Efficient computation of middle levels Gray codes[*]

Torsten Mütze[†]

School of Mathematics
Georgia Institute of Technology
30332 Atlanta GA, USA
muetze@math.gatech.edu

Jerri Nummenpalo

Department of Computer Science
ETH Zürich
8092 Zürich, Switzerland
njerri@inf.ethz.ch

ABSTRACT. For any integer $n \geq 1$ a *middle levels Gray code* is a cyclic listing of all bitstrings of length $2n+1$ that have either n or $n+1$ entries equal to 1 such that any two consecutive bitstrings in the list differ in exactly one bit. The question whether such a Gray code exists for every $n \geq 1$ has been the subject of intensive research during the last 30 years, and has been answered affirmatively only recently [T. Mütze. Proof of the middle levels conjecture. *Proc. London Math. Soc.*, 112(4):677–713, 2016]. In this work we provide the first efficient algorithm to compute a middle levels Gray code. For a given bitstring, our algorithm computes the next ℓ bitstrings in the Gray code in time $\mathcal{O}(n\ell(1 + \frac{n}{\ell}))$, which is $\mathcal{O}(n)$ on average per bitstring provided that $\ell = \Omega(n)$.

KEYWORDS: Gray code, middle levels conjecture

1. INTRODUCTION

Efficiently generating all objects in a particular combinatorial class (e.g. permutations, combinations, partitions or trees) in such a way that each object is generated exactly once is one of the oldest and most fundamental problems in the area of combinatorial algorithms, and such generation algorithms are used as core building blocks in a wide range of practical applications. The survey [Sav97] lists numerous references. A classical example is the so-called binary *Gray code* [Gra53], which lists all 2^n bitstrings of length n such that any two consecutive bitstrings differ in exactly one bit. A straightforward implementation of this algorithm takes time $\mathcal{O}(n)$ to compute from a given bitstring the next one in the list (see Algorithm G in [Knu11, Section 7.2.1.1]), which can be improved to $\mathcal{O}(1)$ [Ehr73, BER76] (see Algorithm L in [Knu11, Section 7.2.1.1]). The space requirement of both algorithms is $\mathcal{O}(n)$. Similar minimum-change generation algorithms have been developed for various other combinatorial classes. We exemplarily cite four examples from the excellent survey [Sav97] on this topic: (1) listing all permutations of $\{1, 2, \ldots, n\}$ so that consecutive permutations differ only by the swap of one pair of adjacent elements [Joh63, Tro62] (see also [Sed77]), (2) listing all k-element subsets of an n-element set such that consecutive sets differ only by exchanging one element [Ehr73, BER76, EHR84, EM84, Rus88], (3) listing all binary trees with n vertices so that consecutive trees differ only by one rotation operation [Luc87, LvBR93], (4) listing all spanning trees of a graph such that consecutive trees differ only by exchanging one edge [Cum66, HH72].

Coming back to Gray codes, we say that a bitstring of length n has *weight* k, if it has exactly k entries equal to 1 and $n - k$ entries equal to 0. Furthermore, for any integer $n \geq 1$ we define a *middle levels Gray code* as a cyclic listing of all bitstrings of length $2n + 1$ that have weight n or $n + 1$ such that any two consecutive bitstrings in the list differ in exactly one bit. The name 'middle levels' becomes clear when considering the relevant bitstrings as subsets in the Hasse diagram of the subset inclusion

lattice. Clearly, a middle levels Gray code has to visit $N := \binom{2n+1}{n} + \binom{2n+1}{n+1} = 2\binom{2n+1}{n} = 2^{\Theta(n)}$ many bitstrings in total, and the weight of the bitstrings will alternate between n and $n+1$ in every step. The existence of a middle levels Gray code for every value of n is asserted by the infamous *middle levels conjecture*, which originated probably with Havel [Hav83] and Buck and Wiedemann [BW84], but has also been attributed to Dejter, Erdős, Trotter [KT88] and various others. It also appears as Exercise 56 in Knuth's book [Knu11, Section 7.2.1.3] and in the popular book [DG12]. This conjecture also became known as *revolving door conjecture* for the following reason: Imagine a set of $2n+1$ people, split into two subsets of size n and $n+1$ that are separated by a revolving door. The conjecture asks whether it is possible to move in each step one person from the larger group through the revolving door to the other side, such that every way to split the people into two subsets of size n and $n+1$ is encountered exactly once. It may come as a surprise that establishing the existence of a middle levels Gray code appears to be a difficult problem, given that by item (2) above one can easily come up with a listing of all bitstrings of length $2n+1$ with weight exactly n such that any two consecutive bitstrings differ in *two* bits.

The middle levels conjecture has attracted considerable attention over the last 30 years [Sav93, FT95, SW95, Joh04, DSW88, KT88, DKS94, HKRR05, GŠ10]. Until recently, middle levels Gray codes had only been found with brute-force computer searches for $n \leq 19$ [SS99, SSS09, SA11]. For $n = 19$ this Gray code already consists of $N = 137.846.528.820$ bitstrings. A complete proof of the conjecture has only been announced very recently.

Theorem 1 ([Müt16])**.** *A middle levels Gray code exists for every $n \geq 1$.*

1.1. **Our results.** Even though the proof of Theorem 1 given in [Müt16] is constructive, a straight-forward implementation takes exponential (in n) time and space to compute for a given bitstring the next one in the middle levels Gray code. Essentially, we need to compute and store the entire list of $N = 2^{\Theta(n)}$ bitstrings. The main contribution of this paper is a time- and space-efficient algorithm to compute a middle levels Gray code for every $n \geq 1$, which can be considered an algorithmic proof of Theorem 1. Specifically, given any bitstring of length $2n+1$ with weight n or $n+1$, our algorithm computes the next ℓ bitstrings in the Gray code in time $\mathcal{O}(n\ell(1 + \frac{n}{\ell}))$, which is $\mathcal{O}(n)$ on average per bitstring provided that $\ell = \Omega(n)$. For most bitstrings the worst-case running time is $\mathcal{O}(n)$, for few it is $\mathcal{O}(n^2)$. The space requirement of our algorithm is $\mathcal{O}(n)$.

An implementation of this algorithm in C++ can be found on the authors' websites [www], and we invite the reader to experiment with this code. We used it to compute a middle levels Gray code for $n = 19$ in less than a day on an ordinary desktop computer. For comparison, the above-mentioned intelligent brute-force computation for $n = 19$ from [SA11] took about 164 days using comparable hardware.

Remark 2. Clearly, the ultimate goal would be a generation algorithm with a worst-case running time of $\mathcal{O}(1)$ per bitstring, but this would require substantial new ideas that would in particular yield a much simpler proof of Theorem 1 than the one presented in [Müt16]. In fact, in a recent paper [MN17] that appeared after the submission of this manuscript, we were able to extend the ideas and techniques presented in this paper, and to improve our algorithm to achieve an optimal $\mathcal{O}(1)$ worst-case running time per generated bitstring. This improved algorithm builds on top of the algorithm presented in this paper.

Remark 3. It was shown in [Müt16] that there are in fact double-exponentially (in n) many different middle levels Gray codes, which is easily seen to be best possible. This raises the question whether our algorithm can be parametrized to compute any of these Gray codes. While this is possible in principle, choosing between doubly-exponentially many different Gray codes would require a parameter of exponential size, spoiling the above-mentioned runtime and space bounds. Moreover, it would introduce a substantial amount of additional complexity in the description and correctness

proof of the algorithm. To avoid all this, our algorithm computes only one particular 'canonical' middle levels Gray code. We will briefly come back to this point at the end of Section 3 (Remark 10).

1.2. **Outline of this paper.** In Section 2 we present the pseudocode of our middle levels Gray code algorithm. In Section 3 we prove the correctness of this algorithm, and in Section 4 we discuss how to implement it to achieve the claimed runtime and space bounds.

2. The Algorithm

It is convenient to reformulate our problem in graph-theoretic language: To this end we define the *middle levels graph*, denoted by $Q_{2n+1}(n, n+1)$, as the graph whose vertices are all bitstrings of length $2n+1$ that have weight n or $n+1$, with an edge between any two bitstrings that differ in exactly one bit. In other words, bitstrings that may appear consecutively in the middle levels Gray code correspond to neighboring vertices in the middle levels graph. Clearly, computing a middle levels Gray code is equivalent to computing a Hamilton cycle in the middle levels graph. Throughout the rest of this paper we talk about middle levels Gray codes using this graph-theoretic terminology.

Our algorithm to compute a Hamilton cycle in the middle levels graph, i.e., to compute a middle levels Gray code, is inspired by the constructive proof of Theorem 1 given in [Müt16]. Efficiency is achieved by reformulating this inductive construction as a recursive procedure. Even though the description of the algorithm in the present paper is completely self-contained and illustrated with several figures that highlight the main ideas, the reader may find it useful to first read an informal overview of the proof of Theorem 1, which can be found in [Müt16, Section 1.2].

Roughly speaking, our algorithm consists of a lower level function that computes sets of disjoint paths in the middle levels graph and several higher level functions that combine these paths to form a Hamilton cycle. In the following we explain these functions from bottom to top. Before doing so we introduce some notation that will be used throughout this paper.

2.1. **Basic definitions.** *Composition of mappings.* We write the composition of mappings f, g as $f \bullet g$, where $(f \bullet g)(x) := f(g(x))$.

Reversing/inverting and concatenating bitstrings. We define $\bar{0} := 1$ and $\bar{1} := 0$. For any bitstring x we let $\overline{\mathrm{rev}}(x)$ denote the bitstring obtained from x by reversing the order of the bits and inverting every bit. Moreover, for any bitstring $x = (x_1, x_2, \ldots, x_{2n})$ we define $\pi(x) := (x_1, x_3, x_2, x_5, x_4, \ldots, x_{2n-1}, x_{2n-2}, x_{2n})$, i.e., except the first and last bit, all adjacent pairs of bits are swapped. E.g., for $x = 11001110$ we have $\overline{\mathrm{rev}}(x) = 10001100$ and $\pi(x) = 10110110$. Here and throughout this paper, we omit commas and brackets when writing strings of 0's and 1's. Note that the mappings $\overline{\mathrm{rev}}$, π and consequently also $\overline{\mathrm{rev}} \bullet \pi$ are self-inverse, and that $\overline{\mathrm{rev}} \bullet \pi = \pi \bullet \overline{\mathrm{rev}}$. For two bitstrings x and y we denote by $x \circ y$ the concatenation of x and y. For any graph G whose vertices are bitstrings and any bitstring y we denote by $G \circ y$ the graph obtained from G by attaching y to every vertex of G.

Layers of the cube. For $n \geq 0$ and k with $0 \leq k \leq n$, we denote by $B_n(k)$ the set of all bitstrings of length n with weight k. For $n \geq 1$ and k with $0 \leq k \leq n-1$, we denote by $Q_n(k, k+1)$ the graph with vertex set $B_n(k) \cup B_n(k+1)$, with an edge between any two bitstrings that differ in exactly one bit.

Oriented paths, first/second/last vertices. An oriented path P in a graph is a path with a particular orientation, i.e., we distinguish its first and last vertex. For an oriented path $P = (v_1, v_2, \ldots, v_\ell)$ we define its first, second and last vertex as $F(P) := v_1$, $S(P) := v_2$ and $L(P) := v_\ell$, respectively.

Bitstrings and Dyck paths. We often identify a bitstring x with a lattice path in the integer lattice $\mathbb{Z}^2$ as follows, see the left hand side of Figure 2: Starting at the coordinate $(0, 0)$, we read the bits

of x from left to right and interpret every 1-bit as an upstep that changes the current coordinate by $(+1, +1)$ and every 0-bit as a downstep that changes the current coordinate by $(+1, -1)$. Specifically, we define the following sets of lattice paths:

- For any $n \geq 0$ and $k \geq 0$ we denote by $D_n(k)$ the set of lattice paths with k upsteps and $n - k$ downsteps (n steps in total) that never move below the line $y = 0$.
- For $n \geq 1$ and $k \geq 0$ we define $D_n^{>0}(k) \subseteq D_n(k)$ as the set of lattice paths that have no point of the form $(x, 0)$, $1 \leq x \leq n$, and $D_n^{=0}(k) \subseteq D_n(k)$ as the set of lattice paths that have at least one point of the form $(x, 0)$, $1 \leq x \leq n$. For $n = 0$ we define $D_0^{=0}(0) := \{()\}$, where () denotes the empty lattice path, and $D_0^{>0}(0) := \emptyset$. We clearly have $D_n(k) = D_n^{=0}(k) \cup D_n^{>0}(k)$.
- Furthermore, for $n \geq 1$ and $k \geq 0$ we let $D_n^-(k)$ denote the set of lattice paths with k upsteps and $n - k$ downsteps (n steps in total) that have exactly one point of the form $(x, -1)$, $1 \leq x \leq n$.

Note that all these lattice paths end at the coordinate $(n, 2k - n)$.

