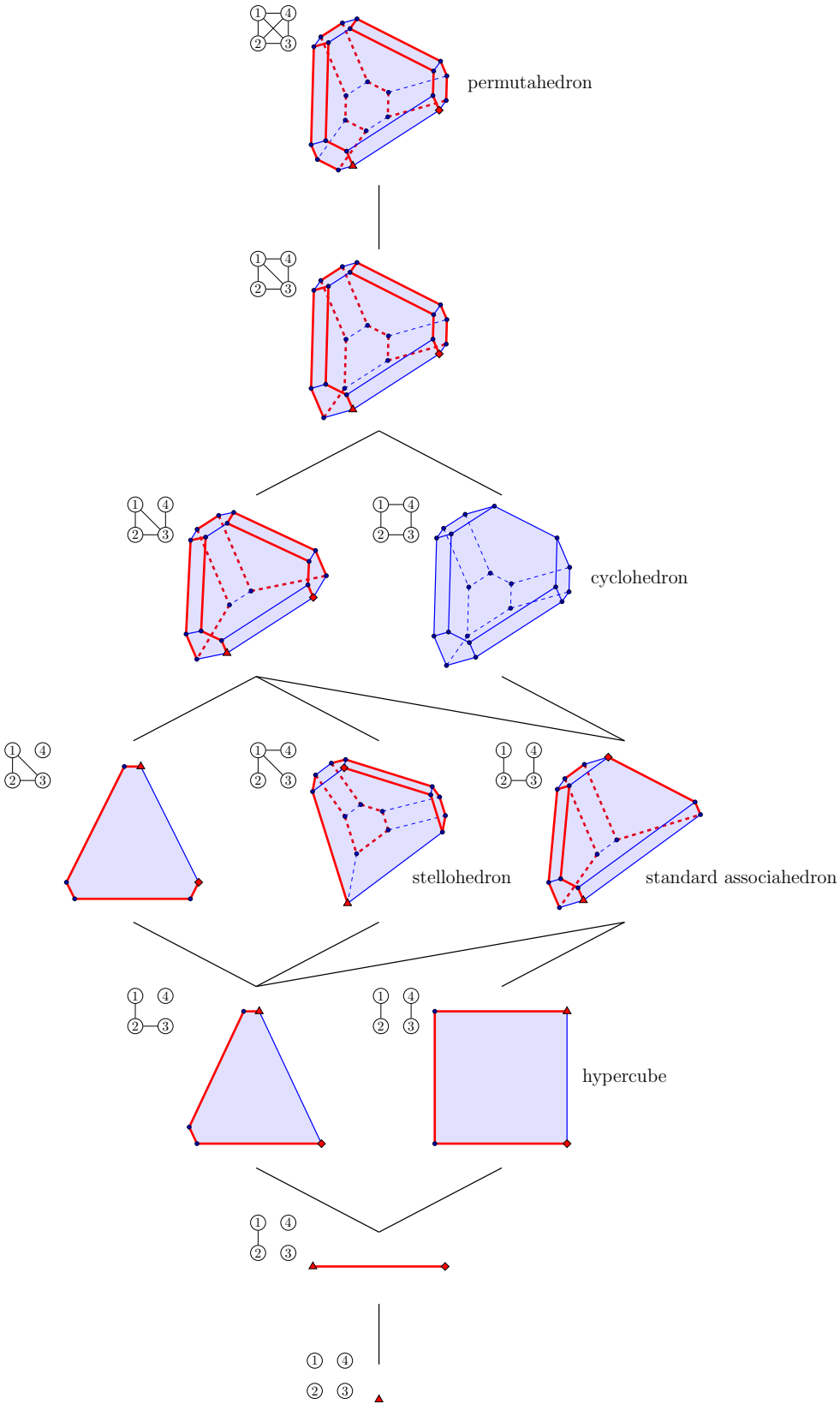

Figure 5: Graph associahedra $\mathcal{A}(G)$ for all graphs G on $n = 4$ vertices, ordered by subgraph inclusion. The Hamilton paths computed by our algorithm for all chordal graphs G are highlighted, with the starting and end vertex marked by a triangle and diamond, respectively. The only non-chordal graph is the 4-cycle.

a cycle. Note that the number of vertices of $\mathcal{A}(G)$ is in general exponential in the number n of vertices of the underlying graph G (for example, the permutahedron has $n!$ vertices), which makes global manipulations on $\mathcal{A}(G)$ prohibitive for combinatorial generation, where we aim for an algorithm that visits each vertex of $\mathcal{A}(G)$ in time polynomial in n, ideally even constant.

