A SIMPLE ALGORITHM FOR COMPUTING HAMILTON PATHS ON INDEPENDENT SET POLYTOPES

JEAN CARDINAL, PIA HERKENRATH, TORSTEN MÜTZE, AND FRANCESCO VERCIANI

Abstract. The independent set polytope, or stable set polytope, of a graph G is the 0/1-polytope defined by the convex hull of the characteristic vectors of all independent sets of G. We present a simple algorithm for computing a Hamilton path on the independent set polytope of a given n-vertex graph G with amortized delay $\mathcal{O}(n)$. The independent sets are listed such that two consecutive sets differ either in removing a vertex, or adding a vertex and removing its neighbors from the independent set, i.e., the symmetric difference between two consecutive independent sets induces a star in G. As applications of this result, we obtain an algorithm to compute a Hamilton path on the matching polytope of an m-edge graph G with worst-case delay $\mathcal{O}(m)$, which lists all matchings of G in such a way that the symmetric difference between two consecutive matchings is a path on at most three edges. Furthermore, we obtain an algorithm to compute a Hamilton path on the chain polytope and order polytope of an n-element poset P with amortized delay $\mathcal{O}(n)$, which lists all antichains of P or all ideals of P, respectively, by star exchanges. Our algorithms are derived from the generic framework proposed by Merino and Mütze [Traversing combinatorial 0/1-polytopes via optimization. SIAM J. Comput., 53(5):1257–1292, 2024] for computing Hamilton paths on arbitrary 0/1-polytopes, which uses a linear optimization procedure as a black box. Our algorithms bypass solving the computationally intractable maximum weight independent set problem by a simple and purely combinatorial greedy rule.

1. Introduction

Polytopes with vertex coordinates in $\{0,1\}$, called *0/1-polytopes*, are ubiquitous in combinatorial optimization. This is because the solutions of many interesting optimization problems can be described as 0/1-vectors, by considering the characteristic vectors of the underlying sets. For example, consider a graph G with m edges labeled $1,\dots,m$, and let M be a matching in G, i.e., a set of edges, no two of which have a vertex in common. Then the *characteristic vector* of M, denoted $x^M \in \{0,1\}^m$ is defined as $x^M_e = 1$ if $e \in M$ and $x^M_e = 0$ if $e \notin M$. The *matching polytope* of G is defined as $\mathscr{M}(G) \coloneqq \operatorname{conv}(\{x^M \mid M \text{ matching in } G\}) \subset \mathbb{R}^m$ [Chv75]. The maximum weight matching problem is to maximize the linear objective function $w \cdot x$ for all $x \in \mathscr{M}(G)$, where $w \in \mathbb{R}^m$ is the vector of weights assigned to the edges of G. This problem has been a cornerstone of computer science research in the past decades, and can be solved efficiently with one of the many algorithms that are now taught in textbooks.

In this work, we focus on the independent set polytope of a graph G, also known as stable set polytope [Chv75]. Specifically, an *independent set* in a graph G is a set of vertices that spans no edges, and the *independent set polytope* is defined as $\mathscr{P}(G) \coloneqq \operatorname{conv}(\{x^U \mid$

(Jean Cardinal) Computer Science Department, Université Libre de Bruxelles, Belgium
(Pia Herkenrath) Institut für Mathematik, Universität Kassel, Germany
(Torsten Mütze) Institut für Mathematik, Universitt Kassel, Germany
(Francesco Verciani) Institut für Mathematik, Universität Kassel, Germany
E-mail addresses: jean.cardinal@ulb.be, pherkenrath@mathematik.uni-kassel.de, tmuetze@mathematik.uni-kassel.de, francesco.verciani@mathematik.uni-kassel.de.

This project was supported by German Science Foundation grant 522790373.

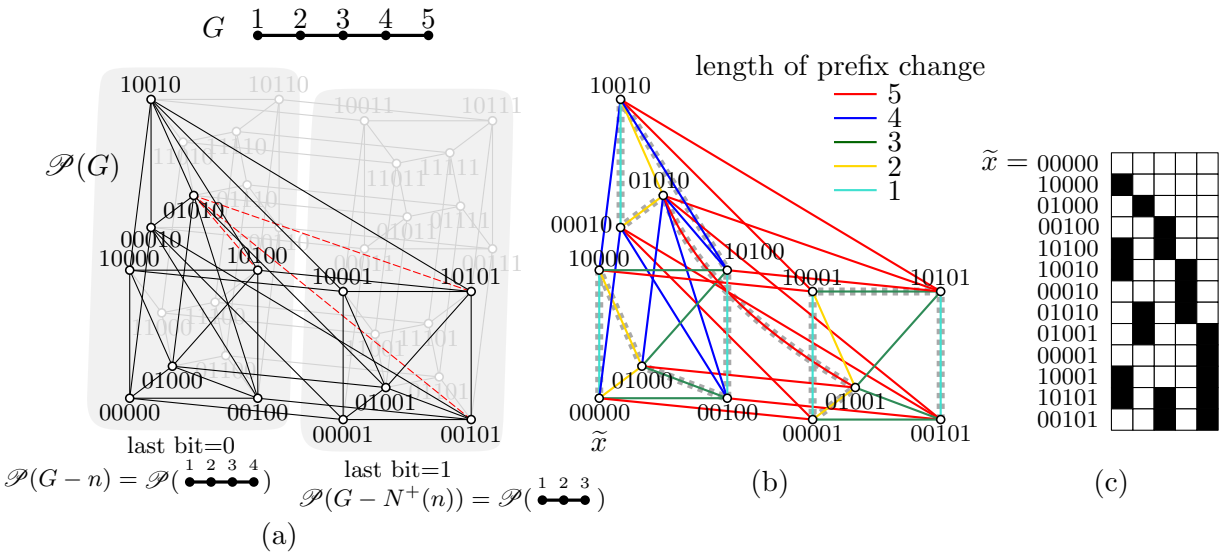


FIGURE 1. (a) Independent set polytope of the 5-vertex path, with vertices numbered $1, \ldots, 5$ from left to right. Characteristic vectors of independent sets of the path are bitstrings of length 5 that avoid two adjacent 1s. Edges of the polytope corresponding to star exchanges are solid, the others dashed. The shading highlights the two faces of the polytope in which the last bit is either 0 or 1. (b) Hamilton path computed by Algorithm I on the independent set polytope from the left, with initial vertex $\widetilde{x} = 00000$. The edges are colored according to the length of the prefix change, and the algorithm changes the shortest possible prefix that leads to a new vertex. (c) Vertices listed in the order of the Hamilton path, with 0-bits and 1-bits visualized as white and black squares, respectively, revealing the genlex property.

U independent set in $G\}$); see Figure 1 (a). We note that matchings of G can be seen as independent sets in the *line graph* $L(G)$, the graph that has a vertex for every edge of G, and an edge between vertices corresponding to edges of G that are incident to each other. Consequently, the matching polytope of G equals the independent set polytope of $L(G)$. For example, the independent set polytope of the 5-vertex path shown in Figure 1 equals the matching polytope of the 6-vertex (=5-edge) path. Unlike the aforementioned maximum weight matching problem, which is polynomial-time solvable, the corresponding optimization problem on the independent set polytope is NP-hard, as it includes the maximum cardinality independent set problem on G as a special case (all weights equal to 1).

Combinatorial 0/1-polytopes as the ones discussed before encode interesting information about the underlying combinatorial objects. For example, the edges of the matching polytope connect pairs of matchings whose symmetric difference is an alternating path or cycle. This is a consequence of the fact that edges of the independent set polytope connect pairs of independent sets whose symmetric difference induces a connected subgraph [Chv75]. The structure of these polytopes has also been studied extensively with the goal of understanding the aforementioned apparent dichotomy between tractability and intractability of the corresponding optimization problems.

1.1. **Hamilton paths on 0/1-polytopes.** In this work, we consider the problem of computing a Hamilton path on the independent set polytope of a given graph G. The goal is to visit every vertex of the polytope exactly once, such that any two consecutive vertices are connected by an

edge. One crucial property of such an algorithm is its *delay*, which is the time between any two consecutive vertices of the Hamilton path and before the first and after the last vertex. The delay can be specified as a worst-case guarantee, which is the maximum of the aforementioned quantities, or as an amortized guarantee, which is the average. The amortized delay is simply the total time of the algorithm to compute the entire path, divided by the total number of vertices.

Combinatorially, we aim to compute every independent set of G exactly once, such that the symmetric difference between any two consecutive independent sets induces a connected subgraph of G. We note that listings of combinatorial objects subject to some minimum-change condition are also known as *Gray codes* in the literature [Sav97, Müt23].

By a classical result of Naddef and Pulleyblank [NP84], every 0/1-polytope admits a Hamilton path. Recently, Merino and Mütze developed a generic algorithm for computing a Hamilton path on any 0/1-polytope [MM24], which was subsequently improved by Fink, Hladík, Merino, Mička and Mütze [FHM$^+$26]. Their algorithm uses as a black box an algorithm for solving the linear optimization problem over the corresponding polytope, and the resulting delay is given by the time to solve the linear optimization problem. As mentioned before, linear optimization over the independent set polytope is hard, so this approach does not directly give an efficient algorithm for computing a Hamilton path on the independent set polytope.

1.2. **Our results.** Our main contribution is a simple algorithm for computing a Hamilton path on the independent set polytope $\mathscr{P}(G)$ of a given graph G; see Figure 1 (b)+(c). The algorithm comes in a basic variant and an improved variant for faster speed (see Algorithms I and I* and Theorems 2 and 3 below). The algorithm has a number of desirable features:

- It works for every graph G, and for every labeling of the vertex set of G with integers $1, \dots, n$, which gives additional freedom that can be exploited in some applications.
- It works for every starting vertex of the polytope, i.e., for every initial independent set.
- The algorithm is purely combinatorial: it does not perform any polyhedral or numerical computations, and also does not rely on solving the corresponding (hard) optimization problem, and it can be implemented with few lines of code.
- The resulting amortized delay is linear in the number of vertices of G.
- To move from one independent set to the next, the algorithm applies one of two operations, referred to as a *star exchange*: either remove a vertex of G, or add a vertex of G and remove all its neighbors from the current independent set. Consequently, the subgraph of G induced by the symmetric difference between two consecutive independent sets is always a star. This also yields an algorithmic proof that the subgraph of the independent set polytope corresponding to star exchanges admits a Hamilton path.
- The computed ordering of independent sets has the so-called genlex property, which means that characteristic vectors with the same suffix appear consecutively.