For example, we have

$$D_6^{=0}(3) = \{111000, 110100, 110010, 101010, 101100\} \ ,$$

$$D_6^{>0}(4) = \{111100, 111010, 111001, 110110, 110101\} \ ,$$

$$D_6^-(3) = \{110001, 101001, 100110, 011100, 011010\} \ .$$

It is well known that in fact $|D_{2n}^{=0}(n)| = |D_{2n}^{>0}(n+1)| = |D_{2n}^-(n)|$ and that the size of these sets is given by the n-th Catalan number (see [Che08] and [Sta99]). Depending on the values of n and k the above-mentioned sets of lattice paths might be empty, e.g., we have $D_{2n}^{>0}(n) = \emptyset$. Observe furthermore that the mappings $\overline{\mathrm{rev}}$, π and therefore also $\overline{\mathrm{rev}} \bullet \pi$ map each of the sets $D_{2n}^{=0}(n)$ and $D_{2n}^-(n)$ onto itself (see [Müt16, Lemma 11] for a formal proof). E.g., we have $\overline{\mathrm{rev}}(\pi(111000)) = 111000$, $\overline{\mathrm{rev}}(\pi(110100)) = 101010$, $\overline{\mathrm{rev}}(\pi(110010)) = 110010$, $\overline{\mathrm{rev}}(\pi(101010)) = 110100$, $\overline{\mathrm{rev}}(\pi(101100)) = 101100$ and therefore $\overline{\mathrm{rev}}(\pi(D_6^{=0}(3))) = D_6^{=0}(3)$.

2.2. **Computing paths in $Q_{2n}(n, n+1)$.** The algorithm PATHS() is at the core of our Hamilton cycle algorithm. Its description is given in Algorithm 1. For simplicity let us ignore for the moment the parameter $\texttt{flip} \in \{\texttt{true}, \texttt{false}\}$ and assume that it is set to $\texttt{false}$. Then for every n and k with $1 \leq n \leq k \leq 2n - 1$ the algorithm PATHS() constructs a set of disjoint oriented paths $\mathcal{P}_{2n}(k, k+1)$ in the graph $Q_{2n}(k, k+1)$ in the following way: Given a vertex $x \in Q_{2n}(k, k+1)$ and the parameter $\texttt{dir} \in \{\texttt{prev}, \texttt{next}\}$, the algorithm computes a neighbor of x on the path that contains the vertex x. The parameter $\texttt{dir}$ controls the search direction, so for $\texttt{dir} = \texttt{prev}$ we obtain the neighbor of x that is closer to the first vertex of the path, and for $\texttt{dir} = \texttt{next}$ the neighbor that is closer to the last vertex of the path. If x is a first vertex or a last vertex of one of the paths — we will see momentarily what these are — then the result of a call to PATHS() with $\texttt{dir} = \texttt{prev}$ or $\texttt{dir} = \texttt{next}$, respectively, is undefined and such calls will not be made from the higher level functions.

The algorithm PATHS() works recursively: For the base case of the recursion $n = 1$ (lines P1–P2) it computes neighbors for the set of paths

$$\mathcal{P}_2(1, 2) := \{(10, 11, 01)\} \ , \tag{1}$$

which consists only of a single path on three vertices. For example, the result of PATHS$(1, 1, 10, \texttt{next}, \texttt{false})$ is 11 and the result of PATHS$(1, 1, 11, \texttt{prev}, \texttt{false})$ is 10. For the recursion step the algorithm considers the last two bits of the current vertex x (see line P5) and, depending on their values (see lines P9, P11, P14 and P19), either flips one of these two bits (see lines P12, P15, P16, P17, P20, P21), or recurses to flip one of the first $2n - 2$ bits instead, leaving the last two bits unchanged (see lines P7, P10, P13, P18, P22). As already mentioned, the algorithm PATHS() is essentially a recursive formulation of the inductive construction of paths described in [Müt16] (and recapitulated in Section 3.1), and the different cases in the algorithm reflect the different cases in this construction. The recursion

step in line P22 is where the mappings $\overline{\text{rev}}$ and π introduced in Section 2.1 come into play (recall that $\overline{\text{rev}}^{-1} = \overline{\text{rev}}$ and $\pi^{-1} = \pi$). To give another example, the result of $\text{PATHS}(3, 3, 111000, \texttt{next}, \texttt{false})$ is 111001, and the result of $\text{PATHS}(3, 3, 111001, \texttt{prev}, \texttt{false})$ is 111000, so the set of paths $\mathcal{P}_6(3, 4)$ in the graph $Q_6(3, 4)$ defined by the algorithm $\text{PATHS}()$ contains a path with the edge $(111000, 111001)$.

Algorithm 1: $\text{PATHS}(n, k, x, \texttt{dir}, \texttt{flip})$

Input: Integers n and k with $1 \le n \le k \le 2n - 1$, a vertex $x \in Q_{2n}(k, k+1)$, parameters $\texttt{dir} \in \{\texttt{prev}, \texttt{next}\}$ and $\texttt{flip} \in \{\texttt{true}, \texttt{false}\}$

Output: A neighbor of x in $Q_{2n}(k, k+1)$

P1 **if** $n = 1$ **then**   /* base cases */
P2 $\quad$ depending on x and $\texttt{dir}$, **return** the previous/next neighbor of x on $\mathcal{P}_2(1, 2)$ defined in (1)

P3 **else if** $n = k = 2$ and $\texttt{flip} = \texttt{true}$ **then**
P4 $\quad$ depending on x and $\texttt{dir}$, **return** the previous/next neighbor of x on $\widetilde{\mathcal{P}}_4(2, 3)$ defined in (3)

P5 Split $x = (x_1, x_2, \ldots, x_{2n})$ into $x^- := (x_1, x_2, \ldots, x_{2n-2})$ and $x^+ := (x_{2n-1}, x_{2n})$
P6 **if** $k \ge n + 1$ **then**
P7 $\quad$ **return** $\text{PATHS}(n - 1, k - x_{2n-1} - x_{2n}, x^-, \texttt{dir}, \texttt{flip}) \circ x^+$

P8 **else**   /* $k = n$ */
P9 $\quad$ **if** $x^+ = 10$ **then**
P10 $\quad\quad$ **return** $\text{PATHS}(n - 1, n - 1, x^-, \texttt{dir}, \texttt{flip}) \circ x^+$

P11 $\quad$ **else if** $x^+ = 00$ **then**
P12 $\quad\quad$ **if** $x^- \in D_{2n-2}^{>0}(n)$ **then return** $x^- \circ 01$
P13 $\quad\quad$ **else return** $\text{PATHS}(n - 1, n, x^-, \texttt{dir}, \texttt{flip}) \circ x^+$

P14 $\quad$ **else if** $x^+ = 01$ **then**
P15 $\quad\quad$ **if** $x^- \in D_{2n-2}^{=0}(n - 1)$ **then return** $x^- \circ 11$
P16 $\quad\quad$ **else if** $x^- \in D_{2n-2}^{-}(n - 1)$ and $\texttt{dir} = \texttt{next}$ **then return** $x^- \circ 11$
P17 $\quad\quad$ **else if** $x^- \in D_{2n-2}^{>0}(n)$ and $\texttt{dir} = \texttt{prev}$ **then return** $x^- \circ 00$
P18 $\quad\quad$ **else return** $\text{PATHS}(n - 1, n - 1, x^-, \texttt{dir}, \texttt{false}) \circ x^+$

P19 $\quad$ **else if** $x^+ = 11$ **then**
P20 $\quad\quad$ **if** $x^- \in D_{2n-2}^{=0}(n - 1)$ and $\texttt{dir} = \texttt{next}$ **return** $x^- \circ 01$
P21 $\quad\quad$ **else if** $x^- \in D_{2n-2}^{-}(n - 1)$ and $\texttt{dir} = \texttt{prev}$ **then return** $x^- \circ 01$
P22 $\quad\quad$ **else return** $\overline{\text{rev}}\left(\pi\left(\text{PATHS}(n - 1, n - 1, \overline{\text{rev}}^{-1}(\pi^{-1}(x^-)), \overline{\texttt{dir}}, \texttt{flip})\right)\right) \circ x^+$ where
$\quad\quad\quad$ $\overline{\texttt{dir}} := \texttt{prev}$ if $\texttt{dir} = \texttt{next}$ and $\overline{\texttt{dir}} := \texttt{next}$ otherwise

---

We will later prove that the set of paths $\mathcal{P}_{2n}(n, n + 1)$ computed by the algorithm $\text{PATHS}()$ (with parameters $k = n$ and $\texttt{flip} = \texttt{false}$) has the following properties:

(i) All paths in $\mathcal{P}_{2n}(n, n+1)$ are disjoint, and together they visit all vertices of the graph $Q_{2n}(n, n+1)$.
(ii) The sets of first and last vertices of the paths in $\mathcal{P}_{2n}(n, n+1)$ are $D_{2n}^{=0}(n)$ and $D_{2n}^{-}(n)$, respectively.

We only state these properties here; the proof will be provided in Section 3. From (ii) we conclude that the number of paths in $\mathcal{P}_{2n}(n, n + 1)$ equals the n-th Catalan number. E.g., the entire set of paths $\mathcal{P}_4(2, 3)$ computed by the algorithm $\text{PATHS}()$ is

$$\mathcal{P}_4(2, 3) = \{P, P'\} \ , \tag{2a}$$

$$P := (1100, 1101, 0101, 0111, 0011, 1011, 1001) \ , \tag{2b}$$

$$P' := (1010, 1110, 0110) \ . \tag{2c}$$

P_1	P_2	P_3	P_4	P_5
111000	110100	110010	101010	101100
111001	110101	110110	111010	111100
011001	010101	010110	011010	011100
011011	011101	011110		
010011	001101	001110		
010111	101101	101110		
000111	100101	100110		
001111	100111			
001011	100011			
101011	110011			
101001	110001			

TABLE 1. The five paths in the set $\mathcal{P}_6(3,4)$ computed by the algorithm PATHS() with parameter `flip = false`. Each path is listed in a separate column, the first vertex from the set $D_6^{=0}(3)$ is at the top, and the last vertex from the set $D_6^-(3)$ is at the bottom. These paths are disjoint and together visit all vertices of the graph $Q_6(3,4)$.

R_1	R_2	R_3	R_4
111000	110100	110010	101010
111001	110101	111010	101110
011001	010101	011010	001110
011011	011101		011110
001011	001101		010110
001111	101101		110110
000111	100101		100110
010111	100111		
010011	100011		
110011	101011		
110001	101001		

TABLE 2. The four paths in the set $\widetilde{\mathcal{P}}_6(3,4)$ computed by the algorithm PATHS() with parameter `flip = true`. Comparing those paths to Table 1, note that (P_1, P_2) and (P_3, P_4) are flippable pairs of paths and the corresponding flipped pairs of paths are (R_1, R_2) and (R_3, R_4), respectively. Note that the paths in $\widetilde{\mathcal{P}}_6(3,4)$ do not visit all vertices of $Q_6(3,4)$ (unlike the paths in $\mathcal{P}_6(3,4)$), as the vertices of P_5 from Table 1 are not covered.

To give another example, the set of paths $\mathcal{P}_6(3,4)$ computed by the algorithm PATHS() consists of the five paths in Table 1. We remark that the length of each of the paths in $\mathcal{P}_{2n}(n, n+1)$ depends only on the length of the first 'hill' of the lattice path corresponding to the first vertex. In particular, not all paths have the same length. The precise length formula will be given in Section 4, when we analyze the running time of our algorithm. For the correctness of the algorithm the lengths of the paths are irrelevant. We will see in the next section how to combine the paths $\mathcal{P}_{2n}(n, n+1)$ in the graph $Q_{2n}(n, n+1)$ computed by our algorithm PATHS() (called with `flip = false`) to compute a Hamilton cycle in the middle levels graph $Q_{2n+1}(n, n+1)$. One crucial ingredient we need for this is another set of paths $\widetilde{\mathcal{P}}_{2n}(n, n+1)$, computed by calling the algorithm PATHS() with `flip = true`. We shall see that the core 'intelligence' of our Hamilton cycle algorithm consists of cleverly combining some paths from $\mathcal{P}_{2n}(n, n+1)$ and some paths from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ to a Hamilton

cycle in the middle levels graph. We will see that from the point of view of all top-level routines that call the algorithm PATHS() only the paths $\mathcal{P}_{2n}(k, k+1)$ and $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ for $k = n$ are used, but the recursion clearly needs to compute these sets also for all other values $k = n+1, n+2, \ldots, 2n-1$.

Specifically, calling the algorithm PATHS() with `flip = true` yields a set of paths $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ that differs from $\mathcal{P}_{2n}(k, k+1)$ as follows: First note that for certain inputs of the algorithm the value of the parameter `flip` is irrelevant, and the computed paths are the same regardless of its value. This is true whenever the recursion ends in the base case in lines P1-P2. One such example is the path P_5 in Table 1. Computing previous or next neighbors for any vertex on this path yields the same result regardless of the value of `flip`. Such paths are not part of $\widetilde{\mathcal{P}}_{2n}(k, k+1)$, and we ignore them in the following. The remaining paths from $\mathcal{P}_{2n}(k, k+1)$ can be grouped into pairs and for every such pair (P, P') the set $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ contains two disjoint paths R and R' that visit the same set of vertices as P and P', but that connect the end vertices of the paths the other way: Formally, denoting by $V(G)$ the vertex set of any graph G, these last two conditions can be captured as $V(P) \cup V(P') = V(R) \cup V(R')$, $F(P) = F(R)$, $F(P') = F(R')$, $L(P) = L(R')$ and $L(P') = L(R)$.