To obtain such an efficient algorithm, we apply the combinatorial generation framework recently proposed by Hartung, Hoang, Mütze, and Williams [HHMW20]. In this framework, the objects to be generated are encoded by permutations, and those permutations are generated by a simple greedy algorithm. Our encoding considers for each elimination tree of an n-vertex graph the set of all elimination orderings (=permutations of $[n]$) corresponding to this tree (recall Figure 2 (b)+(c)), and fixes precisely one representative permutation from this set. These representatives are chosen so that their union, which is a subset of all permutations of $[n]$, forms a so-called *zigzag language*, a term defined in [HHMW20] via a closure property. The algorithm proposed in that paper to generate zigzag languages and the combinatorial objects they encode can be implemented efficiently for many classes of objects, and it subsumes several previously studied Gray codes. In a series of recent papers, this framework was applied to a plethora of combinatorial objects such as pattern-avoiding permutations [HHMW21], lattice quotients of the weak order on permutations [HM21], and rectangulations [MM21]. In this work, we extend the reach of this framework and make it applicable to the efficient generation of structures on graphs, specifically of elimination forests, which is a step forward in exploring the generality of this approach. This is achieved by combining algorithmic, combinatorial, and polytopal insights and methods.

2 Our results

In the following we summarize the main results of this work and sketch the main ideas for proving them. In this extended abstract, no formal proofs of our results are given. They can be found in the preprint [CMM21].

2.1 A simple algorithm for generating elimination forests for chordal graphs. For our algorithm it is convenient to encode the rotation of edges $\{i, j\}$, $i < j$, by the larger end vertex j of the edge, and by the direction in which i is reached from j, namely upwards if i is the parent of j and downwards if i is a child of j. This is the direction in which the vertex j moves as a result of the rotation. We refer to these operations as *up- and down-rotations of* j, respectively, and we use the shorthand notations $j\,\triangle$ and $j\,\triangledown$. Observe that a down-rotation $j\,\triangledown$ is only well-defined if j has a unique child that is smaller than j, otherwise there are several choices for children $i < j$ of j and consequently several possible edges to rotate.

We propose to generate the set $\mathcal{E}(G)$ of all elimination forests for a graph $G = ([n], E)$, $[n] := \{1, 2, \ldots, n\}$, using the following simple greedy algorithm.

Algorithm R *(Greedy rotations).* This algorithm attempts to greedily generate the set $\mathcal{E}(G)$ of elimination forests for a graph $G = ([n], E)$ using rotations starting from an initial elimination forest $F_0 \in \mathcal{E}(G)$.

R1. [Initialize] Visit the initial elimination forest F_0.

R2. [Rotate] Generate an unvisited elimination forest from $\mathcal{E}(G)$ by performing an up- or down-rotation of the largest possible vertex in the most recently visited elimination forest. If no such rotation exists, or the rotation edge is ambiguous, then terminate. Otherwise, visit this elimination forest and repeat R2.

In other words, we consider the vertices $n, n-1, \ldots, 2$ of the current elimination forest in decreasing order, and for each of them we check whether it allows an up- or down-rotation that creates a previously unvisited elimination forest, and we perform the first such rotation we find, unless the same vertex allows several possible rotations, in which case we terminate. We also terminate if no rotation creates an unvisited elimination forest.