We also present the following applications of our algorithm.

- By applying this algorithm to the n-vertex graph without edges, we recover the well-known binary reflected Gray code (Section 3.1). By applying the algorithm to paths and cycles, we obtain Gray codes for Fibonacci and Lucas words, respectively (Theorem 4).
- As vertex covers of a graph G are complements of independent sets of G, we also obtain an algorithm for computing a Hamilton path on the vertex cover polytope of G (Section 3.2).
- As mentioned before, matchings in a graph G are independent sets in its line graph $L(G)$. We thus obtain an algorithm for computing a Hamilton path on the matching polytope $\mathscr{M}(G)$, where any two consecutive matchings differ in an alternating path of length at most 3 (see

Algorithm M and Theorem 5). The (worst-case) delay is $\mathcal{O}(m)$, where m is the number of edges of G.

• Antichains in a poset P are independent sets in the comparability graph of P. We thus obtain an algorithm for computing a star exchange Hamilton path on Stanley's [Sta86] chain polytope $\mathscr{C}(P)$ (see Algorithm A^* and Theorem 6). In every step, either an element is removed from the current antichain, or an element is added and all elements in its downset are removed. The amortized delay per visited antichain is $\mathcal{O}(n)$, where n is the number of elements of P. Similarly, we obtain an algorithm for computing a star exchange Hamilton path on Stanley's order polytope $\mathscr{O}(P)$ (Theorem 7), with amortized delay $\mathcal{O}(n)$ per visited ideal of P. In every step, either a maximal element from the ideal is removed together with all elements that are uniquely covered by it, or an element is added together with its downset.

1.3. **Related work.** Kurita, Wasa, Uno and Arimura [KWUA21] describe an algorithm for listing all independent sets of a graph G with amortized delay $\mathcal{O}(\omega(G))$, where $\omega(G)$ is the clique number of G. In particular, for graphs with bounded clique number, the delay is constant. The computed listing is not a Hamilton path on the independent set polytope, though.

As our algorithm lists all independent sets of a given graph, one may wonder naturally about listing only the maximum independent sets, or only the inclusion-maximal ones. In fact, the maximum independent set polytope is a face of the independent set polytope, obtained by intersecting it with a hyperplane that fixes the cardinality of the sets. As the maximum independent set problem is NP-hard in general graphs, there is likely no polynomial delay algorithm for listing maximum independent sets. Tsukiyama, Ide, Ariyoshi and Shirakawa [TIAS77] give an algorithm for listing all inclusion-maximal independent sets with delay $\mathcal{O}(mn)$ (see also [CN85]), where here and in the following, m and n denote the number of edges and vertices of an input graph, respectively. This was later improved by Makino and Uno [MU04]. Johnson, Yannakakis and Papadimitriou [JYP88] provided an algorithm for listing all inclusion-maximal independent sets in lexicographic order with delay $\mathcal{O}(mn)$.

Kashiwabara, Masuda, Nakajima and Fujisawa [KMNF92] list maximum independent sets of a *bipartite* graph G with delay $\mathcal{O}(1)$, but not according to the associated polytope. Using the aforementioned reduction to an optimization problem, Merino and Mütze [MM24] obtained an algorithm for computing a Hamilton path on the independent set polytope and maximum independent set polytope of a bipartite graph, with delay $\mathcal{O}(m\sqrt{n})$ and $\mathcal{O}(mn)$, respectively.[1] Recall that the algorithm for the independent set polytope presented here has only amortized delay $\mathcal{O}(n)$ and works for general graphs.

Merino and Mütze [MM24] also gave an algorithm for computing Hamilton paths on the matching polytope $\mathscr{M}(G)$ and perfect matching polytope of any graph G, with delay $\mathcal{O}(m\sqrt{n})$ and $\mathcal{O}(m\sqrt{n}\log n)$, respectively. For the matching polytope, our new algorithm with delay $\mathcal{O}(m)$ improves on this. For comparison, the algorithm of Uno [Uno01] lists perfect matchings of a *bipartite* graph with delay $\mathcal{O}(\log n)$, but not according to the polytope.

The authors of [MM24] also obtained an algorithm for computing Hamilton paths on the chain polytope $\mathscr{C}(P)$ and order polytope $\mathscr{O}(P)$ of any poset P, with delay $\mathcal{O}(n^{2.5}/\sqrt{\log n})$ and $\mathcal{O}(n^4)$, respectively, where n is the number of elements of P. The amortized delay $\mathcal{O}(n)$ reported in this paper is a considerable improvement.

In the following, we describe a number of results on listing ideals of a poset in k-Gray code order, which means that the symmetric difference of any two consecutive ideals has size at most k. Note that a 1-Gray code is necessarily a Hamilton path on the order polytope, but a

[1]The delays stated here are without the $\log n$ factor that can be saved as described in [FHM+26].

k-Gray code for $k \geq 2$ is in general not. By considering the maximal elements of the ideals, a 1-Gray code for ideals also gives a Hamilton path on the chain polytope $\mathscr{C}(P)$ (see Section 3.4 below for details).

Koda and Ruskey [KR93] proved that ideals of a poset whose Hasse diagram is a forest of trees rooted at a minimal element admits a 1-Gray code that can be computed with delay $\mathcal{O}(1)$.

Pruesse and Ruskey [PR93] proved that the ideals of any poset admit a 2-Gray code, computable with delay $\mathcal{O}(n)$. Habib, Nourine, Steiner [HNS97] present a 2-Gray code for interval orders that can be computed with delay $\mathcal{O}(1)$. For general posets P, Habib, Medina, Nourine, Steiner [HMNS01] presented another 2-Gray code, computable with delay that is linear in the maximum down-degree of the Hasse diagram of P, but each ideal appears twice in the listing. In a recent breakthrough, Brenner and Fink [BF26] presented an algorithm for listing the ideals and antichains of any poset with amortized delay $\mathcal{O}(1)$, even in 3-Gray code order. As mentioned before, these Gray codes are in general not Hamilton paths on the associated polytopes.

1.4. **Outline.** In Section 2 we present our basic algorithm for computing a Hamilton path on the independent set polytope, and an improved variant of the algorithm for higher speed. In Section 3 we discuss in detail the aforementioned applications of our algorithm. At the end of this paper in Section 4, we present an extensive experimental comparison of both variants of our algorithm on a variety of input graphs.

2. The algorithm

2.1. **Preliminaries.** We define $[n] \coloneqq \{1, \ldots, n\}$. Throughout this paper we consider graphs $G = ([n], E)$ that have $[n]$ as their vertex set. This corresponds to labeling the vertices of G with integers $1, \ldots, n$. The algorithms discussed later depend on this labeling, hence changing the labeling of the graph yields a different output. Given a graph $G = ([n], E)$ we write $\mathrm{rev}(G)$ for the graph in which the labeling is reversed: label i is replaced by $n-i+1$ for all $i \in [n]$. For a subset $U \subseteq [n]$ we write $G[U]$ for the subgraph of G induced by U. We also write $G - U$ for the graph obtained from G by deleting all vertices in U and their incident edges. For a singleton $U = \{i\}$ we simply write $G - i \coloneqq G - \{i\}$. For a vertex $i \in [n]$ we write $N(i) \coloneqq \{j \in [n] \mid ij \in E\}$ and $N^+(i) \coloneqq N(i) \cup \{i\}$ for the *open* and *closed neighborhood* of i in G, respectively. We also partition the open neighborhood $N(i)$ into two sets, namely those that are smaller or larger than i, denoted by $N_<(i) \coloneqq \{j \in N(i) \mid j < i\}$ and $N_>(i) \coloneqq \{j \in N(i) \mid j > i\}$, respectively.

For a graph G with vertex set $[n]$ we define the *backward and forward degeneracy* as $d_<(G) \coloneqq \max\{|N_<(i)| \mid i \in [n]\}$ and $d_>(G) \coloneqq \max\{|N_>(i)| \mid i \in [n]\}$, respectively. Note that we have $d_<(G) = d_>(\mathrm{rev}(G))$ and $d_<(\mathrm{rev}(G)) = d_>(G)$, i.e., they only differ in reversing the order of vertices of G. In the literature there is only one notion of degeneracy, but in the runtime analysis of our algorithms both variants appear simultaneously, which is why we distinguish them. If the labeling of vertices of a graph G is not given, the smallest possible (forward) degeneracy of a labeling of G and a corresponding (forward) ordering can be computed by repeatedly removing a vertex with minimum degree. The maximum degree of G is denoted by $\Delta(G)$, and we clearly have $d_<(G) \leq \Delta(G)$ and $d_>(G) \leq \Delta(G)$.

Furthermore, we write $\mathcal{I}(G)$ for the collection of independent sets in G, i.e.,

$$\mathcal{I}(G) \coloneqq \{U \subseteq [n] \mid U \text{ independent set in } G\} = \{U \subseteq [n] \mid G[U] \text{ has no edges }\}.$$

For any graph $G = ([n], E)$ and any vertex $i \in [n]$, the set $\mathcal{I}(G)$ satisfies the recursion

$$\mathcal{I}(G) = \mathcal{I}(G - i) \cup \{J \cup \{i\} \mid J \in \mathcal{I}(G - N^+(i))\}. \tag{1}$$

The first set on the right hand side contains all independent sets that do not contain the vertex i, which are the same as independent sets in the graph obtained from G by removing the vertex i, and the second set contains all independent sets that contain the vertex i, which are the same as independent sets in the graph obtained from G by removing the closed neighborhood of i, plus the vertex i added.

Given an independent set $U \in \mathcal{I}(G)$, we write $x^U \in \{0,1\}^n$ for its *characteristic vector*, with $x^U_i = 1$ if $i \in U$ and $x^U_i = 0$ if $i \notin U$. We define $X(G) := \{x^U \mid U \in \mathcal{I}(G)\} \subseteq \{0,1\}^n$ for the set of all characteristic vectors of independent sets of G. We write $\mathscr{P}(G) := \operatorname{conv}(X(G)) = \operatorname{conv}(\{x^U \mid U \text{ independent set in } G\}) \subset \mathbb{R}^n$ for the *independent set polytope*, also known as stable set polytope, of G. The recursion (1) corresponds to intersecting the polytope $\mathscr{P}(G)$ with two hyperplanes $x_i = 0$ and $x_i = 1$, which yields two faces of $\mathscr{P}(G)$, namely the independent set polytopes $\mathscr{P}(G - i)$ and $\mathscr{P}(G - N^+(i))$; see Figure 1 (a). The following adjacency condition on the independent set polytope is well-known and has already been mentioned in the introduction. We denote the *symmetric difference* of two sets U, V by $U \triangle V := (U \setminus V) \cup (V \setminus U)$.