We refer to a pair (P, P') satisfying these conditions as a *flippable pair* of paths, and to the corresponding paths (R, R') as a *flipped pair*. This notion of flippable/flipped pairs is extremely valuable for designing our Hamilton cycle algorithm, as it allows the algorithm to decide *independently* for each flippable pair of paths, whether to follow one of the original paths or the flipped paths in the graph $Q_{2n}(n, n+1)$.

The base case for the recursive computation of $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ is the set of paths $\widetilde{\mathcal{P}}_4(2, 3)$ in the graph $Q_4(2, 3)$ defined by

$$\widetilde{\mathcal{P}}_4(2, 3) := \{R, R'\} \ , \tag{3a}$$

$$R := (1100, 1110, 0110) \ , \tag{3b}$$

$$R' := (1010, 1011, 0011, 0111, 0101, 1101, 1001) \tag{3c}$$

(see lines P3–P4). Observe that (P, P') defined in (2) and (R, R') satisfy precisely the conditions for flippable/flipped pairs of paths, i.e., (P, P') is a flippable pair and (R, R') is a corresponding flipped pair of paths. To continue also the previous example, the set of paths $\widetilde{\mathcal{P}}_6(3, 4)$ computed by our algorithm is shown in Table 2.

2.3. Computing a Hamilton cycle in the middle levels graph.

The algorithm HAMCYCLENEXT() uses the paths $\mathcal{P}_{2n}(n, n+1)$ and $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ in the graph $Q_{2n}(n, n+1)$ computed by the algorithm PATHS() as described in the previous section to compute for a given vertex x the next vertex on a Hamilton cycle in the middle levels graph $Q_{2n+1}(n, n+1)$. To simplify the exposition of the algorithm, let us ignore for the moment the parameter `flip` $\in \{$`true`, `false`$\}$ and assume that it is set to `false`, and let us also ignore line N3. With these simplifications only paths from $\mathcal{P}_{2n}(n, n+1)$ are considered, and those from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ are ignored. Then instead of computing a Hamilton cycle the algorithm HAMCYCLENEXT() computes several smaller cycles that together visit all vertices of the middle levels graph. We will correct this later by setting `flip` accordingly.

The algorithm HAMCYCLENEXT() is based on the following decomposition of the middle levels graph $Q_{2n+1}(n, n+1)$, see Figure 1: By partitioning the vertices of the graph $Q_{2n+1}(n, n+1)$ according to the value of the last bit, we observe that it consists of a copy of the graph $Q_{2n}(n, n+1) \circ 0$ and a copy of the graph $Q_{2n}(n-1, n) \circ 1$ plus the set of edges $M_{2n+1} = \{(x \circ 0, x \circ 1) \mid x \in B_{2n}(n)\}$ along which the last bit is flipped. Observe furthermore that the graphs $Q_{2n}(n, n+1)$ and $Q_{2n}(n-1, n)$ are isomorphic, and that the mapping $\overline{\text{rev}}$ is an isomorphism between these graphs. It is easy to check that this isomorphism preserves the sets of end vertices of the paths $\mathcal{P}_{2n}(n, n+1)$: Using property (ii) mentioned in the previous section, we have $\overline{\text{rev}}(D_{2n}^{=0}(n)) = D_{2n}^{=0}(n)$ and $\overline{\text{rev}}(D_{2n}^{-}(n)) = D_{2n}^{-}(n)$ (in
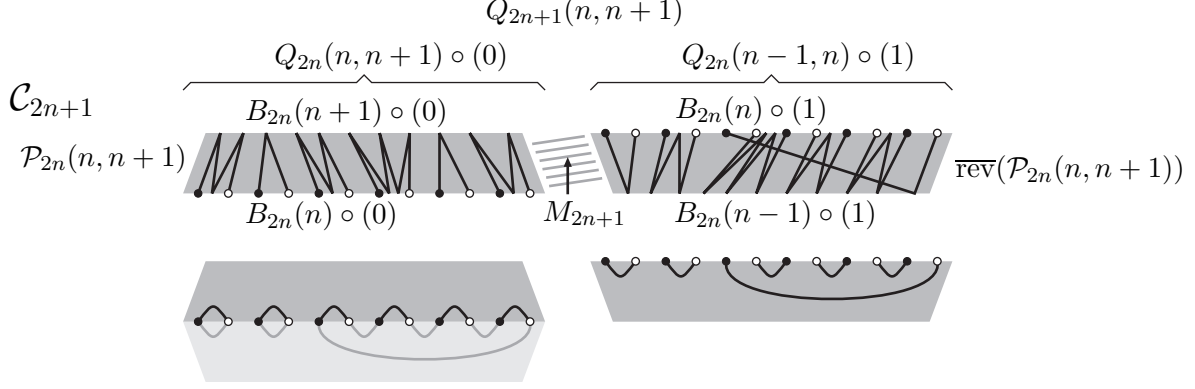
FIGURE 1. The top part of the figure shows the decomposition of the middle levels graph and the definition (4). The 2-factor consists of three disjoint cycles that together visit all vertices of the graph. The first and last vertices of the paths are drawn in black and white, respectively. The bottom part of the figure shows a simplified drawing that helps analyzing the cycle structure of the 2-factor (it has two short cycles and one long cycle).

Figure 1, these sets are the black and white vertices). By property (i) of the paths $\mathcal{P}_{2n}(n, n+1)$, we conclude that

$$\mathcal{C}_{2n+1} := \mathcal{P}_{2n}(n, n+1) \circ 0 \ \cup \ \overline{\text{rev}}(\mathcal{P}_{2n}(n, n+1)) \circ 1 \ \cup \ M'_{2n+1} \tag{4}$$

with $M'_{2n+1} := \{(x \circ 0, x \circ 1) \mid x \in D^{=0}_{2n}(n) \cup D^{-}_{2n}(n)\} \subseteq M_{2n+1}$ is a so-called *2-factor* of the middle levels graph, i.e., a set of disjoint cycles that together visit all vertices of the graph. Note that along each of the cycles in the 2-factor, the paths from $\mathcal{P}_{2n}(n, n+1) \circ 0$ are traversed in forward direction, and the paths from $\overline{\text{rev}}(\mathcal{P}_{2n}(n, n+1)) \circ 1$ in backward direction. The algorithm HAMCYCLENEXT() (called with $\texttt{flip} = \texttt{false}$) computes exactly this 2-factor $\mathcal{C}_{2n+1}$: Given a vertex x of the middle levels graph, it computes the next vertex on one of the cycles from the 2-factor by checking the value of the last bit (line N2), and by returning either the next vertex on the corresponding path from $\mathcal{P}_{2n}(n, n+1) \circ 0$ (line N5) or the previous vertex on the corresponding path from $\overline{\text{rev}}(\mathcal{P}_{2n}(n, n+1)) \circ 1$ (line N8; recall that $\overline{\text{rev}}^{-1} = \overline{\text{rev}}$). The cases that the next cycle edge is an edge from M'_{2n+1} receive special treatment: In these cases the last bit is flipped (lines N4 and N7).

Algorithm 2: HAMCYCLENEXT$(n, x, \texttt{flip})$

Input: An integer $n \geq 1$, a vertex $x \in Q_{2n+1}(n, n+1)$, state variable $\texttt{flip} \in \{\texttt{true}, \texttt{false}\}$
Output: Starting from x the next vertex on a Hamilton cycle in $Q_{2n+1}(n, n+1)$, updated state variable $\texttt{flip} \in \{\texttt{true}, \texttt{false}\}$

N1 Split $x = (x_1, x_2, \ldots, x_{2n+1})$ into $x^- := (x_1, x_2, \ldots, x_{2n})$ and the last bit x_{2n+1}
N2 **if** $x_{2n+1} = 0$ **then**
N3   **if** $x^- \in D^{=0}_{2n}(n)$ **then return** $\left(\text{PATHS}(n, n, x^-, \texttt{next}, a) \circ x_{2n+1}, a \right)$ where $a := \text{ISFLIPVERTEX}(n, x^-)$
N4   **else if** $x^- \in D^{-}_{2n}(n)$ **then return** $(x^- \circ 1, \texttt{false})$
N5   **else return** $\left(\text{PATHS}(n, n, x^-, \texttt{next}, \texttt{flip}) \circ x_{2n+1}, \texttt{flip} \right)$
N6 **else**   /* $x_{2n+1} = 1$ */
N7   **if** $x^- \in D^{=0}_{2n}(n)$ **then return** $(x^- \circ 0, \texttt{false})$
N8   **else return** $\left(\overline{\text{rev}} \left(\text{PATHS}(n, n, \overline{\text{rev}}^{-1}(x^-), \texttt{prev}, \texttt{false}) \right) \circ x_{2n+1}, \texttt{false} \right)$

As mentioned before, calling the algorithm HamCycleNext() with $\texttt{flip} = \texttt{false}$ yields the 2-factor in the middle levels graph defined in (4) that consists of more than one cycle. However, we will show that by replacing in (4) some of the flippable pairs of paths from $\mathcal{P}_{2n}(n, n+1) \circ 0$ by the corresponding flipped paths from the set $\widetilde{\mathcal{P}}_{2n}(n, n+1) \circ 0$, which are computed by calling the algorithm Paths() with $\texttt{flip} = \texttt{true}$, we obtain a 2-factor that has only one cycle, i.e., a Hamilton cycle. The key insight is that replacing a pair of flippable paths that are contained in two different cycles of the 2-factor $\mathcal{C}_{2n+1}$ by the corresponding flipped paths joins these two cycles to one cycle. This is an immediate consequence of the definition of flippable/flipped pairs of paths. This flipping is controlled by the parameter $\texttt{flip}$ of the algorithm HamCycleNext(): It decides whether to compute the next vertex on a path from $\mathcal{P}_{2n}(n, n+1) \circ 0$ (if $\texttt{flip} = \texttt{false}$) or from $\widetilde{\mathcal{P}}_{2n}(n, n+1) \circ 0$ (if $\texttt{flip} = \texttt{true}$). Note that these modifications do not affect the paths $\overline{\text{rev}}(\mathcal{P}_{2n}(n, n+1)) \circ 1$ in the union (4): For the corresponding instructions in lines N6–N8, the value of $\texttt{flip}$ is irrelevant.

The decision whether to follow a path from $\mathcal{P}_{2n}(n, n+1) \circ 0$ or from $\widetilde{\mathcal{P}}_{2n}(n, n+1) \circ 0$ is computed at the vertices $x \circ 0$, $x \in D_{2n}^{=0}(n)$, by calling the function IsFlipVertex(n, x) (line N3). This decision is returned to the caller and maintained until the last vertex of the corresponding path in the graph $Q_{2n}(n, n+1) \circ 0$ is reached. Recall that by the definition of flippable pairs of paths, this decision can be made *independently* for each flippable pair. Of course it has to be consistent for both paths in a flippable pair: either both are flipped or none of them are.

2.4. **The top-level algorithm.** The algorithm HamCycle(n, x, ℓ) takes as input a vertex x of the middle levels graph $Q_{2n+1}(n, n+1)$ and computes the next ℓ vertices that follow x on a Hamilton cycle in this graph. I.e., for $\ell \leq N = 2\binom{2n+1}{n}$, every vertex appears at most once in the output, and for $\ell = N$ every vertex appears exactly once and the vertex x comes last. In terms of Gray codes, the algorithm takes a bitstring x of length $2n+1$ that has weight n or $n+1$ and outputs the ℓ subsequent bitstrings in a middle levels Gray code. A single call HamCycle(n, x, ℓ) yields the same output as ℓ subsequent calls $x_{i+1} := \text{HamCycle}(n, x_i, 1)$, $i = 0, 1, \ldots, \ell - 1$, with $x_0 := x$. However, with respect to running times, the former is faster than the latter.

Algorithm 3: HamCycle(n, x, ℓ)

Input: An integer $n \geq 1$, a vertex $x \in Q_{2n+1}(n, n+1)$, an integer $\ell \geq 1$
Output: Starting from x, the next ℓ vertices on a Hamilton cycle in $Q_{2n+1}(n, n+1)$

H1 Split $x = (x_1, x_2, \ldots, x_{2n+1})$ into $x^- := (x_1, x_2, \ldots, x_{2n})$ and the last bit x_{2n+1}
H2 $\texttt{flip} := \texttt{false}$
H3 **if** $x_{2n+1} = 0$ **then**   /* initialize state variable flip */
H4 $\quad$ $y := x^-$
H5 $\quad$ **while** $y \notin D_{2n}^{=0}(n)$ **do**   /* move backwards to first path vertex */
H6 $\quad\quad$ $y := \text{Paths}(n, n, y, \texttt{prev}, \texttt{false})$
H7 $\quad$ $\texttt{flip} := \text{IsFlipVertex}(n, y)$

H8 $y := x$
H9 **for** $i := 1$ **to** ℓ **do**   /* Hamilton cycle computation */
H10 $\quad$ $(y, \texttt{flip}) := \text{HamCycleNext}(n, y, \texttt{flip})$
H11 $\quad$ **output** y

The algorithm HamCycle() consists of an initialization phase (lines H2–H7) in which the initial value of the state variable $\texttt{flip}$ is computed. This is achieved by following the corresponding path from $\mathcal{P}_{2n}(n, n+1) \circ 0$ backwards to its first vertex (lines H5–H6) and by calling the function IsFlipVertex() (line H7). The actual Hamilton cycle computation (lines H8–H11) repeatedly
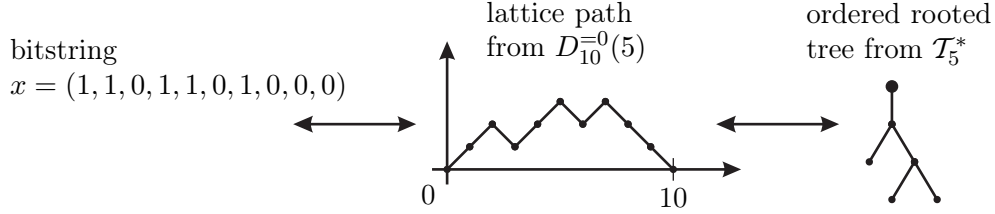
FIGURE 2. Bijections between bitstrings and lattice paths (left hand side), and between lattice paths from $D_{2n}^{=0}(n)$ and ordered rooted trees from $\mathcal{T}_n^*$ (right hand side).

computes the subsequent cycle vertex and updates the state variable `flip` by calling the function HAMCYCLENEXT() discussed in the previous section (line H10).