For example, consider all elimination trees for the 4-cycle with vertices labeled $1, 2, 3, 4$ cyclically; see Figure 6. When initialized with the elimination tree $F_0 = T_0$ shown in the figure, the algorithm visits the 17 elimination trees $T_0, \ldots, T_{16}$. The tree T_0 admits an up-rotation of 4, yielding T_1. The tree T_1 admits an up- and down-rotation of 4, but the latter would yield T_0, which was already visited, so we perform $4\,\triangle$, yielding T_2. One more up-rotation of 4 gives T_3, which does not admit any rotations of 4 to unvisited elimination trees. Consequently, we

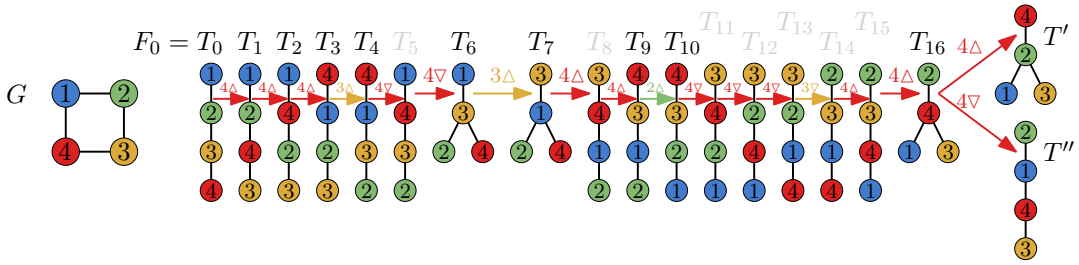

Figure 6: The output $T_0, \ldots, T_{16}$ of Algorithm R for the 4-cycle.

consider the vertex 3, which does admit an up-rotation, yielding T_4. The next interesting step is T_6, which does not admit rotations of 4 to unvisited elimination trees. However, T_6 admits an up- and down-rotation of 3, but the latter would lead to T_0 again, so we perform $3\,\triangle$ to reach T_7. From T_9 to T_{10} we up-rotate 2, as neither 4 nor 3 admit rotations to unvisited elimination trees. The algorithm eventually terminates with T_{16}, which admits both an up- and down-rotation of 4 to two previously unvisited elimination trees T' and T''. Because of this ambiguity, the algorithm terminates without exhaustively generating all elimination trees for G.

Figure 7 shows the output of Algorithm R for four other graphs G, and in all those cases the algorithm terminates because from the last elimination forest in those lists, no rotation leads to a previously unvisited elimination forests. Moreover, those four lists are all exhaustive, i.e., the algorithm succeeds in generating *all* elimination forests for those graphs.

Our main result is that Algorithm R succeeds to generate $\mathcal{E}(G)$ exhaustively for *chordal* graphs G, i.e., graphs in which every induced cycle has length three. Chordal graphs include many interesting subclasses, such as paths, stars, trees, k-trees, complete graphs, interval graphs, and split graphs (in particular, all the graph classes mentioned in Section 1.5). A classical characterization of chordal graphs is that they have *perfect elimination ordering*, i.e., a linear ordering of their vertices such that every vertex x induces a clique together with its neighbors in the graph that come before x in the ordering. In what follows, we consider a chordal graph $G = ([n], E)$, where the ordering $1, 2, \ldots, n$ is a perfect elimination ordering of G.

THEOREM 1. *Given any chordal graph $G = ([n], E)$ in perfect elimination order, Algorithm R visits every elimination forest from $\mathcal{E}(G)$ exactly once, when initialized with the elimination forest F_0 that is obtained by removing vertices in increasing order.*

Theorem 1 thus provides a short proof that the graph associahedron $\mathcal{A}(G)$ has a Hamilton path for chordal graphs G. Figure 5 shows the Hamilton paths on the graph associahedra for all chordal 4-vertex graphs computed by our algorithm.

Algorithm R generalizes several known Gray codes, including the Steinhaus-Johnson-Trotter algorithm for permutations (when G is a complete graph; see Figure 7 (a) and Table 1), the binary tree Gray code due to Lucas, Roelants van Baronaigien, and Ruskey (when G is a path; see Figure 7 (b)), and the binary reflected Gray code for bitstrings (when G is a disjoint union of edges; see Figure 7 (d)). The Gray code for partial permutations (when G is a star; see Figure 7 (c)) via adjacent transpositions or deletions or insertions of a trailing element is new, and it can be implemented in constant time per generated object (see the next section).