Lemma 1 ([Chv75])**.** *For any two distinct independent sets $U, V \in \mathcal{I}(G)$, the pair of vertices $\{x^U, x^V\}$ is an edge of the polytope $\mathscr{P}(G)$ if and only if $G[U \triangle V]$ is connected.*

For two independent sets $U, V \in \mathcal{I}(G)$, the transition $U \to V$ is called a *star exchange* if either $V = U \setminus \{k\}$ for some $k \in U$ or $V = (U \cup \{k\}) \setminus N(k)$ for some $k \notin U$, hence either a single vertex is removed from U, or a single vertex is added to U and its neighbors are removed from it. In both cases, $G[U \triangle V]$ is a star graph. In particular, such a transition corresponds to an edge of the polytope, but not every edge of the polytope is of this type; see Figure 1 (a).

For a string x and integer $k \geq 1$, we write x^k for the k-fold repetition of x. For a bit $b \in \{0,1\}$ we write $\bar{b}$ for its complement.

A listing $L = (x_1, \ldots, x_N)$ of binary strings $x_i \in \{0,1\}^n$ is called *genlex* if all strings with the same suffix appear consecutively in L. Equivalently, this means that all strings ending with 0 appear before all strings ending with 1, or vice versa, and this property is true recursively within each block obtained by ignoring the same last bit. A genlex listing of the set $\mathcal{I}(G)$ lists all elements of the first set on the right hand side of (1) (for $i = n$) before all elements of the second set, or vice versa. Thus, a genlex Hamilton path in $\mathscr{P}(G)$ is one that visits all vertices of the face $\mathscr{P}(G - n)$ before all vertices of the face $\mathscr{P}(G - N^+(n))$, flipping the last bit at most once, and this property is true recursively within each face obtained by projecting out the last coordinate; see Figure 1 (b)+(c).

2.2. **Basic algorithm.** Our algorithm for computing a Hamilton path on the independent set polytope is stated in pseudocode as Algorithm I. It takes as input a graph $G = ([n], E)$ and an initial independent set $\widetilde{x} \in X(G)$. One possible initialization that is valid for any graph is the empty set, that is, $\widetilde{x} = 0^n$. After the initialization in step I1, the algorithm repeatedly performs a loop consisting of steps I2–I5. The current independent set is stored in the variable $x \in \{0,1\}^n$, which is initialized in step I1, visited in step I2, and updated in step I4, before at the end of step I5 the control loops back to the next visit step. The *only* additional data structure used by the algorithm is a binary array $b = (b_1, \ldots, b_n)$ that is initialized in step I1, used in step I3 to compute the length k of the prefix of x that will be modified, and updated in step I5.

Algorithm I *(Independent sets by star exchanges on shortest prefixes).* For a graph $G = ([n], E)$, this algorithm computes a Hamilton path on the independent set polytope $\mathscr{P}(G)$, starting from an initial independent set $\widetilde{x} \in X(G)$.

I1. [Initialize] Set $x \leftarrow \widetilde{x}$ and $b \leftarrow 1^n$.
I2. [Visit] Visit x.
I3. [Shortest prefix change] Compute $k \leftarrow \min\{i \in [n] \mid b_i = 1 \wedge \sum_{j \in N_>(i)} x_j = 0\}$. Terminate if $k = \infty$.
I4. [Update x] Set $x_k \leftarrow \overline{x_k}$. If $x_k = 1$, then for all $i \in N_<(k)$ set $x_i \leftarrow 0$.
I5. [Update b] Set $b_k \leftarrow 0$, and for $i = 1, \ldots, k-1$ set $b_i \leftarrow 1$, then goto I2.

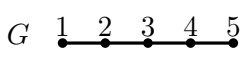
The idea of the algorithm is as follows. The goal is to compute a genlex listing of all independent sets of G, i.e., independent sets with the same suffix must appear consecutively. Thus, the algorithm greedily changes the current independent set x in such a way that a new independent set x' is reached, i.e., one that has not been visited before, such that x' agrees with x in the longest possible suffix, or equivalently, such that x' differs from x in the shortest possible prefix. The length k of the shortest possible prefix of x with this property is computed (somewhat magically) in step I3. Once we know k, all the remaining steps are trivial: Step I4 starts by complementing the kth bit in x. If $x_k = 1$ before the update, hence if vertex k is in the current independent set, then it is removed and we have $x_k = 0$ after the update. No further updates are needed, because the new set is clearly independent. On the other hand, if $x_k = 0$ before the update, hence if vertex k is not in the current independent set, then it is added and we have $x_k = 1$ after the update. Consequently, all neighbors of k in G that are in the independent set have to be removed in order to obtain a valid independent set again, which happens in the second part of step I4. Recall that $N(k) = N_<(k) \cup N_>(k)$, and since all vertices $j \in N_>(k)$ satisfy $x_j = 0$ by the definition of k in step I3, only the bits at positions $i \in N_<(k)$ have to be set to 0 (they may already be 0, in which case they do not change). We see that the next independent set differs from the previous one by a star exchange.

We now explain the magic of the auxiliary array b that allows us to compute which prefix changes lead to a new independent set, without actually keeping track of the visited ones (this would be prohibitively space- and time-expensive). Specifically, we have the following invariant: If $b_i = 0$, then all independent sets of G with suffix $\overline{x_i}, x_{i+1}, \ldots, x_n$ have been visited before. Therefore, prefixes of length i of x with $b_i = 0$ are not eligible to be modified. On the other hand, prefixes of length i with $b_i = 1$ are eligible, provided that all neighbors $j \in N_>(i)$ are not in the independent set (otherwise the change would extend beyond the prefix of length i). This justifies the computation of k in step I3 as the shortest possible prefix length that leads to a new independent set. In particular, if no such length exists ($k = \infty$) the algorithm terminates. It also explains the update in step I5, namely, b_k is toggled from $b_k = 1$ to $b_k = 0$, and all earlier b_i with $i < k$ are set to 1. The initialization $b \leftarrow 1^n$ in step I1 indicates that all prefixes are eligible to be modified, as no independent set has been visited yet.

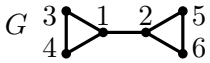
Figure 2 shows two runs of the algorithm on two different input graphs. We encourage the reader to work through these examples in order to digest and appreciate the working of the algorithm (the array s in the figure will be introduced later).

Theorem 2. *For every graph $G = ([n], E)$ and every initial independent set $\widetilde{x} \in X(G)$, Algorithm I computes a genlex Hamilton path on the independent set polytope $\mathscr{P}(G)$ starting at $\widetilde{x}$, where any two consecutive independent sets differ in a star exchange. The worst-case delay is $\mathcal{O}(n(d_>(G)+1)) = \mathcal{O}(n(\Delta(G)+1)) = \mathcal{O}(n^2)$.*

We see that choosing a good labeling of the vertices of the input graph is essential for obtaining a good runtime bound for the algorithm. As emphasized before, Algorithm I works for any (re)labeling of the vertices of the underlying graph. For graphs with bounded degeneracy, in particular with bounded maximum degree, the delay is linear in n.



	x	s	b	k
$\emptyset$	$\widetilde{x} = \underline{0}0000$	00000	11111	1
$\{1\}$	$\underline{1}0000$	00000	01111	2
$\{2\}$	$0\underline{1}000$	10000	10111	3
$\{3\}$	$00\underline{1}00$	01000	11011	1
$\{1,3\}$	$10\underline{1}00$	01000	01011	4
$\{1,4\}$	$\underline{1}00\underline{1}0$	00100	11101	1
$\{4\}$	$000\underline{1}0$	00100	01101	2
$\{2,4\}$	$0\underline{1}0\underline{1}0$	10100	10101	5
$\{2,5\}$	$0\underline{1}00\underline{1}$	10010	11110	2
$\{5\}$	$0000\underline{1}$	00010	10110	1
$\{1,5\}$	$\underline{1}000\underline{1}$	00010	00110	3
$\{1,3,5\}$	$\underline{1}0\underline{1}0\underline{1}$	01010	11010	1
$\{3,5\}$	$00\underline{1}0\underline{1}$	01010	01010	∞



	x	s	b	k
$\{2,4\}$	$\widetilde{x} = 0\underline{1}0\underline{1}00$	201000	111111	2
$\{4\}$	$000\underline{1}00$	101000	101111	4
$\emptyset$	$\underline{0}00000$	000000	111011	1
$\{1\}$	$\underline{1}00000$	000000	011011	2
$\{2\}$	$0\underline{1}0000$	100000	101011	3
$\{2,3\}$	$0\underline{1}1000$	200000	110011	2
$\{3\}$	$00\underline{1}000$	100000	100011	5
$\{3,5\}$	$00\underline{1}0\underline{1}0$	110000	111101	3
$\{5\}$	$000\underline{0}10$	010000	110101	1
$\{1,5\}$	$\underline{1}00\underline{0}10$	010000	010101	4
$\{4,5\}$	$000\underline{1}\underline{1}0$	111000	111001	6
$\{4,6\}$	$000\underline{1}0\underline{1}$	111010	111110	4
$\{6\}$	$000\underline{0}0\underline{1}$	010010	111010	1
$\{1,6\}$	$\underline{1}00\underline{0}0\underline{1}$	010010	011010	3
$\{3,6\}$	$00\underline{1}00\underline{1}$	110010	110010	∞

FIGURE 2. Protocol of Algorithms I and I* for two input graphs, one of them being the 5-vertex path from Figure 1, resulting in the Hamilton path shown there. Both algorithms produce the same results, and the auxiliary array s is only used within Algorithm I*. All toggled bits of x are underlined. Steps marked blue are those where x_k changes from 1 to 0, i.e., vertex k is removed from the independent set. Steps marked red are those where x_k changes from 0 to 1, i.e., vertex k is added to the independent set, and all of its (smaller) neighbors are removed.

Proof. We first establish the correctness of the algorithm, and then the bound on the running time.