2.5. **Flip vertex computation.** To complete the description of our Hamilton cycle algorithm, it remains to specify the auxiliary function ISFLIPVERTEX(n, x). As mentioned before, this function decides for each vertex $x \in D_{2n}^{=0}(n)$ whether our Hamilton cycle algorithm should follow the path from $\mathcal{P}_{2n}(n, n+1)$ that starts with this vertex (return value `false`) or the corresponding flipped path from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ (return value `true`) in the graph $Q_{2n}(n, n+1) \circ 0$ (recall (4)). This function therefore nicely encapsulates the core 'intelligence' of our algorithm so that it produces a 2-factor consisting only of a single cycle and not of several smaller cycles. Admittedly, the definition of this function is rather technical, and we postpone the formal correctness proof until Section 3. We begin by introducing several new concepts related to trees.

Ordered rooted trees. An *ordered rooted tree* is a rooted tree where the children of each vertex have a specified left-to-right ordering. We think of an ordered rooted tree as a tree embedded in the plane with the root on top, with downward edges leading from any vertex to its children, and the children appear in the specified left-to-right ordering. This is illustrated on the right hand side of Figure 2, where the root vertex is drawn boldly. We denote by $\mathcal{T}_n^*$ the set of all ordered rooted trees with n edges. It is well known that the number of trees in $\mathcal{T}_n^*$ is given by the n-th Catalan number (see [Sta99]).

Bijection between lattice paths and ordered rooted trees. We identify each lattice path (=bitstring) $x = (x_1, x_2, \ldots, x_{2n})$ from the set $D_{2n}^{=0}(n)$ with an ordered rooted tree from the set $\mathcal{T}_n^*$ as follows (see the right hand side of Figure 2 and [Sta99]): Starting with a tree that has only a root vertex, we read the bits of x from left to right, and for every 1-bit we add a new rightmost child to the current vertex and move to this child, for every 0-bit we add no edge but simply move back to the parent of the current vertex (we clearly end up back at the root). This construction defines a bijection between the lattice paths $D_{2n}^{=0}(n)$ and the ordered rooted trees $\mathcal{T}_n^*$, and in the following we will repeatedly switch between these two representations.

Thin/thick leaves, clockwise/counterclockwise-next leaves. We call a leaf u of a tree *thin* or *thick*, if the vertex adjacent to u has degree exactly 2 or at least 3, respectively. Clearly, for any tree with at least two edges, every leaf is either thin or thick. Given two leaves u and v of an ordered rooted tree T, we say that v is the *clockwise-next* leaf from u, or equivalently, u is the *counterclockwise-next* leaf from v, if all edges of T not on the path p from u to v lie to the right of the path p when traversing p from u to v in the given embedding of T, respectively. Equivalently, in a (cyclic) depth-first search starting at the root, u is the first leaf encountered after the leaf v. This definition is illustrated in Figure 3.

Tree transformations τ_1 and τ_2. We define sets of ordered rooted trees $\mathcal{T}_{n,1}^*, \mathcal{T}_{n,2}^* \subseteq \mathcal{T}_n^*$ and transformations τ_1, τ_2 operating on these trees as follows: The set $\mathcal{T}_{n,1}^*$ contains all ordered rooted trees T with at least three edges for which the leftmost child u' of the root v has exactly one child u and u
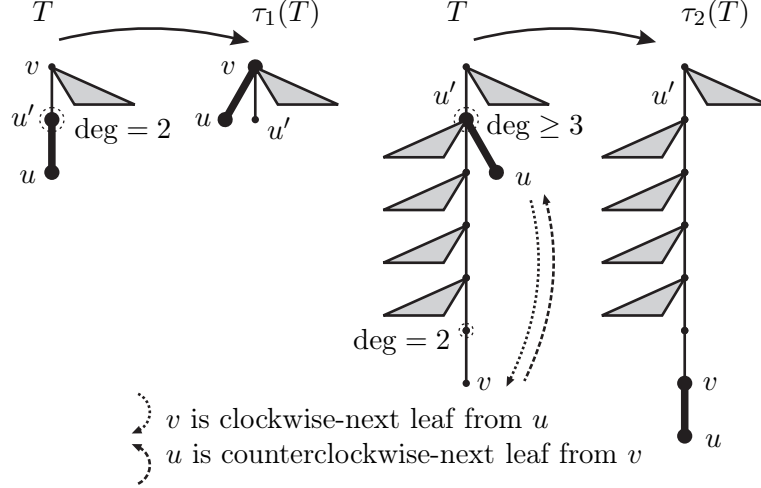
FIGURE 3. Definition of the tree transformations τ_1 (left) and τ_2 (right). The edges in which the trees differ are drawn bold, and the grey areas represent arbitrary subtrees.

is a leaf. In other words, the leftmost subtree of the root is a path on two edges, see the left hand side of Figure 3. For such a tree T, we define $\tau_1(T)$ as the tree obtained by replacing the edge (u, u') by (u, v) so that u becomes the leftmost child of the root. In the bitstring representation, we have $T = 1100 \circ T'$ for some $T' \in \mathcal{T}_{n-2}^*$, and $\tau_1(T) = 1010 \circ T'$. The set $\mathcal{T}_{n,2}^*$ contains all ordered rooted trees T for which the rightmost child u of the leftmost child u' of the root is a thick leaf (so u' has more children to the left of u), and the clockwise-next leaf v of u is thin. Clearly, such a tree must have at least four edges, see the right hand side of Figure 3. For such a tree T, we define $\tau_2(T)$ as the tree obtained by replacing the edge (u, u') by (u, v). In the bitstring representation, we have

$$T = 1 \circ T_1 \circ 1 \circ T_2 \circ 1 \circ \cdots \circ T_{k+1} \circ 1100 \circ 0^k \circ 10 \circ 0 \circ T_0$$

for some $k \geq 0$ and $T_i \in \mathcal{T}_{n_i}^*$ for all $i = 0, 1, \ldots, k+1$, and

$$\tau_2(T) = 1 \circ T_1 \circ 1 \circ T_2 \circ 1 \circ \cdots \circ T_{k+1} \circ 111000 \circ 0^k \circ 0 \circ T_0 \ .$$

Note that the four sets $\mathcal{T}_{n,1}^*$, $\mathcal{T}_{n,2}^*$, $\tau_1(\mathcal{T}_{n,1}^*)$ and $\tau_2(\mathcal{T}_{n,2}^*)$ are all disjoint.

The mapping h. For any lattice path $x \in D_{2n}^{=0}(n)$ we inductively define a lattice path $h(x) \in D_{2n}^{=0}(n)$ as follows: If $x = ()$ then we define

$$h(x) := () \ . \tag{5a}$$

Otherwise there is a unique partition $x = 1 \circ x_\ell \circ 0 \circ x_r$ such that $x_\ell \in D_{2k}^{=0}(k)$ for some $k \geq 0$ and we define

$$h(x) := 1 \circ \pi(h(x_\ell)) \circ 0 \circ h(x_r) \ , \tag{5b}$$

where π is the permutation defined in Section 2.1. It is easy to check that h defines a bijection on $D_{2n}^{=0}(n)$, and that the definition of the inverse mapping h^{-1} can be derived from (5) simply by replacing $\pi(h(x_\ell))$ in (5b) by $h^{-1}(\pi(x_\ell))$ and $h(x_r)$ by $h^{-1}(x_r)$. Clearly, via the bijection between lattice paths and ordered rooted trees discussed before, the mapping h can be interpreted as a bijection on the set $\mathcal{T}_n^*$. As an example, Figure 5 shows the effect of applying h to all 14 trees in the set $\mathcal{T}_4^*$. The preimages under h appear at the top of the figure, the corresponding images below.

With these definitions at hand, consider now the definition of the function $\text{IsFlipVertex}(n, x)$. It first computes the ordered rooted tree T obtained by applying h^{-1} to the bitstring/lattice path x (line T1). On lines T2–T3 the function checks whether T is a preimage or image under τ_1 and calls an auxiliary function $\text{IsFlipTree}_1()$ (defined below) with T or $\tau_1^{-1}(T)$ as an argument,

respectively. A similar check is performed in lines T4–T5 with respect to τ_2, using the auxiliary function IsFlipTree$_2$() (defined below).

Algorithm 4: IsFlipVertex(n, x)

Input: An integer $n \geq 1$, a vertex $x \in D_{2n}^{=0}(n)$
Output: $\{\texttt{true}, \texttt{false}\}$

T1 $T := h^{-1}(x)$ /* compute corresponding rooted tree */
T2   **if** $T \in \mathcal{T}_{n,1}^*$ **then return** IsFlipTree$_1(T)$
T3   **else if** $T \in \tau_1(\mathcal{T}_{n,1}^*)$ **then return** IsFlipTree$_1(\tau_1^{-1}(T))$
T4   **else if** $T \in \mathcal{T}_{n,2}^*$ **then return** IsFlipTree$_2(T)$
T5   **else if** $T \in \tau_2(\mathcal{T}_{n,2}^*)$ **then return** IsFlipTree$_2(\tau_2^{-1}(T))$
T6   **else return** false

We proceed to define the auxiliary functions IsFlipTree$_1$() and IsFlipTree$_2$() that have return values $\{\texttt{true}, \texttt{false}\}$. For brevity we define them in textual form, not as pseudocode. For the definitions we need to introduce the notion of tree rotation.

Rotation rot() *of ordered rooted trees.* For an ordered rooted tree $T \in \mathcal{T}_n^*$ we let rot(T) denote the tree that is obtained from T by shifting the root to the leftmost child of the root. In terms of lattice paths, considering the unique decomposition $T = 1 \circ T_\ell \circ 0 \circ T_r$ with $T_\ell \in D_{2k}^{=0}(k)$ for some $k \geq 0$, we have rot$(T) = T_\ell \circ 1 \circ T_r \circ 0$. Note that we obtain the same tree again after $2n/s$ such rotations, where the value of s depends on the symmetries of the tree. As an example, the top part of Figure 5 shows the effect of applying the rotation operation to all 14 trees in the set $\mathcal{T}_4^*$.

The function IsFlipTree$_1(T)$: Given a tree $T \in \mathcal{T}_{n,1}^*$, compute all rotated versions of it that are also in the set $\mathcal{T}_{n,1}^*$. Return $\texttt{true}$ if T is lexicographically smallest among them, and $\texttt{false}$ otherwise.

The function IsFlipTree$_2(T)$: Given a tree $T \in \mathcal{T}_{n,2}^*$, if $T = 1^{n-1} \circ 0^{n-2} \circ 100$ or if T has more than one thin leaf, return $\texttt{false}$. Otherwise, let v be the unique thin leaf of T, v' the parent of v, v'' the parent of v', and w the clockwise-next leaf of v, and let T' be the tree obtained from T by replacing the edge (v, v') by (v, v'') so that v becomes a child of v'' to the left of v' (so T' has only thick leaves), and by rotating it such that the leaf w becomes the root. Let d be the distance between the root w and the leftmost leaf v of T'. Compute all other rotated versions of T' for which the root is a leaf and the parent of the leftmost leaf x has another leaf as its child to the right of x (initially, v and v' are these two leaves). For each of them compute the distance d' between the root and the leftmost leaf. Return $\texttt{true}$ if $d' \leq d$ for all of them and if T' is lexicographically smallest among all rotated trees with $d' = d$, and $\texttt{false}$ otherwise.

When referring to lexicographic comparisons between ordered rooted trees in the previous definitions we mean lexicographic comparisons between the corresponding bitstrings.

3. Correctness of the algorithm

In this section we prove the correctness of the algorithm presented in the previous section. Before doing this, we recapitulate the inductive construction of paths $\mathcal{P}_n(k, k+1)$ and $\widetilde{\mathcal{P}}_n(k, k+1)$ described in [Müt16] in Sections 3.1 and 3.2 below. These sections are essentially duplicate material from [Müt16], and readers familiar with this construction may want to skip them. The main goal of Section 3.3 is to prove that those paths are the same as the ones produced by our recursive algorithm Paths(), so this will justify denoting both with the same symbol $\mathcal{P}_n(k, k + 1)$ or $\widetilde{\mathcal{P}}_n(k, k + 1)$, respectively. Once this is established, we apply several lemmas from [Müt16] to prove the correctness of the algorithm HamCycle() in Section 3.4. When looking up those lemmas the reader should be

aware of some minor differences in notation between [Müt16] and our paper: We use bitstrings to also denote the corresponding lattice paths and ordered rooted trees, implicitly using the corresponding bijections. In the paper [Müt16], those bijections are made explicit and denoted φ and ψ, and upsteps and downsteps of a lattice path are denoted by $\nearrow$ and $\searrow$ rather than 1 and 0.