Intuitively, the reason why Algorithm R succeeds for chordal graphs is that in every elimination forest for a chordal graph, every vertex j has at most one child that is smaller than j. We show that this property characterizes chordality, i.e., a graph is chordal if and only if all of its elimination forests have this property. It ensures that to every vertex, at most one up-rotation and at most one down-rotation is applicable, and if both are applicable, then one of the two resulting elimination forests has been visited before by the algorithm, and hence the other one is visited next. In other words, there will never be ambiguity about two possible down-rotations of a vertex that lead to unvisited elimination forests, or one possible up-rotation and one down-rotation (as in the last step in Figure 6), so the algorithm does not terminate prematurely. By definition, the algorithm generates a previously unvisited

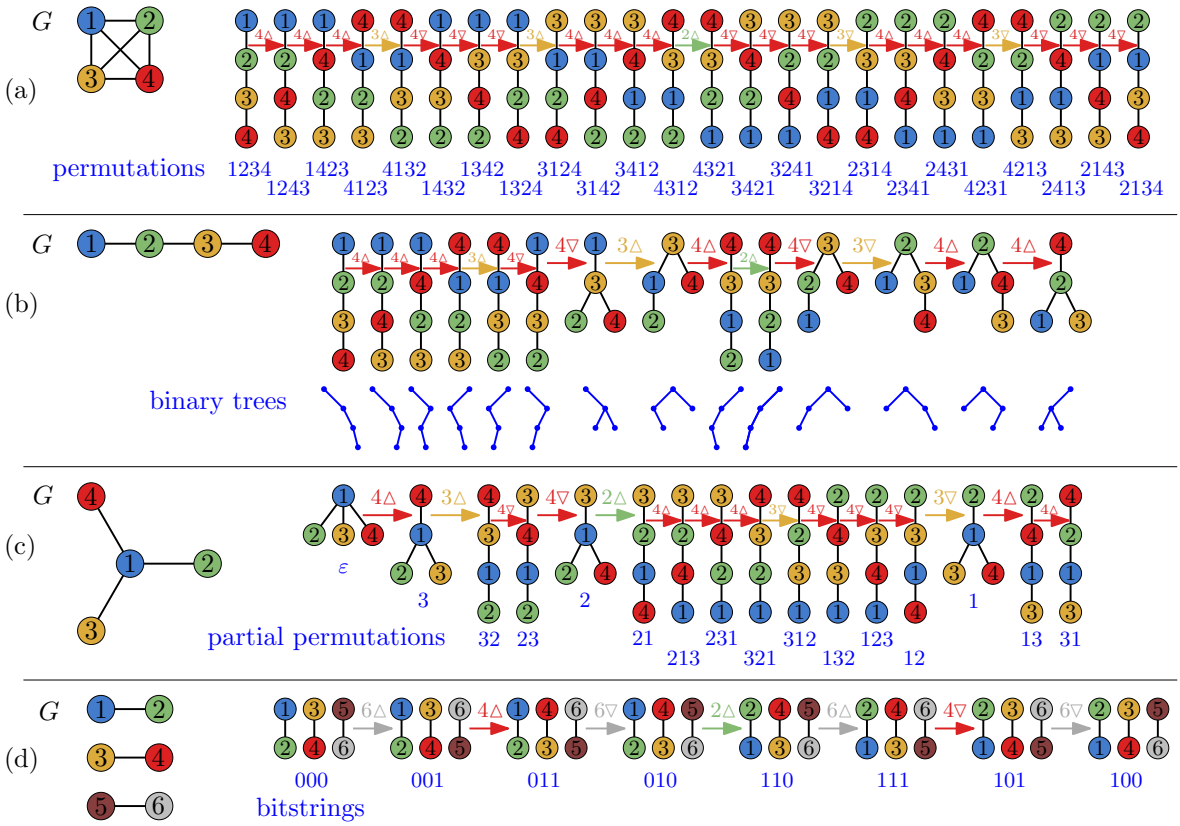

Figure 7: The output of Algorithm R for four different chordal graphs G, and the corresponding Gray codes of combinatorial objects.

elimination forest in every step, so avoiding premature termination guarantees that $\mathcal{E}(G)$ is generated exhaustively.

In fact, we show that Algorithm R generates a Hamilton path on the graph associahedron $\mathcal{A}(G)$ if and only if G is chordal. As Algorithm R is oblivious of the notion of a chordal graph, this is an interesting new characterization of graph chordality.