When describing the algorithm before, we have already argued that it performs only star exchanges, and that every $x \in \{0,1\}^n$ visited indeed encodes an independent set of G. It remains to argue that the algorithm visits each independent set of G exactly once, hence that none of them is missed and none of them is visited repeatedly, and that the produced listing has the genlex property. This argument uses induction on n, the number of vertices of G. To settle the induction basis $n = 1$, consider the 1-vertex graph $G = ([1], \emptyset)$, which has two independent sets, namely $\mathcal{I}(G) = \{\emptyset, \{1\}\}$, encoded by $x = x_1 = 0$ and $x = x_1 = 1$. One can check that for either of the two possible initializations $\widetilde{x} = 0$ or $\widetilde{x} = 1$ the algorithm first visits $\widetilde{x}$, then its complement, and then terminates, as it should. This settles the induction basis.

For the induction step, we assume that $n \geq 2$, and let $\widetilde{x} \in X(G)$ be an arbitrary initial independent set of $G = ([n], E)$. We first consider the case that $\widetilde{x}_n = 0$. Observe that as long as the current independent set x satisfies $x_n = 0$, the algorithm operates as if the last bit of x was not present. This is because $x_n = 0$ does not contribute to the sum $\sum_{j \in N_{>(i)}} x_j$ in the definition of k (step I3). Thus, the algorithm behaves as it would on the input graph $G - n$, i.e., the graph obtained from G by removing the last vertex n. Therefore, by induction, Algorithm I will visit each independent set of $G - n$ exactly once, thus producing each independent set of G that does not contain the vertex n exactly once. Let x be the last independent set visited in step I2 in this first phase of the algorithm, just before the algorithm would terminate in step I3 if the bit b_n was not present. Now that the last bit $b_n = 1$ (recall step I1) is present, the algorithm does not terminate, but computes $k \leftarrow n$ as the shortest possible prefix to modify. Consequently,

x_n is flipped from 0 to 1, b_n is flipped from 1 to 0, and all $b_1, \dots, b_{n-1}$ are reset to 1^{n-1}. At the end of this iteration of the main loop x is the first independent set of G that contains the vertex n (to be visited after the goto). This is the beginning of the second phase of the algorithm. In the second phase, while $x_n = 1$, step I3 will never choose k as one of the neighbors $i \in N_<(n) = N(n)$ of n, and the corresponding bits of x will remain 0 and not be modified (also, the corresponding bits of b will remain 1 and not be modified). As a result, we can ignore those bits, and observe that the algorithm behaves as it would on the input graph $G - N^+(n)$, that is, the graph obtained from G by removing the closed neighborhood of n. Therefore, again by induction, Algorithm I will visit each independent set of $G - N^+(n)$ exactly once, thus producing each independent set of G that does contain the vertex n exactly once. By (1), Algorithm I thus visits each independent set of G exactly once. All $x \in X(G)$ with $x_n = 0$ are visited before all those with $x_n = 1$, and by induction this property holds recursively within each block obtained by ignoring the same last bit. It follows that the listing produced by the algorithm is indeed genlex.

To complete the correctness proof, note that the other case $\widetilde{x}_n = 1$ is proved analogously, by swapping the order of the two phases discussed before.

It remains to discuss the running time of the algorithm. We conveniently assume that the graph $G = ([n], E)$ is represented by adjacency lists. We write $m \coloneqq |E(G)|$ for the number of edges of G. Once in the beginning, we precompute the sets $N_<(i)$ and $N_>(i)$ for all $i \in [n]$ in time $\mathcal{O}(m)$. In addition, the initialization in step I1 takes time $\mathcal{O}(n)$. Using that $m \le n \cdot d_>(G)$ gives the bound $\mathcal{O}(n(d_>(G) + 1))$ for the time until the first vertex is visited (the $+1$ prevents that the product is 0 in the exceptional case that $d_>(G) = 0$).

Now consider the main loop. As $|N_<(k)| < n$ and $k \le n$, steps I4 and I5 can clearly be computed in time $\mathcal{O}(n)$. The bottleneck of the computation in the main loop is step I3. Indeed, each of the sums $\sum_{j \in N_>(i)} x_j$ in the minimization to determine k can be computed in time $d_>(G)$, and there are at most n such sums to compute, which yields the bound $\mathcal{O}(n(d_>(G) + 1))$. □

Note that in an actual implementation we do not need to compute the entire sums $\sum_{j \in N_>(i)} x_j$ in step I3, but can shortcut once the first non-zero summand x_j is encountered, which immediately makes the sum nonzero, and thus allows us to discard the corresponding index i in the minimization. However, this shortcut does not improve the theoretical guarantees proved before.

2.3. **Improved algorithm.** In the analysis before we noted that the bottleneck in the running time of Algorithm I is the minimization to determine k in step I3. This directly motivates the introduction of an auxiliary array of integers $s = (s_1, \dots, s_n)$, where

$$s_i \coloneqq \sum_{j \in N_>(i)} x_j \tag{2}$$

for $i \in [n]$. With those values at hand, the minimization simply becomes $\min\{i \in [n] \mid b_i = 1 \wedge s_i = 0\}$, which can be computed in linear time if the s_i are known. However, introducing the array s comes at the cost of updating it once any bits of x are modified. The required updates are straightforward, see lines I40 and I41 in the resulting pseudocode stated as Algorithm I* below.

Algorithm I* *(Independent sets by star exchanges on shortest prefixes).* For a graph $G = ([n], E)$, this algorithm computes a Hamilton path on the independent set polytope $\mathscr{P}(G)$, starting from an initial independent set $\widetilde{x} \in X(G)$.

I1. [Initialize] Set $x \leftarrow \widetilde{x}$, $s_i \leftarrow \sum_{j \in N_>(i)} x_j$ for $i = 1, \dots, n$, and $b \leftarrow 1^n$.

I2. [Visit] Visit x.
I3. [Shortest prefix change] Compute $k \leftarrow \min\{i \in [n] \mid b_i = 1 \wedge s_i = 0\}$. Terminate if $k = \infty$.
I4. [Update x and s] Set $x_k \leftarrow \overline{x_k}$.
  I40. [Flipped $1 \to 0$] If $x_k = 0$, then for all $i \in N_<(k)$ set $s_i \leftarrow s_i - 1$.
  I41. [Flipped $0 \to 1$] If $x_k = 1$, then for all $i \in N_<(k)$ set $s_i \leftarrow s_i + 1$, and if $x_i = 1$ set $x_i \leftarrow 0$ and for all $j \in N_<(i)$ also set $s_j \leftarrow s_j - 1$.
I5. [Update b] Set $b_k \leftarrow 0$, and for $i = 1, \ldots, k-1$ set $b_i \leftarrow 1$, then goto I2.

Note that the initialization of s in step I1 can be simplified to $s \leftarrow 0^n$ if the initial independent set $\widetilde{x} = 0^n$ is used.

Theorem 3. *For every graph $G = ([n], E)$ and every initial independent set $\widetilde{x} \in X(G)$, Algorithm I* computes the same Hamilton path as Algorithm I. After the initialization in time $\mathcal{O}(n(d_<(G)+1))$, the worst-case delay is $\mathcal{O}(n + d_<(G)^2) = \mathcal{O}(n + \Delta(G)^2) = \mathcal{O}(n^2)$. The amortized delay is $\mathcal{O}(n)$.*

We see again that choosing a good labeling of the vertices of the input graph is important. However, while Algorithm I benefits from small forward degrees in the vertex ordering of G, captured by the forward degeneracy $d_>(G)$, Algorithm I* benefits from small backward degrees, captured by the backward degeneracy $d_<(G)$. Importantly, the amortized delay of Algorithm I*, which matters most, is linear in n, regardless of the degrees in G.

An experimental comparison of both algorithms on a variety of input graphs, exploring in particular the issue of vertex orderings, is presented in Section 4.

Proof. The first part of the statement follows from the correctness of Algorithm I and the definition (2) of the entries of the array s.

We now argue about the claimed delay bounds. We first consider the worst-case delay. Precomputing the sets $N_<(i)$ and $N_>(i)$ for all $i \in [n]$ and the initialization in step I1 can be done in time $\mathcal{O}(m)$. Using that $m \leq n \cdot d_<(G)$ gives the bound $\mathcal{O}(n(d_<(G)+1))$ for the time until the first vertex is visited. Now consider the main loop. Clearly, steps I3, I40 and I5 can be computed in time $\mathcal{O}(n)$. Step I41 contains a double loop that takes time $\mathcal{O}(d_<(G)^2)$. Adding these contributions proves the delay bound.

We now consider the amortized delay. The time $\mathcal{O}(n(d_<(G)+1))$ until the first vertex is visited can be amortized to $\mathcal{O}(d_<(G)) = \mathcal{O}(n)$ by the observation that G has at least n distinct independent sets (containing exactly one vertex only). It remains to argue about the amortization inside the double loop of step I41. Whenever a 0-bit at position k in x is toggled to a 1-bit, the outer loop of step I41 ('for all $i \in N_<(k)$ set $s_i \leftarrow s_i + 1$') is executed. We can thus, at no asymptotic cost, prepay costs for the 1-bit x_k in the amount of $|N_<(k)|$: Think about placing 1 coin on each edge going from k to a smaller neighbor. Now observe that that the inner loop in step I41 ('for all $j \in N_<(i)$ also set $s_j \leftarrow s_j - 1$') is executed only for positions $i \in [n]$ for which $x_i = 1$. These are exactly the costs that have been prepaid: Think about taking back the previously placed coins from the edges going from i to a smaller neighbor while iterating through the loop. The 1-bits in the initial string $\widetilde{x}$ are prepaid for in the initialization of s in step I1. The amortized cost of one iteration of the main loop is therefore $\mathcal{O}(n + d_<(G)) = \mathcal{O}(n)$. □

3. Applications

3.1. **Independent sets of special families of graphs.** We first apply our algorithms to compute independent sets of special families of graphs, which recovers several Gray codes known from the literature.

We need the following definitions. For any sequence of bitstrings $L = (x_1, \dots, x_N)$, we write $\mathrm{rev}(L) \coloneqq (x_N, x_{N-1}, \dots, x_1)$ for the reverse sequence. Furthermore, for any two bitstrings u, v we define $uLv \coloneqq (ux_1v, ux_2v, \dots, ux_Nv)$, where the prefix u is prepended and suffix v is appended to every bitstring in L. For a bitstring x and integer $n \geq 1$ we write $[x]_n$ for the length n substring of the left-concatenation of infinitely many copies of x. In other words, $[x]_n$ is obtained by prepending x with more copies of x until all n positions are filled, truncating the extra bits at the beginning. For example, we have $[100]_5 = 00100$ and $[100]_{10} = 0100100100$.