3.1. **Recap: Inductive construction of paths from** [Müt16].

The construction is parametrized by some sequence $(\alpha_{2i})_{i \geq 1}$, $\alpha_{2i} \in \{0, 1\}^{i-1}$. Given this sequence, we inductively construct a set $\mathcal{P}_{2n}(k, k+1)$ of disjoint oriented paths in $Q_{2n}(k, k+1)$ that start and end in $B_{2n}(k)$ for all $n \geq 1$ and all $k = n, n+1, \ldots, 2n-1$ such that the following conditions hold:

(a) The paths in $\mathcal{P}_{2n}(n, n+1)$ visit all vertices in the sets $B_{2n}(n+1)$ and $B_{2n}(n)$.
(b) For $k = n+1, \ldots, 2n-1$, the paths in $\mathcal{P}_{2n}(k, k+1)$ visit all vertices in the set $B_{2n}(k+1)$, and the only vertices not visited in the set $B_{2n}(k)$ are exactly the elements in the set $S(\mathcal{P}_{2n}(k-1, k))$.

Induction basis $n = 1$ (Q_2): The induction basis is the set of paths $\mathcal{P}_2(1, 2) := \{(10, 11, 01)\}$, consisting only of a single oriented path on three vertices. It is easily checked that this set of paths satisfies properties (a) and (b) (property (b) is satisfied trivially).

Induction step $n \to n+1$ ($Q_{2n} \to Q_{2n+2}$), $n \geq 1$: The inductive construction consists of two intermediate steps.

First intermediate step: Construction of a 2-factor in the middle levels graph $Q_{2n+1}(n, n+1)$. Using only the paths in the set $\mathcal{P}_{2n}(n, n+1)$ and the parameter $\alpha_{2n} = (\alpha_{2n}(1), \ldots, \alpha_{2n}(n-1)) \in \{0, 1\}^{n-1}$ we first construct a 2-factor in the middle levels graph $Q_{2n+1}(n, n+1)$.

The parameter α_{2i} is used to generalize the permutation π introduced before: Specifically, we let $\pi_{\alpha_{2n}}$ denote the permutation on the set $\{0, 1\}^{2n}$ that swaps any two adjacent bits at positions $2i$ and $2i + 1$ for all $i = 1, 2, \ldots, n-1$, if and only if $\alpha_{2n}(i) = 1$, and that leaves the bits at position 1 and $2n$ unchanged. Note that if $\alpha_{2n} = (1, 1, \ldots, 1)$, then $\pi_{\alpha_{2n}}$ equals the mapping π from before. On the other hand, if $\alpha_{2n} = (0, 0, \ldots, 0)$, then no bits are swapped and $\pi_{\alpha_{2n}} = \mathrm{id}$ is simply the identity mapping. For any bitstring $x \in \{0, 1\}^{2n}$ we then define

$$f_{\alpha_{2n}}(x) := \overline{\mathrm{rev}}(\pi_{\alpha_{2n}}(x)) \ . \tag{6}$$

Note that we have $f_{\alpha_{2n}} = \overline{\mathrm{rev}} \bullet \pi$ for $\alpha_{2n} = (1, 1, \ldots, 1)$ and $f_{\alpha_{2n}} = \overline{\mathrm{rev}}$ for $\alpha_{2n} = (0, 0, \ldots, 0)$.

It was proved in [Müt16, Lemma 3] that for any $\alpha_{2n} \in \{0, 1\}^{n-1}$ the mapping $f_{\alpha_{2n}}$ maps each of the sets $F(\mathcal{P}_{2n}(n, n+1))$ and $L(\mathcal{P}_{2n}(n, n+1))$ onto itself. As explained in Section 2.3, the middle layer graph $Q_{2n+1}(n, n+1)$ can be decomposed into the graphs $Q_{2n}(n, n+1) \circ 0$ and $Q_{2n}(n-1, n) \circ 1$ plus the edges from the matching $M_{2n+1} = \{(x \circ 0, x \circ 1) \mid x \in B_{2n}(n)\}$ along which the last bit is flipped. Denoting by M_{2n+1}^{FL} the edges from M_{2n+1} that have one end vertex in the set $\big(F(\mathcal{P}_{2n}(n, n+1)) \cup L(\mathcal{P}_{2n}(n, n+1))\big) \circ 0 \subseteq B_{2n}(n) \circ 0$ (and the other in the set $\big(F(\mathcal{P}_{2n}(n, n+1)) \cup L(\mathcal{P}_{2n}(n, n+1))\big) \circ 1 \subseteq B_{2n}(n) \circ 1$), the union

$$\mathcal{C}_{2n+1} := \mathcal{P}_{2n}(n, n+1) \circ 0 \cup f_{\alpha_{2n}}(\mathcal{P}_{2n}(n, n+1)) \circ 1 \cup M_{2n+1}^{FL} \tag{7}$$

therefore yields a 2-factor in the middle levels graph, with the property that on every cycle of $\mathcal{C}_{2n+1}$, every edge of the form $(F(P), S(P)) \circ 0$ for some $P \in \mathcal{P}_{2n}(n, n+1)$ is oriented the same way. Note that (4) is the special case of (7) with $\alpha_{2n} = (0, 0, \ldots, 0)$. In the correctness proof for our algorithm we will later also consider the case $\alpha_{2n} = (1, 1, \ldots, 1)$. Even though we are eventually only interested in the 2-factor $\mathcal{C}_{2n+1}$ defined in (7), we need to specify how to proceed with the inductive construction of the sets of paths $\mathcal{P}_{2n+2}(k, k+1)$.

Second intermediate step: Splitting up the 2-factor into paths. We proceed by describing how the sets of paths $\mathcal{P}_{2n+2}(k, k+1)$ for all $k = n+1, n+2, \ldots, 2n+1$ satisfying properties (a) and (b) are defined, using the previously constructed sets $\mathcal{P}_{2n}(k, k+1)$ and the 2-factor $\mathcal{C}_{2n+1}$ defined in the first intermediate step.

Consider the decomposition of Q_{2n+2} into $Q_{2n} \circ 00$, $Q_{2n} \circ 10$, $Q_{2n} \circ 01$ and $Q_{2n} \circ 11$ plus the two perfect matchings M_{2n+2} and M'_{2n+2} along which the last and second to last bit are flipped, respectively. For all $k = n+2, \ldots, 2n+1$ we define

$$\mathcal{P}_{2n+2}(k, k+1) := \mathcal{P}_{2n}(k, k+1) \circ 00 \cup \mathcal{P}_{2n}(k-1, k) \circ 10$$
$$\cup \, \mathcal{P}_{2n}(k-1, k) \circ 01 \cup \mathcal{P}_{2n}(k-2, k-1) \circ 11 \, , \tag{8}$$

where we use the convention $\mathcal{P}_{2n}(2n, 2n+1) := \emptyset$ and $\mathcal{P}_{2n}(2n+1, 2n+2) := \emptyset$ to unify treatment of the sets of paths $\mathcal{P}_{2n+2}(2n, 2n+1)$ and $\mathcal{P}_{2n+2}(2n+1, 2n+2)$ between the uppermost levels of Q_{2n+2}. Note that so far none of the edges from the matchings M_{2n+2} or M'_{2n+2} is used.

The definition of the set $\mathcal{P}_{2n+2}(n+1, n+2)$ is slightly more involved. Note that the graph $Q_{2n+2}(n+1, n+2)$ can be decomposed into $Q_{2n+1}(n+1, n+2) \circ 0$ and $Q_{2n+1}(n, n+1) \circ 1$ plus the edges from M_{2n+2} that connect the vertices in the set $B_{2n+1}(n+1) \circ 0$ to the vertices in the set $B_{2n+1}(n+1) \circ 1$. The first graph can be further decomposed into $Q_{2n}(n+1, n+2) \circ 00$ and $Q_{2n}(n, n+1) \circ 10$ plus some matching edges that are not relevant here. The second graph is the middle levels graph of $Q_{2n+1} \circ 1$. Let $\mathcal{C}_{2n+1}^{-}$ denote the graph obtained from the 2-factor $\mathcal{C}_{2n+1}$ defined in (7) by removing every edge of the form $(F(P), S(P)) \circ 0$ for some $P \in \mathcal{P}_{2n}(n, n+1)$. As on every cycle of $\mathcal{C}_{2n+1}$ every such edge is oriented the same way, $\mathcal{C}_{2n+1}^{-}$ is a set of paths (visiting all vertices of the middle levels graph $Q_{2n+1}(n, n+1)$), with the property that each of those paths starts at a vertex of the form $S(P) \circ 0$ and ends at a vertex of the form $F(P') \circ 0$ for two paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$. Denoting by M_{2n+2}^{S} the edges from M_{2n+2} that have one end vertex in the set $S(\mathcal{P}_{2n}(n, n+1)) \circ 00 \subseteq B_{2n+1}(n+1) \circ 00$ (and the other in the set $S(\mathcal{P}_{2n}(n, n+1)) \circ 01 \subseteq B_{2n+1}(n+1) \circ 01$), it follows that

$$\mathcal{P}_{2n+2}^{*} := M_{2n+2}^{S} \cup \mathcal{C}_{2n+1}^{-} \circ 1 \tag{9}$$

is a set of oriented paths that start and end in $B_{2n+2}(n+1)$, where we choose the orientation of each path such that the edge from the set M_{2n+2}^{S} is the first edge. Note that we have

$$F(\mathcal{P}_{2n+2}^{*}) = S(\mathcal{P}_{2n}(n, n+1)) \circ 00 \, . \tag{10}$$

We then define

$$\mathcal{P}_{2n+2}(n+1, n+2) := \mathcal{P}_{2n}(n+1, n+2) \circ 00 \cup \mathcal{P}_{2n}(n, n+1) \circ 10 \cup \mathcal{P}_{2n+2}^{*} \, , \tag{11}$$

where in the case $n = 1$ we use the convention $\mathcal{P}_2(2, 3) := \emptyset$.

It was argued in [Müt16, Section 2.2] that the sets of paths $\mathcal{P}_{2n+2}(k, k+1)$, $k = n+1, n+2, \ldots, 2n+1$, defined in (8) and (11) have properties (a) and (b). We omit those arguments here.

3.2. **Recap: Inductive construction of flippable pairs from** [Müt16]. Let $\mathcal{P}_{2n}(k, k+1)$ be the sets of paths defined in Section 3.1 for an arbitrary parameter sequence $(\alpha_{2i})_{i \geq 1}$, $\alpha_{2i} \in \{0, 1\}^{i-1}$. In the following we show how to inductively construct a set of flippable pairs $\mathcal{X}_{2n}(k, k+1)$ for the set $\mathcal{P}_{2n}(k, k+1)$ for all $n \geq 2$ and all $k = n, n+1, \ldots, 2n-1$, and the corresponding flipped paths $\widetilde{\mathcal{P}}_{2n}(k, k+1)$. So the set $\mathcal{X}_{2n}(k, k+1)$ contains pairs of paths (P, P'), $P, P' \in \mathcal{P}_{2n}(k, k+1)$, that form a flippable pair, where every path appears in at most one such pair, and the set $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ is the union of all paths R, R' that form a corresponding flipped pair (R, R') for (P, P'). This construction arises very naturally from the inductive construction of the sets $\mathcal{P}_{2n}(k, k+1)$ given in the previous section.

Induction basis $n = 2$ (Q_4): It is easy to check that the set of paths $\mathcal{P}_4(2, 3)$ arising from the inductive construction described in the previous section after one step satisfies $\mathcal{P}_4(2, 3) = \{P, P'\}$ with P and P' as defined in (2) (as $\alpha_2 = ()$ this step does not involve any parameter choices yet). The set $\mathcal{X}_4(2, 3) := \{(P, P')\}$ is a set of flippable pairs for $\mathcal{P}_4(2, 3)$, which can be seen by considering the flipped paths (R, R') with R and R' as defined in (3). For completeness we also define $\mathcal{X}_4(3, 4) := \emptyset$, which is trivially a set of flippable pairs for the set of paths $\mathcal{P}_4(3, 4)$ constructed before. The corresponding flipped paths are therefore $\widetilde{\mathcal{P}}_4(2, 3) := \{R, R'\}$ and $\widetilde{\mathcal{P}}_4(3, 4) := \emptyset$.

Induction step $n \to n+1$ $(Q_{2n} \to Q_{2n+2})$, $n \geq 2$: Consider the sets of flippable pairs $\mathcal{X}_{2n}(k, k+1)$, $k = n, n+1, \ldots, 2n-1$, for the sets of paths $\mathcal{P}_{2n}(k, k+1)$, and the corresponding flipped paths $\widetilde{\mathcal{P}}_{2n}(k, k+1)$. In the following we describe how to use them to construct sets of flippable pairs $\mathcal{X}_{2n+2}(k, k+1)$, $k = n+1, n+2, \ldots, 2n+1$, for the sets $\mathcal{P}_{2n+2}(k, k+1)$ in $Q_{2n+2}(k, k+1)$, and corresponding flipped paths $\widetilde{\mathcal{P}}_{2n}(k, k+1)$.