2.2 Efficient implementation of the algorithm. When implemented naively, Algorithm R requires storing all previously visited elimination forests, in order to decide upon the next rotation. We can get rid of this defect and make the algorithm memoryless and efficient.

THEOREM 2. *Algorithm R can be implemented such that for any chordal graph $G = ([n], E)$ with $m = |E|$ edges in perfect elimination order, the algorithm visits each elimination forest for G in time $\mathcal{O}(m+n)$. For trees G, this can be improved to $\mathcal{O}(1)$ for visiting each elimination tree for G.*

The memory and initialization time required for these algorithms is $\mathcal{O}(m+n)$ for chordal graphs G and $\mathcal{O}(n)$ for trees G, respectively. The initialization time includes the time for testing chordality and computing a perfect elimination ordering. The obtained algorithm for trees G is loopless, i.e., the time bound $\mathcal{O}(1)$ holds in every iteration. Recall that trees G are of particular interest in view of the special cases mentioned in Section 1.5 and the data structure applications discussed at the end of Section 1.3.

We implemented both of these algorithms in C++, and made the code available for download, experimentation and visualization on the Combinatorial Object Server [cos].

To achieve the runtime bounds stated in Theorem 2, we maintain an array of direction pointers $o = (o_1, \ldots, o_n)$, where an entry $o_j = \triangle$ indicates that the vertex j is rotating up in its elimination tree when it is rotated next by

the algorithm, and $o_j = \triangledown$ indicates that j is rotating down upon the next rotation. The direction is reversed if after an up-rotation $j\,\triangle$ the vertex j has become the root of its elimination tree or its parent is larger than j, or if after a down-rotation $j\,\triangledown$ the vertex j has become a leaf or its children are all larger than j. In addition, we introduce an array that allows us to determine in constant time which vertex j is rotating in the next step, i.e., which is the 'largest possible vertex' in step R2 of Algorithm R.

The bottleneck in the $\mathcal{O}(m+n)$ bound for chordal graphs is the time needed for one graph search in G. Indeed, to perform the rotation of an edge $\{i,j\}$ in an elimination tree T as illustrated in Figure 4, we consider the subgraph H of G induced by the vertices in the subtree rooted at i, which is obtained by marking all ancestors of i in T as removed (without actually removing them). In addition, we mark j as removed, but not i. Observe that a subtree of j changes its parent from j to i in this rotation operation if and only if the root of this subtree is reachable from i via non-marked vertices, i.e., vertices in $H - j$. This reachability of vertices in $H - j$ from i can be decided by a single graph search on non-marked vertices starting at i.

For trees G, every pair of vertices is connected by a unique path, and at most one subtree changes parent upon a tree rotation. This allows us to dispense with the graph search and obtain a loopless $\mathcal{O}(1)$ algorithm.

2.3 Hamilton cycles for 2-connected chordal graphs. Lastly, we investigate when Algorithm R produces a cyclic Gray code, i.e., a Hamilton cycle on the graph associahedron $\mathcal{A}(G)$, rather than just a Hamilton path. We aim to understand under which conditions on G the first and last elimination forest generated by our algorithm differ in a single tree rotation. In the examples from Figure 7, this is the case for (a) and (d), but not for (b) and (c). We derive a number of such conditions, two of which are summarized in the following theorem. A graph is *2-connected*, if it has at least three vertices and removing any vertex leaves a connected graph.

THEOREM 3. *Let $G = ([n], E)$ be a chordal graph in perfect elimination order. If G is 2-connected, then the rotation Gray code for $\mathcal{E}(G)$ generated by Algorithm R is cyclic. On the other hand, if G is a tree with at least four vertices, then this Gray code is not cyclic.*

To appreciate the number of cases covered by our results in Theorems 1–3, we remark that the number of non-isomorphic n-vertex graphs is $2^{n^2/2(1+o(1))}$ [HP73], and the number of chordal graphs and of 2-connected chordal graphs is $2^{n^2/4(1+o(1))}$ [Wor85] (with different $o(1)$ terms).

3 Open problems

We conclude this paper with the following remarks and open problems.