We write E_n for the graph with vertex set $[n]$ and no edges. Clearly, we have $\mathcal{I}(E_n) = 2^{[n]}$, i.e., every possible subset of $[n]$ is independent. The listing of independent sets obtained when running Algorithms I and I* with input graph E_n and initial independent set $\widetilde{x} = 0^n$ is the well-known *binary reflected Gray code*, defined inductively by

$$B_1 = 0, 1 \quad \text{and} \quad B_n \coloneqq B_{n-1}0, \mathrm{rev}(B_{n-1})1 \text{ for } n \geq 2;$$

see Figure 3 (a). I.e., we obtain a listing of all bitstrings of length n, such that any two consecutive strings differ in a single bit. In the following, we use the term *1-Gray code* for such a listing of bitstrings. In terms of independent sets, this further restricts the allowed exchanges to removing or adding only a single vertex in every step (i.e., it is never necessary to remove the neighbors of an added vertex from the independent set).

We write P_n and C_n for the path graph and cycle graph with vertex set $[n]$, with vertices labeled $1, \dots, n$ along the path or cycle, respectively. Clearly, independent sets of P_n and C_n are subsets of $[n]$ that contain no two consecutive vertices or no two cyclically consecutive vertices, respectively; in the latter case vertex 1 and vertex n are also considered consecutive. The corresponding characteristic vectors have been studied in the pattern-avoidance community. Specifically, *Fibonacci words* are bitstrings that avoid two consecutive 1s, and *Lucas words*

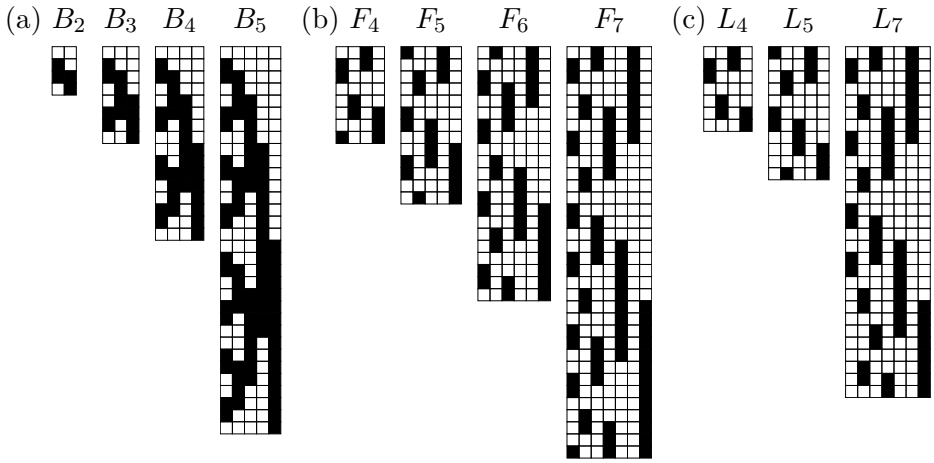


Figure 3. 1-Gray codes produced by Algorithms I and I* for various special input graphs (0-bits=white squares, 1-bits=black squares), namely (a) empty graphs E_2, E_3, E_4, E_5 (binary reflected Gray code), (b) path graphs P_4, P_5, P_6, P_7 (Fibonacci words), and (c) cycle graphs C_4, C_5, C_7 (Lucas words).

are bitstrings that avoid two cyclically consecutive 1s. Parametrizing by the length n of the words, they are counted by the Fibonacci numbers and Lucas numbers, respectively (OEIS sequences A000045 and A000032).

We inductively define a listing F_n of all Fibonacci words of length n by

$$F_1 := 0, 1 \text{ and } F_n := \mathrm{rev}(F_{n-1})0, \mathrm{rev}(F_{n-2})01 \text{ for } n \geq 2; \tag{3}$$

see Figure 3 (b) (cf. [Vaj01] and [MS05]). Similarly, for all n that are not multiples of 3 we define a listing L_n of all Lucas words of length n by

$$L_1 := 0, 1, L_2 := 10, 00, 01 \text{ and } L_n := \mathrm{rev}(F_{n-1})0, 0\,\mathrm{rev}(F_{n-3})01 \text{ for } n \geq 4 \text{ with } 3 \nmid n; \tag{4}$$

see Figure 3 (c) (cf. [BV05]). Maybe slightly unexpectedly, the definition of L_n recurses to F_{n-1} and F_{n-3}, *not* to L_{n-1} and L_{n-3}. Using these definitions, one can check easily by induction that both listings F_n and L_n are genlex 1-Gray codes, where F_n starts with $[010]_n$ and ends with $[001]_n$, and L_n starts with $[010]_n$ and ends with $0[001]_{n-1}$. We note that Figure 1 (c) also shows a listing of Fibonacci words of length 5, but the initial bitstring is different and in some steps, more than one bit changes, so the listing is not a 1-Gray code.

Theorem 4. *Running Algorithms I or I* with input graph $G = P_n$ for $n \geq 1$ or $G = C_n$ for $n \geq 1$ with $3 \nmid n$ and initial independent set $\widetilde{x} := [010]_n$ yields the genlex 1-Gray codes F_n and L_n of Fibonacci and Lucas words defined in* (3) *and* (4)*, respectively.*

Proof. The following auxiliary claim can be proved by a straightforward induction on n: For any graph $G = ([n], E)$ and any $\widetilde{x} \in X(G)$, there is a unique genlex listing of $X(G)$ by star exchanges starting at $\widetilde{x}$. Flipping a single bit is clearly a star exchange, so the 1-Gray codes F_n and L_n are genlex listings by star exchanges. From Theorem 2 and 3 and the auxiliary claim it follows that Algorithms I and I* compute exactly those listings. □

3.2. **Vertex Covers.** A *vertex cover* of a graph $G = ([n], E)$ is a subset of vertices $U \subseteq [n]$, such that every edge of G is incident with at least one vertex in U. It is easy to see that $U \subseteq [n]$ is a vertex cover if and only if its complement $[n] \setminus U$ is an independent set. Consequently, the *vertex cover polytope* $\overline{\mathscr{P}}(G) := \mathrm{conv}\{x^U \mid U \text{ vertex cover of } G\}$ is obtained by applying the reflection $x \mapsto 1 - x$ to the independent set polytope $\mathscr{P}(G)$, and thus both polytopes have the same combinatorial structure. Both Algorithm I and Algorithm I* can be used to compute a Hamilton path on the vertex cover polytope $\overline{\mathscr{P}}(G)$, by computing the complement $\overline{x}$ just before every visit step. Both algorithms compute the same genlex Hamilton path on the vertex cover polytope $\overline{\mathscr{P}}(G)$ within the time bounds guaranteed by Theorems 2 and 3, where any two consecutive vertex covers differ in a complemented star exchange, i.e., either a single vertex is added to the cover, or a vertex is removed and its neighbors are added.

3.3. **Matchings.** Matchings of a graph are independent sets in its line graph, so we can apply our algorithms to compute a Hamilton path on the corresponding polytope. We proceed to translate our algorithm to this setting.

A *matching* in a graph G is a subset of edges of G, no two of which have a vertex in common. Given a graph G, its *line graph* $L(G)$ has a vertex for every edge of G, and an edge between any two vertices corresponding to edges of G that are incident to each other. Note that the line graph $L(G)$ is *claw-free*, that is, it has no induced star with three rays. This expresses the fact that an edge $\{u, v\}$ in G can be incident with at most two other edges that have no end vertices in common with each other (one incident with u and one incident with v). We observe that matchings in G are in one-to-one correspondence with independent sets in the line graph $L(G)$.

We write $\mathcal{M}(G)$ for the set of all matchings of G and $X(G)$ for the corresponding set of characteristic vectors, i.e., $X(G) := \{x^M \mid M \in \mathcal{M}(G)\}$. The *matching polytope* is defined $\mathscr{M}(G) := \operatorname{conv}(X(G)) = \operatorname{conv}(\{x^M \mid M \text{ matching in } G\})$ [Chv75]. By Lemma 1, for any two distinct matchings $M, N \in \mathcal{M}(G)$, the pair of vertices $\{x^M, x^N\}$ is an edge of the polytope $\mathscr{M}(G)$ if and only if the symmetric difference $M \triangle N$ is a single path or cycle.

We consider a graph $G = ([n], E)$ with m edges. We formalize the labeling of the edges of G by integers $1, \ldots, m$ by two mappings $\alpha, \beta : [m] \to [n]$, where $\alpha(i)$ and $\beta(i)$ are the two end vertices of the edge with label i. We thus refer to the pair of mappings α, β as an *edge labeling*. We translate Algorithm I to the setting of matchings as follows: The array $x = (x_1, \ldots, x_m)$ is the characteristic vector of the current matching, and the array $b = (b_1, \ldots, b_m)$ has the same meaning as before. In addition, we introduce an array $e = (e_1, \ldots, e_n)$ where $e_i \in [m] \cup \{0\}$, with the following meaning: $e_j = i \in [m]$ means that vertex j is incident with edge i of the matching, whereas $e_j = 0$ means that vertex $j \in [n]$ is not incident with any edge of the matching; see Figure 4. With these data structures, we obtain Algorithm M stated below. The condition $(x_i = 1 \vee \max\{e_{\alpha(i)}, e_{\beta(i)}\} < i)$ in line M2 is satisfied if $x_i = 1$, i.e., edge i is currently in the matching, or if $x_i = 0$, i.e., edge i is not currently in the matching, and both matching edges incident to either $\alpha(i)$ or $\beta(i)$ have smaller labels (this includes the case that there is no such edge, because then the special label 0 is used). The star exchanges of Algorithm I thus translate to either removing an edge from the matching or adding an edge and removing at most two other edges. Formally, for two matchings $M, N \in \mathcal{M}(G)$, the transition $M \to N$ is called a $\{1,2,3\}$-*exchange*, if either $N = M \setminus \{e\}$ for some $e \in M$ or $N = (M \cup \{e\}) \setminus F$ for some $e \notin M$ and subset $F \subseteq M$ such that e shares an endpoint with every edge $f \in F$ (thus $|F| \leq 2$). In both cases, $M \triangle N$ is a path of length at most 3.