For all $k = n+2, \ldots, 2n+1$ we define, in analogy to (8),

$$\mathcal{X}_{2n+2}(k, k+1) := \mathcal{X}_{2n}(k, k+1) \circ 00 \cup \mathcal{X}_{2n}(k-1, k) \circ 10$$
$$\cup\, \mathcal{X}_{2n}(k-1, k) \circ 01 \cup \mathcal{X}_{2n}(k-2, k-1) \circ 11 ~, \tag{12}$$

where we use the convention $\mathcal{X}_{2n}(2n, 2n+1) := \emptyset$ and $\mathcal{X}_{2n}(2n+1, 2n+2) := \emptyset$ to unify treatment of the sets of flippable pairs $\mathcal{X}_{2n+2}(2n, 2n+1)$ and $\mathcal{X}_{2n+2}(2n+1, 2n+2)$ between the uppermost levels of Q_{2n+2}. The sets of flippable pairs on the right hand side of (12) clearly lie in four disjoint subgraphs of $Q_{2n+2}(k, k+1)$, so by induction $\mathcal{X}_{2n+2}(k, k+1)$ is indeed a set of flippable pairs for $\mathcal{P}_{2n+2}(k, k+1)$.

The corresponding flipped paths are defined as

$$\widetilde{\mathcal{P}}_{2n+2}(k, k+1) := \widetilde{\mathcal{P}}_{2n}(k, k+1) \circ 00 \cup \widetilde{\mathcal{P}}_{2n}(k-1, k) \circ 10$$
$$\cup\, \widetilde{\mathcal{P}}_{2n}(k-1, k) \circ 01 \cup \widetilde{\mathcal{P}}_{2n}(k-2, k-1) \circ 11 ~,$$

where we use the convention $\widetilde{\mathcal{P}}_{2n}(2n, 2n+1) := \emptyset$ and $\widetilde{\mathcal{P}}_{2n}(2n+1, 2n+2) := \emptyset$.

To define the set $\mathcal{X}_{2n+2}(n+1, n+2)$, we consider the oriented paths $\mathcal{P}^*_{2n+2}$ defined in (9) (recall that these paths originate from splitting up the 2-factor $\mathcal{C}_{2n+1}$ defined in (7)). By (7) and (9), every path $P^+ \in \mathcal{P}^*_{2n+2}$ has the following structure: There are two paths $P, \widehat{P} \in \mathcal{P}_{2n}(n, n+1)$ with $f_{\alpha_{2n}}(L(\widehat{P})) = L(P)$ such that P^+ contains all edges except the first one from $P \circ 01$ and all edges from $f_{\alpha_{2n}}(\widehat{P}) \circ 11$ (P^+ has three more edges, two from the matching $M^{FL}_{2n+1} \circ 1$ and one from the matching M^S_{2n+2}). By this structural property of paths from $\mathcal{P}^*_{2n+2}$ and the fact that $f_{\alpha_{2n}}$ is an isomorphism between the graphs $Q_{2n}(n, n+1)$ and $Q_{2n}(n-1, n)$, the set

$$\mathcal{X}^*_{2n+2} := \big\{ (P^+, P^{+'}) \mid P^+, P^{+'} \in \mathcal{P}^*_{2n+2} \text{ and there is a flippable pair } (\widehat{P}, \widehat{P}') \in \mathcal{X}_{2n}(n, n+1)$$
$$\text{with } f_{\alpha_{2n}}(\widehat{P}) \circ 11 \subseteq P^+ \text{ and } f_{\alpha_{2n}}(\widehat{P}') \circ 11 \subseteq P^{+'} \big\} ~. \tag{13}$$

is a set of flippable pairs for $\mathcal{P}^*_{2n+2}$. We now define, in analogy to (11),

$$\mathcal{X}_{2n+2}(n+1, n+2) := \mathcal{X}_{2n}(n+1, n+2) \circ 00 \cup \mathcal{X}_{2n}(n, n+1) \circ 10 \cup \mathcal{X}^*_{2n+2} ~. \tag{14}$$

The sets of flippable pairs on the right hand side of (14) lie in three disjoint subgraphs of $Q_{2n+2}(n+1, n+2)$, so by induction $\mathcal{X}_{2n+2}(n+1, n+2)$ is indeed a set of flippable pairs for $\mathcal{P}_{2n+2}(n+1, n+2)$.

The corresponding flipped paths are defined as

$$\widetilde{\mathcal{P}}_{2n+2}(n+1, n+2) := \widetilde{\mathcal{P}}_{2n}(n+1, n+2) \circ 00 \cup \widetilde{\mathcal{P}}_{2n}(n, n+1) \circ 10 \cup \widetilde{\mathcal{P}}^*_{2n+2} ~, \tag{15}$$

where $\widetilde{\mathcal{P}}^*_{2n+2}$ is the union of all paths obtained by considering every flippable pair $(P^+, P^{+'})$ in the set $\mathcal{X}^*_{2n+2}$ defined in (13) and by modifying P^+ and $P^{+'}$ as follows: Let $(\widehat{P}, \widehat{P}') \in \mathcal{X}_{2n}(n, n+1)$ be such that $f_{\alpha_{2n}}(\widehat{P}) \circ 11 \subseteq P^+$ and $f_{\alpha_{2n}}(\widehat{P}') \circ 11 \subseteq P^{+'}$ (see the right hand side of (13)) and let $\overline{P}$ and $\overline{P}'$ be the corresponding flipped paths from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ so that $F(\widehat{P}) = F(\overline{P})$, $F(\widehat{P}') = F(\overline{P}')$, $L(\widehat{P}) = L(\overline{P}')$, $L(\widehat{P}') = L(\overline{P})$. We replace the subpath $f_{\alpha_{2n}}(\widehat{P}) \circ 11$ of P^+ by the subpath $f_{\alpha_{2n}}(\overline{P}') \circ 11$ and the last edge $(F(\widehat{P}) \circ 11, F(\widehat{P}) \circ 01)$ of P^+ by the edge $(F(\overline{P}') \circ 11, F(\overline{P}') \circ 01)$. Similarly, we replace the subpath $f_{\alpha_{2n}}(\widehat{P}') \circ 11$ of $P^{+'}$ by the subpath $f_{\alpha_{2n}}(\overline{P}) \circ 11$ and the last edge $(F(\widehat{P}') \circ 11, F(\widehat{P}') \circ 01)$ of $P^{+'}$ by the edge $(F(\overline{P}) \circ 11, F(\overline{P}) \circ 01)$. Clearly, the resulting paths are a flipped pair of paths corresponding to the flippable pair of paths $(P^+, P^{+'})$ that was added to $\mathcal{X}^*_{2n+2}$.

3.3. **Correctness of the algorithm PATHS().** In this section we show that the paths $\mathcal{P}_{2n}(k, k+1)$ and $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ in the graph $Q_{2n}(k, k+1)$ computed by the algorithm PATHS() that form the basic building blocks of our Hamilton cycle algorithm (recall (4)) are exactly the same as the ones defined in Sections 3.1 and 3.2. We shall see that essentially, the algorithm PATHS() is a recursive reformulation of this inductive construction. This allows us to exploit all the properties proved in [Müt16] about these paths. In particular, we establish that the paths $\mathcal{P}_{2n}(n, n+1)$ satisfy properties (i) and (ii) mentioned in Section 2.2. Recall that all paths we consider are oriented from the first vertex to the last, and the notions of previous and next vertices on a path are defined with respect to this orientation.

Lemma 4. *Let $n \geq 1$ and $k \in \{n, n+1, \ldots, 2n-1\}$ be fixed, and let $\mathcal{P}_{2n}(k, k+1)$ be the set of paths in $Q_{2n}(k, k+1)$ defined in Section 3.1 for the parameter sequence $\alpha_{2i} = (1, 1, \ldots, 1) \in \{0, 1\}^{i-1}$, $i = 1, 2, \ldots, n-1$. For any path $P \in \mathcal{P}_{2n}(k, k+1)$ and any two consecutive vertices u and v on P (i.e., P has the form $P = (\ldots, u, v, \ldots)$) we have $v = \text{PATHS}(n, k, u, \text{next}, \text{false})$ and $u = \text{PATHS}(n, k, v, \text{prev}, \text{false})$.*

In the following we will repeatedly use that by [Müt16, Lemma 11] the sets of first, second and last vertices of the paths in $\mathcal{P}_{2n}(k, k+1)$ satisfy the relations

$$F(\mathcal{P}_{2n}(k, k+1)) = D_{2n}^{=0}(k) \ , \tag{16a}$$

$$S(\mathcal{P}_{2n}(k, k+1)) = D_{2n}^{>0}(k+1) \ , \tag{16b}$$

$$L(\mathcal{P}_{2n}(k, k+1)) = D_{2n}^{-}(k) \ . \tag{16c}$$

Proof. Note that when calling the algorithm PATHS() with the parameter $\text{flip} = \text{false}$, this parameter retains its value in all recursive calls, so we can ignore lines P3–P4 of the algorithm for this proof. We prove the lemma by induction on n. To settle the base case $n = 1$ observe that the set of paths $\mathcal{P}_2(1, 2)$ defined in Section 3.1 and containing only a single path is the same as the one defined in (1) and returned in lines P1–P2.

For the induction step $n \to n+1$ we assume that the lemma holds for $n \geq 1$ and all $k = n, n+1, \ldots, 2n-1$, and prove it for $n+1$ and all $k = n+1, n+2, \ldots, 2n+1$. We distinguish the cases $k = n+1$ and $k \in \{n+2, \ldots, 2n+1\}$.

For the case $k \in \{n+2, \ldots, 2n+1\}$ let $\mathcal{P}_{2n+2}(k, k+1)$ be the set of paths defined in (8), let P be a path from this set and let u and v be two consecutive vertices on P. Note that for both calls $\text{PATHS}(n+1, k, u, \text{next}, \text{false})$ and $\text{PATHS}(n+1, k, v, \text{prev}, \text{false})$ the condition in line P6 is satisfied, entailing the recursive call in line P7. The claim therefore follows from the definition (8) and by induction: The quantity $k - u_{2n} - u_{2n+1}$ computed in line P7 evaluates to k, $k-1$ or $k-2$, depending on the values of the last two bits of u and an analogous statement holds for v.

For the case $k = n+1$ let $\mathcal{P}_{2n+2}(n+1, n+2)$ be the set of paths defined in (11), let P be a path from this set and let u and v be two consecutive vertices on P. We distinguish three cases depending on which of the three sets on the right hand side of (11) the path P is contained in.

Case 1: If P is contained in the second set on the right hand side of (11), i.e., in the set $\mathcal{P}_{2n}(n, n+1) \circ 10$, then the last two bits of u and v are equal to 10, so for both calls $\text{PATHS}(n+1, k, u, \text{next}, \text{false})$ and $\text{PATHS}(n+1, k, v, \text{prev}, \text{false})$ the condition in line P9 is satisfied, entailing the recursive call in line P10. In this case the claim follows by induction.

Case 2: If P is contained in the first set on the right hand side of (11), i.e., in the set $\mathcal{P}_{2n}(n+1, n+2) \circ 00$, then the last two bits of u and v are equal to 00, so for both calls $\text{PATHS}(n+1, k, u, \text{next}, \text{false})$ and $\text{PATHS}(n+1, k, v, \text{prev}, \text{false})$ the condition in line P11 is satisfied. However, neither u nor v satisfies the condition in line P12, as by (10) and (16b), the vertices in $D_{2n}^{>0}(n+1) \circ 00$ are all visited by the paths $\mathcal{P}_{2n+2}^*$ that form the third set on the right hand side of set of (11), so P does

not contain any of these vertices. Consequently, both cases entail the recursive call in line P13 and the claim follows by induction.

Case 3: If P is contained in the third set on the right hand side of (11), i.e., in the set $\mathcal{P}_{2n+2}^*$, then by the definitions (7) and (9) and the fact that for $\alpha_{2n} = (1, 1, \ldots, 1) \in \{0, 1\}^{n-1}$ the mapping defined in (6) satisfies $f_{\alpha_{2n}} = \overline{\mathrm{rev}} \bullet \pi$, the sequence of edges of the path P when traversing it from its first to its last vertex has the form $(e_1, E_2, e_3, E_4, e_5)$, where e_1, e_3 and e_5 are single edges and E_2 and E_4 are sequences of edges that satisfy the following conditions: There are two paths $P', P'' \in \mathcal{P}_{2n}(n, n+1)$ such that

$$
\begin{aligned}
e_1 &= \big(S(P') \circ 00, S(P') \circ 01\big) \ , \\
e_3 &= \big(L(P') \circ 01, L(P') \circ 11\big) = \big(L(P') \circ 01, \overline{\mathrm{rev}}(\pi(L(P''))) \circ 11\big) \ , \\
e_5 &= \big(\overline{\mathrm{rev}}(\pi(F(P''))) \circ 11, \overline{\mathrm{rev}}(\pi(F(P''))) \circ 01\big) \ ,
\end{aligned}
$$

E_2 is given by traversing the edges of $P' \circ 01$ starting at the vertex $S(P') \circ 01$ and ending at the vertex $L(P') \circ 01$, and E_4 is given by the traversing all the edges of $\overline{\mathrm{rev}}(\pi(P'')) \circ 11$ in reverse order starting at the vertex $\overline{\mathrm{rev}}(\pi(L(P''))) \circ 11$ and ending at the vertex $\overline{\mathrm{rev}}(\pi(F(P''))) \circ 11$. We distinguish five subcases depending on which part of P the edge (u, v) belongs to.