- While we precisely characterized the class of graphs on which our method applies, one may wonder whether it could be applied to more general families of polytopes. *Nestohedra*, for instance, generalize graph associahedra, and can be defined as Minkowski sums of standard simplices corresponding to families of subsets of $\{1, 2, \ldots, n\}$ known as building sets [Pos09]. The main property of building sets is that the union of two intersecting subsets must also be in the building set, which clearly holds for connected subsets of vertices of a graph. In this special case, the building set is said to be graphical, and we recover the definition of graph associahedra. Postnikov, Reiner, and Williams [PRW08] define chordal nestohedra as a generalization of chordal graph associahedra, via the definition of chordal building sets. We omit details here, but our generation algorithm should apply directly to chordal nestohedra, further extending its scope of applicability. Considering arbitrary hypergraphs instead of building sets yields the class of *hypergraphic polytopes* [AA17, BBM19], not all of which are Hamiltonian. The question of the applicability of our generation algorithm in this wider setting is worth considering.
- Can we generalize the methods from this paper to efficiently compute Hamilton paths and cycles in graph associahedra $\mathcal{A}(G)$ for non-chordal graphs G? As a first step, one might try to tackle the case of G being a cycle, i.e., cyclohedra. As we saw for the 4-cycle in Figure 6, and more generally for non-chordal graphs, the simple greedy rule of Algorithm R will stop prematurely because of ambiguity. To overcome this, a more global algorithmic control seems to be necessary, possibly using ingredients from Manneville and Pilaud's Hamiltonicity proof [MP15]. However, the cycles resulting from their inductive gluing argument have a completely different

structure from the ones produced by our Algorithm R for chordal graphs, namely they are made of n blocks and one of the n vertices of the graph is root in all elimination trees of one block. In contrast to that, the largest vertex n in our cycles constantly moves up and down the elimination tree, and is root for only two consecutive steps each.

- Another worthwile goal is to efficiently compute Hamilton cycles in graph associahedra $\mathcal{A}(G)$ for any chordal graph G, specifically for graphs that are not 2-connected, in particular for trees G. Hurtado and Noy [HN99] provide a simple construction when G is a path (i.e., the standard associahedron), which can probably be turned into an efficient algorithm and possibly be generalized.
- Can the runtime bound of $\mathcal{O}(m+n)$ of our algorithm per generated elimination tree be improved? Recall that m and n are the number of edges and vertices of G, respectively.
- The function that counts the number of elimination forests for a graph G, referred to as the G-Catalan number by Postnikov [Pos09], deserves further study. For example, what is the complexity of computing it? This question is directly related to unranking and ranking in the orderings computed by our algorithm, or to use our generation approach for random sampling.

Acknowledgements

We thank Vincent Pilaud for inspiring discussions about graph associahedra, and we thank Petr Gregor for helpful comments on an earlier version of this paper. We also thank the reviewers of this paper for their remarks and suggestions.

References

[AA17] M. Aguiar and F. Ardila. Hopf monoids and generalized permutahedra. `https://arxiv.org/abs/1709.07504`, 2017.

[AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1-3):21–46, 1996. First International Colloquium on Graphs and Optimization (GOI), 1992 (Grimentz).

[AH94] B. Aspvall and P. Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT*, 34(4):484–509, 1994.

[AVL62] G. M. Adel'son-Vel'skiĭ and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146:263–266, 1962.

[BAFN99] Y. Ben-Asher, E. Farchi, and I. Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999.

[BBM19] C. Benedetti, N. Bergeron, and J. Machacek. Hypergraphic polytopes: combinatorial properties and antipode. *J. Comb.*, 10(3):515–544, 2019.

[BCI+20] P. Bose, J. Cardinal, J. Iacono, G. Koumoutsos, and S. Langerman. Competitive online search trees on trees. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 1878–1891. SIAM, Philadelphia, PA, 2020.

[BDJ+98] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Zs. Tuza. Rankings of graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998.

[BDPP99] E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO: a methodology for the enumeration of combinatorial objects. *J. Differ. Equations Appl.*, 5(4-5):435–490, 1999.

[BGHK95] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

[BK20] B. A. Berendsohn and L. Kozma. Splay trees on trees. To appear in *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms*; preprint available at `https://arxiv.org/abs/2010.00931`, 2020.

[BM21] E. Barnard and T. McConville. Lattices from graph associahedra and subalgebras of the Malvenuto-Reutenauer algebra. *Algebra Universalis*, 82(1):Paper No. 2, 53, 2021.