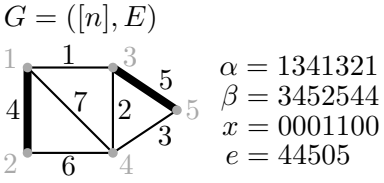


FIGURE 4. Data structures used in Algorithm M. Matching edges are bold.

Algorithm M *(Matchings by shortest prefix exchanges).* For an m-edge graph $G = ([n], E)$ with an edge labeling $\alpha, \beta \in [n]^{[m]}$, this algorithm computes a Hamilton path on the matching polytope $\mathscr{M}(G)$, starting from an initial matching $x \in X(G)$.

M1. [Initialize] Set $x \leftarrow \widetilde{x}$, $b \leftarrow 1^m$ and $e \leftarrow 0^n$. For $i = 1, \ldots, m$, if $x_i = 1$ set $e_{\alpha(i)} \leftarrow i$ and $e_{\beta(i)} \leftarrow i$.

M2. [Visit] Visit x.

M3. [Shortest prefix change] Compute $k \leftarrow \min\{i \in [m] \mid b_i = 1 \wedge (x_i = 1 \vee \max\{e_{\alpha(i)}, e_{\beta(i)}\} < i)\}$. Terminate if $k = \infty$.

M4. [Update x and e] Set $x_k \leftarrow \overline{x_k}$.

M40. [Flipped $1 \to 0$] If $x_k = 0$, then set $e_{\alpha(k)} \leftarrow 0$ and $e_{\beta(k)} \leftarrow 0$.

M41. [Flipped $0 \to 1$] If $x_k = 1$, then for each $j \in \{\alpha(k), \beta(k)\}$, if $e_j > 0$ set $x_{e_j} \leftarrow 0$, $e_{\alpha(e_j)} \leftarrow 0$ and $e_{\beta(e_j)} \leftarrow 0$. Then set $e_j \leftarrow k$.

M5. [Update b] Set $b_k \leftarrow 0$, and for all $i = 1, \ldots, k-1$ set $b_i \leftarrow 1$, then goto M2.

Theorem 5. *For every m-edge graph $G = ([n], E)$ and every initial matching $\widetilde{x} \in X(G)$, Algorithm M computes a genlex Hamilton path on the matching polytope $\mathscr{M}(G)$ starting at $\widetilde{x}$, where any two consecutive matchings differ in a $\{1,2,3\}$-exchange. The worst-case delay is $\mathcal{O}(m+n)$.*

Proof. The correctness of the algorithm follows from Theorem 2, and the running time can be seen directly. □

3.4. **Antichains and ideals.** Antichains of a poset are independent sets in its comparability graph, so we can apply our algorithms to compute a Hamilton path on the corresponding polytope. We now set up the required notation to translate all notions and results to the setting of posets.

Given a poset $P = ([n], <)$, we write $D(i) := \{j \in [n] \mid j \leq i\}$ and $U(i) := \{j \in [n] \mid i \leq j\}$ for the *downset of i* and *upset of i*, respectively. We also define the *proper downset* and *proper upset* as $D_<(i) := D(i) \setminus \{i\} = \{j \in [n] \mid j < i\}$ and $U_>(i) := U(i) \setminus \{i\} = \{j \in [n] \mid i < j\}$, respectively. The *comparability graph* of P is the graph with vertex set $[n]$ and an edge $\{i, j\}$ if either $i < j$ or $j < i$ in P, i.e., it has edges between all comparable pairs of elements in the poset. We say that P is *connected* if its comparability graph is connected. For a subset $A \subseteq [n]$ we write $P[A]$ for the subposet induced by A, consisting of all elements in A with the comparabilities given by P.

A *chain* or *total order* is a poset in which every two elements are comparable. An *antichain* is a poset in which no two elements are comparable. For a poset P, we write $\mathcal{A}(P)$ for the set of all antichains in P and $X(P)$ for the corresponding set of characteristic vectors, i.e., $X(P) := \{x^A \mid A \in \mathcal{A}(P)\}$. Note that antichains in P are in one-to-one correspondence with independent sets in the comparability graph of P. The *chain polytope* of P is defined as $\mathscr{C}(P) := \operatorname{conv}(X(P)) = \operatorname{conv}(\{x^A \mid A \text{ antichain in } P\})$ [Sta86][2]. By Lemma 1, for any two distinct antichains $A, B \in \mathcal{A}(P)$, the pair of vertices $\{x^A, x^B\}$ is an edge of the polytope $\mathscr{C}(P)$ if and only if the subposet $P[A \triangle B]$ is connected. In particular, a star exchange between two independent sets in the comparability graph corresponds to either removing an element from the antichain, or adding an element and removing its proper downset or proper upset.

A *linear extension* of a poset $P = ([n], <_P)$ is a total order $L = ([n], <_L)$ that extends P, hence such that if $i <_P j$ then $i <_L j$. We can think of a linear extension as a permutation on $[n]$ such that if $i <_P j$ then the value i appears before the value j in the permutation. We denote the identity permutation by $\mathrm{id}_n := (1, 2, \ldots, n)$.

As mentioned in Section 1.2, our algorithms work for any labeling of the ground set. In the context of posets, it is convenient to choose a labeling that is a linear extension. Specifically, we refer to $P = ([n], <)$ as an *upward* poset if it has the identity permutation $\mathrm{id}_n = (1, 2, \ldots, n)$ as a linear extension. Clearly, any poset can be (re)labeled in this way, by numbering the elements from the bottom up. With this labeling, the set of left neighbors $N_<(i)$ and the set of right neighbors $N_>(i)$ in the comparability graph simply become the proper downset $D_<(i)$ and proper upset $U_>(i)$, respectively. Consequently, the entries of the array s defined in (2) have the

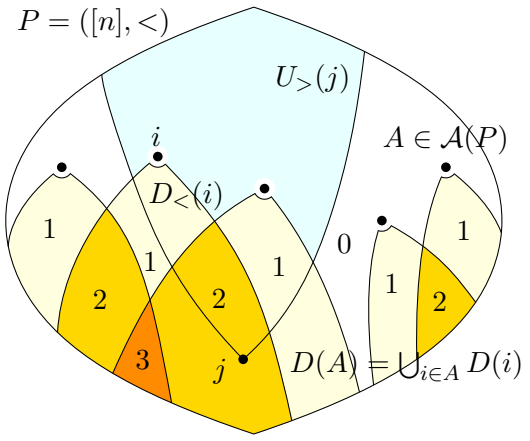
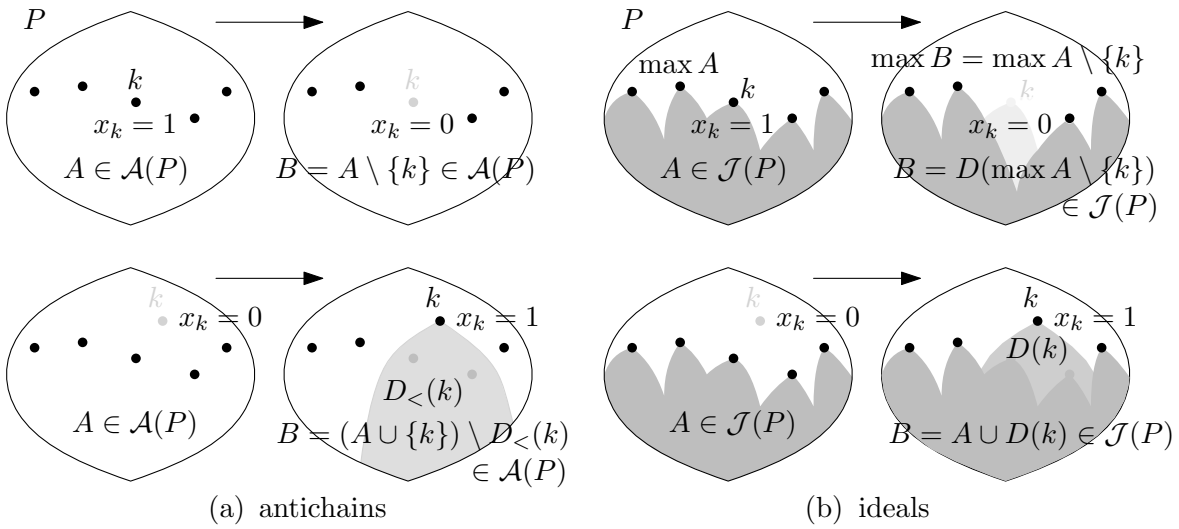


FIGURE 5. An antichain A and its corresponding ideal $D(A)$. The numbers in the colored regions indicate the number of antichain elements in the upset of an element, which are stored in the variables s_i (cf. (5)).

[2]The motivation for the name chain polytope (instead of antichain polytope) is that its hyperplane representation is based on chains in the poset. However, in our context the vertex representation is more convenient.



(a) antichains (b) ideals

FIGURE 6. Star exchanges for (a) antichains and (b) ideals of posets.

interpretation

$$s_i = \sum_{j \in U_>(i)} x_j = |U_>(i) \cap A|, \tag{5}$$

where $A \in \mathcal{A}(P)$ denotes the antichain whose characteristic vector is x, i.e., $x = x^A$. In words, s_i counts the number of elements of the antichain A in the proper upset of i, or equivalently, the number of elements of A that contain i in their proper downset; see Figure 5. Furthermore, for an upward poset, a star exchange between two independent sets in the comparability graph corresponds to either removing an element from the antichain, or adding an element and removing its proper downset (the operation of removing the proper upset does not occur). Thus, for two antichains $A, B \in \mathcal{A}(P)$, the transition $A \to B$ is called a *star exchange* if either $B = A \setminus \{k\}$ for some $k \in A$ or $B = (A \cup \{k\}) \setminus D_<(k)$ for some $k \notin A$; see Figure 6 (a).

Algorithm A* *(Antichains by star exchanges on shortest prefixes).* For an upward poset $P = ([n], <)$, this algorithm computes a Hamilton path on the chain polytope $\mathscr{C}(P)$, starting from an initial antichain $\widetilde{x} \in X(P)$.

A1. [Initialize] Set $x \leftarrow \widetilde{x}$, $s_i \leftarrow \sum_{j \in U_>(i)} x_j$ for $i = 1, \ldots, n$, and $b \leftarrow 1^n$.

A2. [Visit] Visit x.

A3. [Shortest prefix change] Compute $k \leftarrow \min\{i \in [n] \mid b_i = 1 \wedge s_i = 0\}$. Terminate if $k = \infty$.