Case 3.1: If $(u, v) = e_1 = \big(S(P') \circ 00, S(P') \circ 01\big)$, then the vertices u and v differ in the last bit. It follows from (16b) that for the call $\mathrm{PATHS}(n+1, k, u, \mathtt{next}, \mathtt{false})$ the conditions in line P11 and P12 are both satisfied, so the algorithm correctly returns v which is obtained from u by flipping the last bit. Similarly, for the call $\mathrm{PATHS}(n+1, k, v, \mathtt{prev}, \mathtt{false})$ the conditions in line P14 and P17 are both satisfied, so the algorithm correctly returns u which is obtained from v by flipping the last bit.

Case 3.2: If (u, v) is an edge from E_2, then the last two bits of u and v are equal to 01, so for both calls $\mathrm{PATHS}(n+1, k, u, \mathtt{next}, \mathtt{false})$ and $\mathrm{PATHS}(n+1, k, v, \mathtt{prev}, \mathtt{false})$ the condition in line P14 is satisfied. However, the conditions in line P15, P16 and P17 are not satisfied: To see this recall (16a), (16b) and (16c), and that by the definition of E_2, u and v are different from $F(P') \circ 01$, u is different from $L(P') \circ 01$, and v is different from $S(P') \circ 01$. Consequently, both cases entail the recursive call in line P18 and the claim follows by induction.

Case 3.3: If $(u, v) = e_3 = \big(L(P') \circ 01, L(P') \circ 11\big)$, then the vertices u and v differ in the second to last bit. It follows from (16c) that for the call $\mathrm{PATHS}(n+1, k, u, \mathtt{next}, \mathtt{false})$ the conditions in line P14 and P16 are both satisfied, so the algorithm correctly returns v which is obtained from u by flipping the second to last bit. Similarly, for the call $\mathrm{PATHS}(n+1, k, v, \mathtt{prev}, \mathtt{false})$ the conditions in line P19 and P21 are both satisfied, so the algorithm correctly returns u which is obtained from v by flipping the second to last bit.

Case 3.4: If (u, v) is an edge from E_4, then the last two bits of u and v are equal to 11, so for both calls $\mathrm{PATHS}(n+1, k, u, \mathtt{next}, \mathtt{false})$ and $\mathrm{PATHS}(n+1, k, v, \mathtt{prev}, \mathtt{false})$ the condition in line P19 is satisfied. However, the conditions in line P20 and P21 are not satisfied: To see this recall (16a) and (16c) and that the mapping $\overline{\mathrm{rev}} \bullet \pi$ maps each of the sets $F(\mathcal{P}_{2n}(n, n+1)) = D_{2n}^{=0}(n)$ and $L(\mathcal{P}_{2n}(n, n+1)) = D_{2n}^-(n)$ onto itself, and note that by the definition of E_4, u is different from $\overline{\mathrm{rev}}(\pi(F(P''))) \circ 11$ and v is different from $\overline{\mathrm{rev}}(\pi(L(P''))) \circ 11$. Consequently, both cases entail the recursive call in line P22. The claim follows by induction, noting that inverting the value of the variable $\mathtt{dir}$ accounts for the fact that in E_4 the edges of $\overline{\mathrm{rev}}(\pi(P'')) \circ 11$ are traversed in reverse order.

Case 3.5: If $(u, v) = e_5 = \big(\overline{\mathrm{rev}}(\pi(F(P''))) \circ 11, \overline{\mathrm{rev}}(\pi(F(P''))) \circ 01\big)$, then the vertices u and v differ in the second to last bit. It follows from (16a) that $\overline{\mathrm{rev}}(\pi(F(P''))) \in D_{2n}^{=0}(n)$, so for the call $\mathrm{PATHS}(n+1, k, u, \mathtt{next}, \mathtt{false})$ the conditions in line P19 and P20 are both satisfied and the algorithm correctly returns v which is obtained from u by flipping the second to last bit. Similarly,

for the call $\textsc{Paths}(n+1, k, v, \textbf{prev}, \texttt{false})$ the conditions in line P14 and P15 are both satisfied, so the algorithm correctly returns u which is obtained from v by flipping the second to last bit.

This completes the proof. $\qquad\square$

Lemma 5. *Let $n \geq 2$ and $k \in \{n, n+1, \ldots, 2n-1\}$ be fixed, let $\mathcal{P}_{2n}(k, k+1)$ be the set of paths defined in Section 3.1 and $\mathcal{X}_{2n}(k, k+1)$ the set of flippable pairs for $\mathcal{P}_{2n}(k, k+1)$ and $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ the corresponding set of flipped paths in $Q_{2n}(k, k+1)$ defined in Section 3.2 for the parameter sequence $\alpha_{2i} = (1, 1, \ldots, 1) \in \{0, 1\}^{i-1}$, $i = 1, 2, \ldots, n-1$. For any path $P \in \widetilde{\mathcal{P}}_{2n}(k, k+1)$ and any two consecutive vertices u and v on P (i.e., P has the form $P = (\ldots, u, v, \ldots)$) we have $v = \textsc{Paths}(n, k, u, \textbf{next}, \textbf{true})$ and $u = \textsc{Paths}(n, k, v, \textbf{prev}, \textbf{true})$.*

From (16) and from the definition of flippable/flipped pairs of paths we obtain that

$$F(\widetilde{\mathcal{P}}_{2n}(k, k+1)) = D_{2n}^{=0}(k) \ , \tag{17a}$$

$$L(\widetilde{\mathcal{P}}_{2n}(k, k+1)) = D_{2n}^{-}(k) \ . \tag{17b}$$

Proof. We prove the lemma by induction on n, in an analogous way as the proof of Lemma 4. One crucial difference, however, is the following: When calling the algorithm $\textsc{Paths}()$ with the parameter $\texttt{flip} = \texttt{true}$, this parameter retains its value in all recursive calls except in line P18, where it is set to $\texttt{false}$.

To settle the base case $n = 2$ observe that the set of paths $\widetilde{\mathcal{P}}_4(2, 3)$ defined in Section 3.2 (consisting of two paths on three and seven vertices, respectively) is the same as the one defined in (3) and returned in lines P3–P4. The set of paths $\widetilde{\mathcal{P}}_4(3, 4)$ defined in Section 3.2 is empty, so the claim is trivially true.

The induction step $n \to n+1$ proceeds in an analogous way as in the proof of Lemma 4. We only give the details for case 3, which is proved in a slightly different way:

Case 3': If P is contained in the third set on the right hand side of (15), i.e., in the set $\widetilde{\mathcal{P}}_{2n+2}^*$, then by the definition given after (15) and the fact that for $\alpha_{2n} = (1, 1, \ldots, 1) \in \{0, 1\}^{n-1}$ the mapping defined in (6) satisfies $f_{\alpha_{2n}} = \overline{\text{rev}} \bullet \pi$, the sequence of edges of the path P when traversing it from its first to its last vertex has the form $(e_1, E_2, e_3, E_4, e_5)$, where e_1, e_3 and e_5 are single edges and E_2 and E_4 are sequences of edges that satisfy the following conditions: There is a path $P' \in \mathcal{P}_{2n}(n, n+1)$ and a path $P'' \in \widetilde{\mathcal{P}}_{2n}(n, n+1)$ (this is the crucial difference to before, where P' and P'' where both from the set $\mathcal{P}_{2n}(n, n+1)$) such that

$$e_1 = \big(S(P') \circ 00, S(P') \circ 01\big) \ ,$$
$$e_3 = \big(L(P') \circ 01, L(P') \circ 11\big) = \big(L(P') \circ 01, \overline{\text{rev}}(\pi(L(P''))) \circ 11\big) \ ,$$
$$e_5 = \big(\overline{\text{rev}}(\pi(F(P''))) \circ 11, \overline{\text{rev}}(\pi(F(P''))) \circ 01\big) \ ,$$

E_2 is given by traversing the edges of $P' \circ 01$ starting at the vertex $S(P') \circ 01$ and ending at the vertex $L(P') \circ 01$, and E_4 is given by the traversing all the edges of $\overline{\text{rev}}(\pi(P'')) \circ 11$ in reverse order starting at the vertex $\overline{\text{rev}}(\pi(L(P''))) \circ 11$ and ending at the vertex $\overline{\text{rev}}(\pi(F(P''))) \circ 11$.

As in the proof of Lemma 4, we distinguish five cases depending on which part of P the edge (u, v) belongs to. The last two cases are treated analogously to cases 3.4 and 3.5 in the proof of Lemma 4, using (17a) and (17b) instead of (16a) and (16c). The first three cases are exactly the same (not just analogous) as cases 3.1, 3.2 and 3.3. In particular, in case 3.2 where (u, v) is an edge from E_2 note that the edges in E_2 originate from a path $P' \in \mathcal{P}_{2n}(n, n+1)$, which is accounted for by setting the value of the variable $\texttt{flip}$ to $\texttt{false}$ in the recursive call in line P18 of the algorithm $\textsc{Paths}()$. $\quad\square$

Given Lemma 4 and Lemma 5, we may from now on use the notations $\mathcal{P}_{2n}(k, k+1)$ and $\widetilde{\mathcal{P}}_{2n}(k, k+1)$ interchangeably for the sets of paths defined in Sections 3.1 and 3.2 and for the sets of paths computed by the algorithm PATHS() called with parameter $\texttt{flip} = \texttt{false}$ or $\texttt{flip} = \texttt{true}$, respectively.

3.4. Correctness of the algorithm HAMCYCLE().

3.4.1. HAMCYCLE() *computes a 2-factor.* By combining Lemma 4, property (a) from Section 3.1 and the relations (16a) and (16c), we obtain that the paths $\mathcal{P}_{2n}(n, n+1)$ computed by the algorithm PATHS() indeed have properties (i) and (ii) claimed in Section 2.2. Consequently, assuming for a moment that the auxiliary functions $\text{ISFLIPTREE}_1()$ and $\text{ISFLIPTREE}_2()$ always return $\texttt{false}$ (i.e., ISFLIPVERTEX() always returns $\texttt{false}$), the arguments given in Sections 2.3 and 2.4 show that the algorithm HAMCYCLE() correctly computes the 2-factor $\mathcal{C}_{2n+1}$ defined in (4) in the middle levels graph $Q_{2n+1}(n, n+1)$. We proceed to show that the algorithm HAMCYCLE() computes a different 2-factor (but still a 2-factor) for each possible choice of boolean functions $\text{ISFLIPTREE}_1()$ and $\text{ISFLIPTREE}_2()$ that are called from within ISFLIPVERTEX(). Later we prove that the functions $\text{ISFLIPTREE}_1()$ and $\text{ISFLIPTREE}_2()$ specified in Section 2.5 yield a 2-factor that consists only of a single cycle, i.e., a Hamilton cycle. As already mentioned in Section 2.3, the different 2-factors are obtained by replacing some flippable pairs of paths from $\mathcal{P}_{2n}(n, n+1)$ in the first set on the right hand side of (4) by the corresponding flipped pairs of paths from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$. There are two potential problems that the function ISFLIPVERTEX() as it is called from HAMCYCLENEXT() and HAMCYCLE() (see line N3 and line H7, respectively) could cause in this approach, and the next lemma shows that none of these problems occurs. First, the function might return $\texttt{true}$ for a vertex $x \in D_{2n}^{=0}(n)$ for which the path $P \in \mathcal{P}_{2n}(n, n+1)$ that starts with this vertex (i.e., $F(P) = x$) is not contained in a flippable pair, so the subsequent calls to PATHS() with parameter $\texttt{flip} = \texttt{true}$ would produce undefined output. Second, given a flippable pair of paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$, the results of the calls $\text{ISFLIPVERTEX}(n, x)$ and $\text{ISFLIPVERTEX}(n, x')$ with $x := F(P)$ and $x' := F(P')$ might be different/inconsistent, so our algorithm would not compute a valid 2-factor.

Lemma 6. *Let $n \geq 1$, and let the sets $\mathcal{P}_{2n}(n, n+1)$ and $\mathcal{X}_{2n}(n, n+1)$ be as in Lemma 5. Furthermore, let $\text{ISFLIPTREE}_1()$ and $\text{ISFLIPTREE}_2()$ be arbitrary boolean functions on the sets of ordered rooted trees $\mathcal{T}_{n,1}^*$ and $\mathcal{T}_{n,2}^*$ defined in Section 2.5, respectively, and let ISFLIPVERTEX() be as defined in Algorithm 4. For any lattice path $x \in D_{2n}^{=0}(n)$ with $\text{ISFLIPVERTEX}(n, x) = \textbf{true}$ there is another lattice path $x' \in D_{2n}^{=0}(n)$ with $\text{ISFLIPVERTEX}(n, x') = \textbf{true}$ and two paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$ with $(F(P), F(P')) = (x, x')$ that form a flippable pair $(P, P') \in \mathcal{X}_{2n}(n, n+1)$.*

Proof. By the definitions in lines T1–T5, the tree $T := h^{-1}(x)$ is contained in exactly one of the four sets $\mathcal{T}_{n,1}^*$, $\tau_1(\mathcal{T}_{n,1}^*)$, $\mathcal{T}_{n,2}^*$, $\tau_2(\mathcal{T}_{n,2}^*)$. All these sets of trees are disjoint. We consider four cases depending on which of the sets the tree T is contained in.