[CD06] M. P. Carr and S. L. Devadoss. Coxeter complexes and graph-associahedra. *Topology Appl.*, 153(12):2155–2168, 2006.

[CIRS03] L. S. Chandran, L. Ibarra, F. Ruskey, and J. Sawada. Generating and characterizing the perfect elimination orderings of a chordal graph. *Theoret. Comput. Sci.*, 307(2):303–317, 2003.

[CMM21] J. Cardinal, A. Merino, and T. Mütze. Combinatorial generation via permutation languages. IV. Elimination trees. `https://arxiv.org/abs/2106.16204`, 2021.

[cos] The Combinatorial Object Server: Generate elimination trees. `http://www.combos.org/elim`.

[CRS+00] K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C. R. Miers. Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF(2). *J. Algorithms*, 37(2):267–282, 2000.

[Dev09] S. L. Devadoss. A realization of graph associahedra. *Discrete Math.*, 309(1):271–276, 2009.

[DKKM94] J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller. On vertex ranking for permutations and other graphs. In *STACS 94 (Caen, 1994)*, volume 775 of *Lecture Notes in Comput. Sci.*, pages 747–758. Springer, Berlin, 1994.

[DKKM99] J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Appl. Math.*, 98(1-2):39–63, 1999.

[Ehr73] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. Assoc. Comput. Mach.*, 20:500–513, 1973.

[EL05] S. Eisenstat and J. W. H. Liu. The theory of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 26(3):686–705, 2005.

[EL07] S. C. Eisenstat and J. W. H. Liu. Algorithmic aspects of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 29(4):1363–1381, 2007.

[EZKS16] E. Emamjomeh-Zadeh, D. Kempe, and V. Singhal. Deterministic and probabilistic binary search in graphs. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 519–532. ACM, New York, 2016.

[FGP15] F. V. Fomin, A. C. Giannopoulou, and M. Pilipczuk. Computing tree-depth faster than 2^n. *Algorithmica*, 73(1):202–216, 2015.

[FM94] K. Fukuda and T. Matsui. Finding all the perfect matchings in bipartite graphs. *Appl. Math. Lett.*, 7(1):15–18, 1994.

[Gra53] F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

[GS78] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978)*, pages 8–21. IEEE, Long Beach, Calif., 1978.

[HHMW20] E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1214–1225. SIAM, 2020.

[HHMW21] E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. To appear in *Trans. Amer. Math. Soc.*; preprint available at `https://arxiv.org/abs/1906.06069`, 2021.

[HL07] C. Hohlweg and C. E. M. C. Lange. Realizations of the associahedron and cyclohedron. *Discrete Comput. Geom.*, 37(4):517–543, 2007.

[HM21] H. P. Hoang and T. Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. To appear in *Israel J. Math.*; preprint available at `https://arxiv.org/abs/1911.12078`, 2021.

[HN99] F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Comput. Geom.*, 13(3):179–188, 1999.

[HP73] F. Harary and E. M. Palmer. *Graphical enumeration.* Academic Press, New York-London, 1973.

[IRV88a] A. V. Iyer, H. D. Ratliff, and G. Vijayan. Optimal node ranking of trees. *Inform. Process. Lett.*, 28(5):225–229, 1988.

[IRV88b] A. V. Iyer, H. D. Ratliff, and G. Vijayan. Parallel assembly of modular products, Technical Report 88-06. Technical report, Production and Distribution Research Center, Georgia Institute of Technology, Atlanta, GA, 1988.

[IRV91] A. V. Iyer, H. D. Ratliff, and G. Vijayan. On an edge ranking problem of trees and graphs. *Discrete Appl. Math.*, 30(1):43–52, 1991.

[Joh63] S. Johnson. Generation of permutations by adjacent transposition. *Math. Comp.*, 17:282–285, 1963.

[Kay76] R. Kaye. A Gray code for set partitions. *Information Processing Lett.*, 5(6):171–173, 1976.

[Knu11] D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011.

[KR18] K. Kawarabayashi and B. Rossman. A polynomial excluded-minor approximation of treedepth. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 234–246. SIAM, Philadelphia, PA, 2018.

[Lei80] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 270–281. IEEE Computer Society, 1980.