A4. [Update x and s] Set $x_k \leftarrow \overline{x_k}$.

I40. [Flipped $1 \to 0$] If $x_k = 0$, then for all $i \in D_<(k)$ set $s_i \leftarrow s_i - 1$.

I41. [Flipped $0 \to 1$] If $x_k = 1$, then for all $i \in D_<(k)$ set $s_i \leftarrow s_i + 1$, and if $x_i = 1$ set $x_i \leftarrow 0$ and for all $j \in D_<(i)$ also set $s_j \leftarrow s_j - 1$.

A5. [Update b] Set $b_k \leftarrow 0$, and for $i = 1, \ldots, k-1$ set $b_i \leftarrow 1$, then goto A2.

The algorithm obtained by translating Algorithm I* to the poset setting is stated as Algorithm A*. Of course, the more basic but slower variant analogous to Algorithm I, without the array s, also exists, but is omitted.

The initialization of s in step A1 can be simplified to $s \leftarrow 0^n$ if the initial antichain $\widetilde{x} = 0^n$ is used.

From Theorems 2 and 3 we thus derive the following result.

Theorem 6. *For every upward poset $P = ([n], <)$ and every initial antichain $\widetilde{x} \in X(P)$, Algorithm A* computes a genlex Hamilton path on the chain polytope $\mathscr{C}(P)$ starting at $\widetilde{x}$, where any two consecutive antichains differ in a star exchange. The amortized delay is $\mathcal{O}(n)$.*

For the amortized delay stated in the theorem we assume that the poset $P = ([n], <)$ is represented as a directed graph, with separate adjacency lists for the in- and out-neighbors of each element $i \in [n]$, corresponding to all elements of $D_<(i)$ and $U_>(i)$, respectively.

Given a poset $P = ([n], <)$, an element $i \in [n]$ is *maximal* if there is no $j \in [n]$ with $i < j$. An *ideal* is a downward closed set $A \subseteq [n]$, hence such that if $i \in A$ and $j < i$ then we have $j \in A$. We write $\mathcal{J}(P)$ for the set of all ideals of P and $Y(P)$ for the corresponding set of characteristic vectors, i.e., $Y(P) \coloneqq \{x^A \mid A \in \mathcal{J}(P)\}$. There is a one-to-one correspondence between antichains and ideals in P, namely, the maximal elements of an ideal form an antichain, and conversely, the union of downsets of all elements in an antichain forms an ideal. Given an ideal $A \in \mathcal{J}(P)$, we write $\max A$ for the antichain determined by the maximal elements of A. Conversely, given an antichain $A \in \mathcal{A}(P)$, we write $D(A) \coloneqq \bigcup_{i \in A} D(i)$ for the ideal generated by the antichain A.

The *order polytope* of P is defined as $\mathscr{O}(P) \coloneqq \operatorname{conv}(Y(P)) = \{x^A \mid A \text{ ideal in } P\}$ [Sta86][3]. For two distinct ideals $A, B \in \mathcal{J}(P)$, the pair of vertices $\{x^A, x^B\}$ is an edge of the polytope $\mathscr{O}(P)$ if and only if $P[A \triangle B]$ is a connected subposet [Sta86]. Note that this condition implies $A \subsetneq B$ or $B \subsetneq A$. Due to the one-to-one correspondence between antichains and ideals, both polytopes $\mathscr{C}(P)$ and $\mathscr{O}(P)$ have the same number of vertices. In general, edges and higher-dimensional faces of both polytopes are different (see [HLSS17, Ex. 1.2]). Nonetheless, edges on the order polytope coming from star exchanges do translate to edges on the chain polytope, as follows: We say that two ideals $A, B \in \mathcal{J}(P)$ differ in a *star exchange* if the corresponding antichains $\max A$ and $\max B$ differ in a star exchange. In other words, a transition $A \to B$ is a star exchange if either

$$B = D(\max A \setminus \{k\}) = A \setminus \{i \in D(k) \mid U(i) \cap \max A = \{k\}\}$$

for some $k \in \max A$, or $B = A \cup D(k)$; see Figure 6 (b).

Using the aforementioned correspondence between antichains and ideals, we can thus modify Algorithm A* and compute the ideals corresponding to the antichains along the way. The resulting Hamilton paths on the chain polytope are in one-to-one correspondence to the Hamilton paths on the order polytope computed by Algorithm A*.

Theorem 7. *For every upward poset $P = ([n], <)$ and every initial ideal $\widetilde{y} \in Y(P)$, the modified Algorithm A* computes a genlex Hamilton path on the order polytope $\mathscr{O}(P)$ starting at $\widetilde{y}$, where any two consecutive ideals differ in a star exchange. The amortized delay is $\mathcal{O}(n)$.*

Proof. We add an extra variable $y \in \{0,1\}^n$ that stores the characteristic vector of the current ideal. All computations are done with the corresponding antichain $x \in X(P)$, and the variable y is only used in the initialization step and the visit step. Specifically, we replace line A1 by:

A1. [Initialize] For $i = 1, \ldots, n$, set $x_i \leftarrow 1$ if $\widetilde{y}_i = 1$ and $\sum_{j \in U_>(i)} \widetilde{y}_j = 0$, and $x_i \leftarrow 0$ otherwise. Also set $s_i \leftarrow \sum_{j \in U_>(i)} x_j$ for $i = 1, \ldots, n$, and $b \leftarrow 1^n$.

The initialization of x and s can be simplified to $x \leftarrow 0^n$ and $s \leftarrow 0^n$ if the initial ideal $\widetilde{y} = 0^n$ is used.

[3]Stanley considers upper ideals (upward closed), whereas we consider lower ideals (downward closed). However, the two notions are equivalent by turning the poset upside down.

We observe that the ideal $D(A)$ of an antichain $A \in \mathcal{A}(P)$ satisfies

$$D(A) = \bigcup_{i \in A} D(i) = A \cup \bigcup_{i \in A} D_<(i) = A \cup \{i \in [n] \mid |U_>(i) \cap A| > 0\}.$$

From (5) we see that the quantity $|U_>(i) \cap A|$ for the current antichain A, i.e., for $x \in X(P)$ with $x = x^A$, is stored in the variable s_i; recall Figure 5. Thus, we replace line A2 by:

A2. [Visit] For $i = 1, \ldots, n$, set $y_i \leftarrow 1$ if $x_i = 1$ or $s_i > 0$, and $y_i \leftarrow 0$ otherwise. Visit y.

The listing of ideals produced by this modified algorithm is genlex, because $n \in A$ implies that $n \in D(A)$, and $n \notin A$ implies that $n \notin D(A)$.

These additional computations in the visit step take time $\mathcal{O}(n)$, so they do not affect the $\mathcal{O}(n)$ amortized delay bound. □

4. Experimental comparison of Algorithms I and I*

In this section, we aim to understand how well the bounds on the (amortized) delay established by Theorems 2 and 3 reflect the 'running time in practice' when using Algorithms I and I* to compute independent sets for a variety of input graphs. In particular, we aim to measure the effect of the vertex ordering of an input graph on the running time. For this we implemented both algorithms in C++, incorporating the shortcut discussed after the proof of Theorem 2.

The results of the experiments are depicted in the four plots shown in Figure 7. The quantity D/n displayed on the vertical axes is defined as follows: For each of the Algorithms I (top row) or Algorithm I* (bottom row) and a given input graph $G = ([n], E)$, we count the total number C of read and write accesses to the arrays x and b, or x, s and b, respectively. The visit step does not contribute to this count. Thus, the number C is a proxy for the total work performed by the algorithm for computing the set $\mathcal{I}(G)$ of all independent sets of G, which is independent of the choice of implementation language and computing power. Then the average delay is simply

$$D := \frac{C}{|\mathcal{I}(G)|}, \tag{6}$$

which is the work performed per independent set. Thus, the quantity D/n displayed in the figures is the average delay per bit of the vector x that encodes each independent set.

The horizontal axis in all plots shows the average degree $\overline{d}(G) := 2m/n$, where $m = |E|$ is the number of edges of G. This quantity was chosen because it is independent of the vertex ordering, and yet roughly corresponds to the forward degeneracy $d_<(G)$ and backward degeneracy $d_>(G)$, because of the lower bounds $d_<(G), d_>(G) \geq m/n = \overline{d}(G)/2$, which are reasonably tight for graphs that are nearly regular.

The two plots in the left column are for sparse graphs with average degree in the interval $[0, 5]$, and the two plots in the right column are for dense graphs with average degree in the interval $[5, 115]$. The ranges of values on the vertical axes are the same in the top and bottom row, to allow direct comparison between the algorithms, but they are drastically different in the left and right column, to provide a more fine-grained picture for sparse graphs.

The following graphs were chosen as input graphs for both algorithms.

- As in Section 3.1 before, E_n denotes the graph with n vertices and no edges, and P_n and C_n denote the path and cycle with n vertices.
- Q_n denotes the n-dimensional hypercube with vertex set $\{0, 1\}^n$ and an edge between any two bitstrings that differ in exactly one bit.
- $K_{a,b,c}$ denotes the complete tripartite graph with vertex sets of sizes a, b, c.

- $\mathrm{Kg}_{n,k}$ denotes the Kneser graph whose vertices are all k-element subsets of $[n]$, with edges between disjoint sets.
- $\mathrm{Ci}_{60,13}$ denotes the 60-vertex circulant graph with the 13-element set of displacement values $\{1, 3, 5, 6, 7, 8, 9, 13, 15, 17, 18, 19, 20\}$.
- $G_{n,p}$ denotes the binomial random graph with n vertices and edge probability p.

For each input graph G, we consider the following five different orderings of its vertex set.

- `min`: vertex ordering obtained by repeatedly removing a vertex of minimum degree from G; this is the ordering that minimizes the forward degeneracy $d_>(G)$.
- `revmin`: reverse of `min`; this is the ordering that minimizes the backward degeneracy $d_<(G)$.
- `max`: vertex ordering obtained by repeatedly removing a vertex of maximum degree from G.
- `revmax`: reverse of `max`.
- `rand`: average of 10 uniformly random vertex orderings.

4.1. **Evaluation.** For the smallest and sparsest graphs, Algorithm I has a small advantage, likely because it does not have to maintain the extra array s, which is useless in this case, whereas for all other graphs, Algorithm I* is faster, and the advantage of the extra array s pays off. Most importantly, as the average degree grows, the average delay per bit remains constant for Algorithm I*, as predicted by Theorem 3, but grows linearly for Algorithm I, as predicted by Theorem 2.

As can be seen for empty graphs with 5, 10 and 25 vertices, paths and cycles with 5, 10 and 40 vertices, and hypercubes of dimension 3, 4 and 5, the average delay per bit decreases considerably as the number of vertices increases. To understand this, it helps to consider the family of empty graphs E_n, and the correspondence to the binary reflected Gray code discussed in Section 3.1 before. In this case, the worst-case delay bound of Theorem 2 and the average delay bound of Theorem 3, which are both linear in n, are overly pessimistic, as the average delay is in fact constant. Specificially, for E_n the average delay is given by the average length of the prefixes that change in the binary reflected Gray code, which can be shown to be $2 - o(1) < 2$. Consequently, when dividing by n, we get a quantity that vanishes with n. More generally, for very sparse graphs that have a lot of independent sets, most changes occur on very short prefixes of x, leading to a sublinear average delay for both Algorithm I and Algorithm I*, and thus the quantity D/n decreases as n grows.

The effect of vertex orderings can be seen most clearly for Algorithm I when running it on the relatively dense complete tripartite graphs and random graphs. Namely, the algorithm is fastest for the `min` ordering, as expected and predicted by Theorem 2. The `revmax` ordering comes second, as it places large degree vertices at the end, which consequently leaves small degree vertices in the beginning (this is what matters most). The `rand` ordering and `revmin` ordering come next. The `max` ordering is clearly worst, as it places large degree vertices at the beginning. The difference in running time between the best and worst ordering is substantial and grows with the average degree, emphasizing the necessity for choosing a good ordering.

A more fine-grained comparison of the `min` and `revmin` ordering for Algorithm I* shows the following. While Theorem 3 suggests that the delay of Algorithm I* depends primarily on the backward degeneracy $d_<(G)$, which is minimized by the `revmin` ordering, experiments with more irregular graphs, such as $K_{a,b,c}$ where $a < b < c$ are such that $b - a, c - b \gg 1$, show that actually the `min` ordering is by a factor of approximately 2 faster. This somewhat counterintuitive behavior can be explained as follows: While the forward and backward degeneracy minimize the maximum degree in either forward or backward direction, what is more important for the running time of the algorithms is that vertices with small labels have small degrees. These are

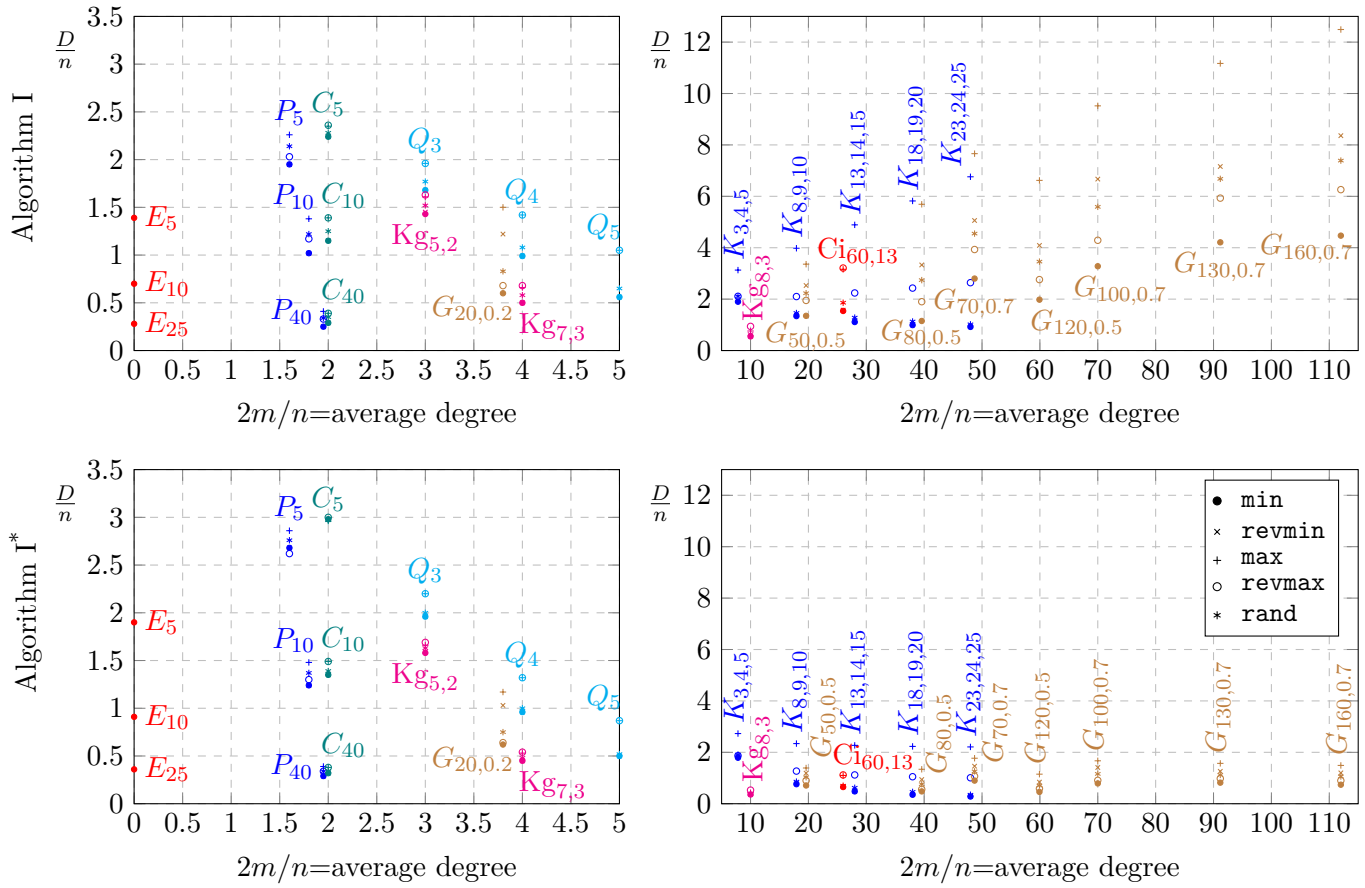


FIGURE 7. Experimental comparison of Algorithm I (top row) and Algorithm I* (bottom row) on a variety of input graphs. See (6) for the definition of the quantity plotted on the vertical axes. The legend at the bottom right applies to all four plots, showing the marker styles used for the five different vertex orderings.

the vertices for which the neighborhoods are examined most frequently, typically exponentially more often than for vertices with large labels, as a consequence of the genlex ordering of the independent sets. Therefore, while the `min` ordering places vertices with small forward degree first, this also seems to yield relatively small backward degrees in the beginning. On the other hand, the `revmin` ordering places vertices with small backward degrees last, leaving vertices with potentially larger degrees in the beginning. Therefore, in practice it might be best to always use the `min` ordering for both algorithms.

REFERENCES

[BF26] S. Brenner and J. Fink. Constant delay Gray code enumeration of ideals and antichains in posets. `https://arxiv.org/abs/2603.28118`, 2026.

[BV05] J.-L. Baril and V. Vajnovszki. Minimal change list for Lucas strings and some graph theoretic consequences. *Theoret. Comput. Sci.*, 346(2–3):189–199, 2005.

[Chv75] V. Chvátal. On certain polytopes associated with graphs. *J. Combinatorial Theory Ser. B*, 18:138–154, 1975.

[CN85] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[FHM+26] J. Fink, P. Hladík, A. Merino, O. Mička, and T. Mütze. Faster and simpler traversal of 0/1-polytopes. `https://arxiv.org/abs/2607.17590`, 2026.

[HLSS17] T. Hibi, N. Li, Y. Sahara, and A. Shikama. The numbers of edges of the order polytope and the chain polytope of a finite partially ordered set. *Discrete Math.*, 340(5):991–994, 2017.

[HMNS01] M. Habib, R. Medina, L. Nourine, and G. Steiner. Efficient algorithms on distributive lattices. *Discrete Appl. Math.*, 110(2-3):169–187, 2001.

[HNS97] M. Habib, L. Nourine, and G. Steiner. Gray codes for the ideals of interval orders. *J. Algorithms*, 25(1):52–66, 1997.

[JYP88] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Inform. Process. Lett.*, 27(3):119–123, 1988.

[KMNF92] T. Kashiwabara, S. Masuda, K. Nakajima, and T. Fujisawa. Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc graph. *J. Algorithms*, 13(1):161–174, 1992.

[KR93] Y. Koda and F. Ruskey. A Gray code for the ideals of a forest poset. *J. Algorithms*, 15(2):324–340, 1993.

[KWUA21] K. Kurita, K. Wasa, T. Uno, and H. Arimura. A constant amortized time enumeration algorithm for independent sets in graphs with bounded clique number. *Theoret. Comput. Sci.*, 874:32–41, 2021.

[MM24] A. Merino and T. Mütze. Traversing combinatorial 0/1-polytopes via optimization. *SIAM J. Comput.*, 53(5):1257–1292, 2024.

[MS05] K. Mikawa and I. Semba. An $O(1)$ time algorithm for generating Fibonacci strings. *Electronics and Communications in Japan, Part 2*, 88(9):67–72, 2005.

[MU04] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Algorithm theory—SWAT 2004*, volume 3111 of *Lecture Notes in Comput. Sci.*, pages 260–272. Springer, Berlin, 2004.

[Müt23] T. Mütze. Combinatorial Gray codes—an updated survey. *Electron. J. Combin.*, DS26(Dynamic Surveys):93 pp., 2023.

[NP84] D. J. Naddef and W. R. Pulleyblank. Hamiltonicity in (0-1)-polyhedra. *J. Combin. Theory Ser. B*, 37(1):41–52, 1984.

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

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

[Sta86] R. P. Stanley. Two poset polytopes. *Discrete Comput. Geom.*, 1(1):9–23, 1986.

[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.

[Uno01] T. Uno. A fast algorithm for enumerating bipartite perfect matchings. In *Algorithms and computation (Christchurch, 2001)*, volume 2223 of *Lecture Notes in Comput. Sci.*, pages 367–379. Springer, Berlin, 2001.

[Vaj01] V. Vajnovszki. A loopless generation of bitstrings without p consecutive ones. In *Combinatorics, computability and logic (Constanţa, 2001)*, Springer Ser. Discrete Math. Theor. Comput. Sci., pages 227–240. Springer, London, 2001.