[Liu88] J. W. H. Liu. Equivalent sparse matrix reordering by elimination tree rotations. *SIAM J. Sci. Statist. Comput.*, 9(3):424–444, 1988.

[Liu89] J. W. H. Liu. A graph partitioning algorithm by node separators. *ACM Trans. Math. Software*, 15(3):198–219, 1989.

[Liu90] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[Lod04] J.-L. Loday. Realization of the Stasheff polytope. *Arch. Math. (Basel)*, 83(3):267–278, 2004.

[LRR93] J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993.

[LS09] Y. Li and J. Sawada. Gray codes for reflectable languages. *Inform. Process. Lett.*, 109(5):296–300, 2009.

[MM21] A. Merino and T. Mütze. Efficient generation of rectangulations via permutation languages. In *37th International Symposium on Computational Geometry*, volume 189 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. 54, 18. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2021.

[MOW08] S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–1105. ACM, New York, 2008.

[MP15] T. Manneville and V. Pilaud. Graph properties of graph associahedra. *Sém. Lothar. Combin.*, 73:Art. B73d, 31, 2015.

[NO12] J. Nešetřil and P. Ossona de Mendez. *Sparsity*, volume 28 of *Algorithms and Combinatorics*. Springer, Heidelberg, 2012. Graphs, structures, and algorithms.

[NW89] J. L. Nevins and D. E. Whitney, editors. *Concurrent Design of Products and Processes*. McGraw-Hill, 1989.

[OP06] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 379–388. IEEE Computer Society, 2006.

[Pos09] A. Postnikov. Permutohedra, associahedra, and beyond. *Int. Math. Res. Not. IMRN*, (6):1026–1106, 2009.

[Pou14] L. Pournin. The diameter of associahedra. *Adv. Math.*, 259:13–42, 2014.

[PR93] G. Pruesse and F. Ruskey. Gray codes from antimatroids. *Order*, 10(3):239–252, 1993.

[PR94] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM J. Comput.*, 23(2):373–386, 1994.

[PRW08] A. Postnikov, V. Reiner, and L. Williams. Faces of generalized permutohedra. *Doc. Math.*, 13:207–273, 2008.

[RRSS14] F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In *Automata, languages, and programming. Part I*, volume 8572 of *Lecture Notes in Comput. Sci.*, pages 931–942. Springer, Heidelberg, 2014.

[RSW12] F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. Combin. Theory Ser. A*, 119(1):155–169, 2012.

[Rus16] F. Ruskey. Combinatorial Gray code. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 342–347. Springer, 2016.

[Sav97] C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997.

[Sch89] A. A. Schäffer. Optimal node ranking of trees in linear time. *Inform. Process. Lett.*, 33(2):91–96, 1989.

[Sch93] P. Scheffler. Node ranking and searching on graphs. In *Proceedings of the Third Twente Workshop on Graphs and Combinatorial Optimization*, pages 159–162, 1993.

[SDG92] A. Sen, H. Deng, and S. Guha. On a graph partition problem with application to VLSI layout. *Inform. Process. Lett.*, 43(2):87–94, 1992.

[Smi97] M. J. Smith. Generating spanning trees. Master's thesis, University of Victoria, 1997.

[ST85] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32(3):652–686, 1985.

[Sta15] R. P. Stanley. *Catalan numbers*. Cambridge University Press, New York, 2015.

[Ste64] H. Steinhaus. *One hundred problems in elementary mathematics*. Basic Books, Inc., Publishers, New York, 1964. With a foreword by Martin Gardner.

[Ste86] G. Steiner. An algorithm to generate the ideals of a partial order. *Oper. Res. Lett.*, 5(6):317–320, 1986.

[STT88] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc.*, 1(3):647–681, 1988.

[TIAS77] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977.

[Tro62] H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5(8):434–435, 1962.

[VR81] Y. L. Varol and D. Rotem. An algorithm to generate all topological sorting arrangements. *Comput. J.*, 24(1):83–84, 1981.

[Wil13] A. Williams. The greedy Gray code algorithm. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 525–536, 2013.

[Wor85] N. C. Wormald. Counting labelled chordal graphs. *Graphs Combin.*, 1(2):193–200, 1985.