Case 1: If $T \in \mathcal{T}_{n,1}^*$, then by the instructions in line T2 we have $\text{ISFLIPTREE}_1(T) = \texttt{true}$. Now consider the tree $T' := \tau_1(T)$ and the lattice path $x' := h(T') = h(\tau_1(T)) \in D_{2n}^{=0}(n)$. Clearly, the instructions in line T3 ensure that $\text{ISFLIPVERTEX}(n, x') = \texttt{true}$. By the definition of $\mathcal{T}_{n,1}^*$ and τ_1, the lattice paths corresponding to the trees T and T' satisfy the preconditions of [Müt16, Lemma 21], implying that the pair of lattice paths $(h(T), h(T')) = (x, x')$ is contained in the set H_1 defined in [Müt16, Lemma 24], so by this lemma there are two paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$ with $(F(P), F(P')) = (x, x')$ that form a flippable pair $(P, P') \in \mathcal{X}_{2n}(n, n+1)$.

Case 2: If $T \in \tau_1(\mathcal{T}_{n,1}^*)$, then by the instructions in line T3 we have $\text{ISFLIPTREE}_1(\tau_1^{-1}(T)) = \texttt{true}$. Now consider the tree $T' := \tau_1^{-1}(T)$ and the lattice path $x' := h(T') = h(\tau_1^{-1}(T)) \in D_{2n}^{=0}(n)$. Clearly, the instructions in line T2 ensure that $\text{ISFLIPVERTEX}(n, x') = \texttt{true}$. From here the proof continues as in case 1, however, the roles of T and T' are interchanged.

The cases $T \in \mathcal{T}_{n,2}^*$ and $T \in \tau_2(\mathcal{T}_{n,2}^*)$ are dealt with analogously to case 1 and case 2, respectively, replacing the application of [Müt16, Lemma 21] by [Müt16, Lemma 22] and the set H_1 by H_2 defined in [Müt16, Lemma 24]. □

The following simple but powerful lemma follows immediately from the definition of flippable/flipped pairs of paths given in Section 2.2.

Lemma 7. *Let G be a graph, $\mathcal{C}$ a 2-factor in G, C and C' two cycles in $\mathcal{C}$, (P, P') a flippable pair of paths such that P is contained in C and P' is contained in C', and (R, R') a corresponding flipped pair of paths. Replacing the paths P and P' in the 2-factor $\mathcal{C}$ by R and R' yields a 2-factor in which the two cycles C and C' are joined to a single cycle.*

The Hamilton cycle computed by our algorithm HAMCYCLE() is obtained from the 2-factor $\mathcal{C}_{2n+1}$ defined in (4) by repeatedly applying the flipping operation from Lemma 7. Specifically, we replace several flippable pairs of paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$ by the corresponding flipped paths $R, R' \in \widetilde{\mathcal{P}}_{2n}(n, n+1)$ in the first set on the right hand side of (4) such that all cycles of the 2-factor $\mathcal{C}_{2n+1}$ are joined to a single cycle. This of course requires that all flippable pairs of paths are disjoint, which is guaranteed by the definitions of $\mathcal{P}_{2n}(n, n+1)$ and $\mathcal{X}_{2n}(n, n+1)$: The paths in $\mathcal{P}_{2n}(n, n+1)$ are all disjoint, and each path from this set appears in at most one flippable pair in the set $\mathcal{X}_{2n}(n, n+1)$. To reasonably choose which flippable pairs of paths to replace by the corresponding flipped paths, we need to understand the cycle structure of the 2-factor $\mathcal{C}_{2n+1}$.

3.4.2. *Structure of cycles in the 2-factor $\mathcal{C}_{2n+1}$.* To describe the structure of the cycles in the 2-factor $\mathcal{C}_{2n+1}$ defined in (4), we introduce the concept of plane trees.

Plane trees. A *plane tree* is a tree with a cyclic ordering of all neighbors of each vertex. We think of a plane tree as a tree embedded in the plane such that for each vertex v the order in which its neighbors are encountered when walking around v in counterclockwise direction is precisely the specified cyclic ordering, see the middle of Figure 4. We denote by $\mathcal{T}_n$ the set of all plane trees with n edges. The notions of thin/thick leaves and of clockwise/counterclockwise-next leaves introduced in Section 2.5 for ordered rooted trees can be defined for plane trees in a completely analogous fashion.

Transformations plane() *and* ROOT() *between ordered rooted trees and plane trees.* For any ordered rooted tree $T^* \in \mathcal{T}_n^*$, we define a plane tree $T = \text{plane}(T^*) \in \mathcal{T}_n$ as follows: The underlying (abstract) tree of T^* and T is the same. For the root r of T^*, the cyclic ordering of neighbors of r in T is given by the left-to-right ordering of the children of r in T^*. For any other vertex v of T^*, if $(u_1, u_2, \ldots, u_k)$ is the left-to-right ordering of the children of v in T^* and w is the parent of v, then we define $(w, u_1, u_2, \ldots, u_k)$ as the cyclic ordering of neighbors of v in T.

For any plane tree $T \in \mathcal{T}_n$ and any edge (r, u) of T, we define an ordered rooted tree $T^* = \text{ROOT}(T, (r, u)) \in \mathcal{T}_n^*$ as follows: The underlying (abstract) tree of T and T^* is the same. The vertex r is the root of T^*, and if $(u_1, u_2, \ldots, u_k)$ with $u_1 = u$ is the cyclic ordering of neighbors of r in T, then $(u_1, u_2, \ldots, u_k)$ is the left-to-right ordering of the children of r in T^*. For any other vertex v of T, if w is the vertex on the path from v to r and $(u_0, u_1, \ldots, u_k)$ with $u_0 = w$ is the cyclic ordering of neighbors of v in T, then $(u_1, u_2, \ldots, u_k)$ is the left-to-right ordering of the children of v in T^*.

Informally speaking, plane(T^*) is obtained from T^* by 'forgetting' the root vertex, and ROOT($T, (r, u)$) is obtained from T by 'pulling out' the vertex r as root such that u is the leftmost child of the root, see Figure 4.

For any cycle C of the 2-factor $\mathcal{C}_{2n+1}$ defined in (4) we let $F(C)$ denote the cyclic sequence of all vertices of the form $F(P)$, $P \in \mathcal{P}_{2n}(n, n+1)$, for which the path $P \circ 0$ is contained in C (when walking along C in the direction such that $P \circ 0$ is traversed starting at the first vertex and ending at the
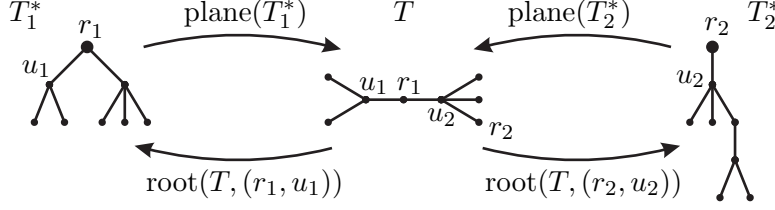
FIGURE 4. Two ordered rooted trees $T_1^*, T_2^* \in \mathcal{T}_7^*$ (left and right), a plane tree $T \in \mathcal{T}_7$ (middle), and the transformations between them.
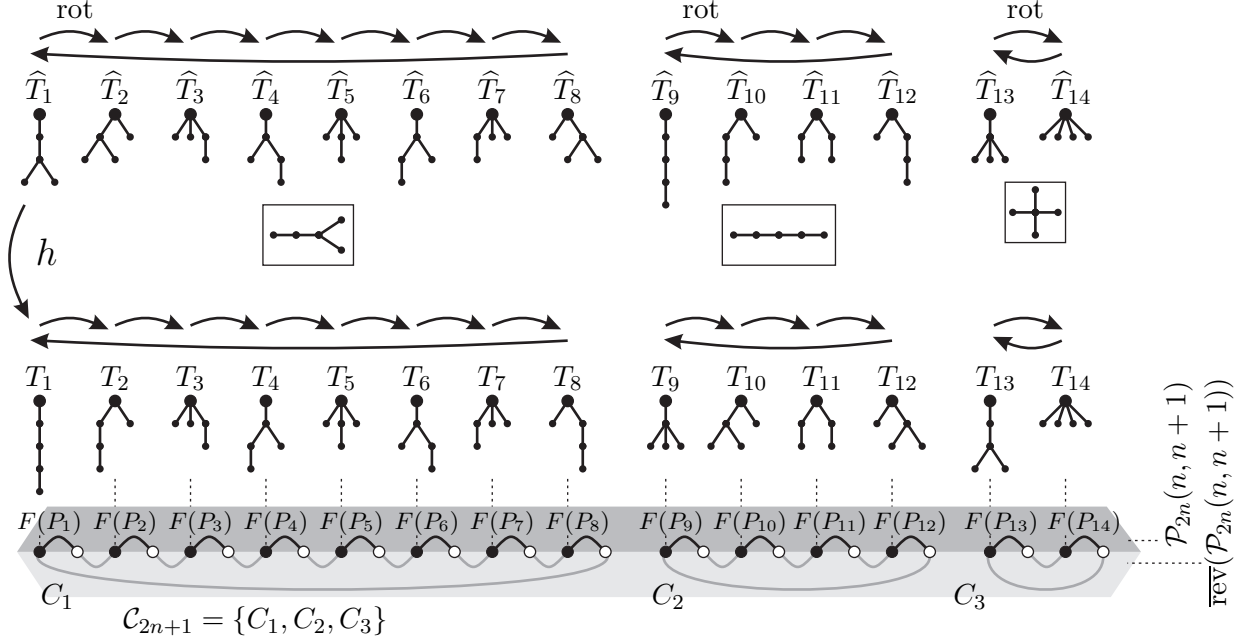

FIGURE 5. Structure of the 2-factor $\mathcal{C}_{2n+1}$ for $n = 4$ (cf. the bottom part of Figure 1). The figure shows the cyclic sequences $F(C)$ of ordered rooted trees for all cycles $C \in \mathcal{C}_{2n+1}$ (bottom) and their preimages under the mapping h (top). Note that trees rotate along each of the sequences $h^{-1}(F(C))$, $C \in \mathcal{C}_{2n+1}$. The boxes show the plane trees associated with each cycle $C \in \mathcal{C}_{2n+1}$. The sets of trees $\mathcal{T}_{n,1}^*$ and $\mathcal{T}_{n,2}^*$ defined in Section 2.5 are given by $\mathcal{T}_{n,1}^* = \{\widehat{T}_7, \widehat{T}_{11}\}$ and $\mathcal{T}_{n,2}^* = \{\widehat{T}_6\}$ and we have $\tau_1(\widehat{T}_7) = \widehat{T}_{14}$, $\tau_1(\widehat{T}_{11}) = \widehat{T}_3$ and $\tau_2(\widehat{T}_6) = \widehat{T}_9$.

last vertex). It turns out to be very useful to consider the ordered rooted trees corresponding to the elements of the sequence $F(C)$ (recall (16a) and the bijection between lattice paths in $D_{2n}^{=0}(n)$ and ordered rooted trees in $\mathcal{T}_n^*$). The following lemma is a slight reformulation of [Müt16, Lemma 20]. For the reader's convenience, the lemma is illustrated in Figure 5.

Lemma 8. *For any $n \geq 1$ and any cycle C of the 2-factor $\mathcal{C}_{2n+1}$ defined in (4), considering the cyclic sequence of ordered rooted trees $(T_1, T_2, \ldots, T_k) := F(C)$ we have $h^{-1}(T_{i+1}) = \mathrm{rot}(h^{-1}(T_i))$ for all $i = 1, 2, \ldots, k$ (indices are considered modulo k) with h^{-1} and rot as defined in Section 2.5. I.e., we can associate the cycle C with the plane tree $\mathrm{plane}(h^{-1}(T_1)) = \cdots = \mathrm{plane}(h^{-1}(T_k)) \in \mathcal{T}_n$. Moreover, for any plane tree $T \in \mathcal{T}_n$ there is a cycle $C \in \mathcal{C}_{2n+1}$ to which the tree T is associated. In particular, the number of cycles of the 2-factor is $|\mathcal{C}_{2n+1}| = |\mathcal{T}_n|$.*

3.4.3. HamCycle() *computes a Hamilton cycle.* Lemmas 6, 7 and 8 motivate the following definitions of two directed multigraphs $\mathcal{G}_n$ and $\mathcal{H}_n \subseteq \mathcal{G}_n$ that are illustrated in Figure 6.

Graphs $\mathcal{G}_n$ and $\mathcal{H}_n$. The nodes of $\mathcal{G}_n$ are sets of ordered rooted trees from $\mathcal{T}_n^*$ such that all trees that differ by the rotation operation rot are in the same set. The edges of $\mathcal{G}_n$ are defined as follows: For $i \in \{1, 2\}$ and any pair of trees (T, T') with $T \in \mathcal{T}_{n,i}^*$ and $T' = \tau_i(T)$ we add an edge directed from the set of trees containing T to the set of trees containing T'. We refer to the edges of $\mathcal{G}_n$ induced by the mappings τ_1 and τ_2 as τ_1-edges or τ_2-edges, respectively. The graph $\mathcal{H}_n$ has the same nodes as $\mathcal{G}_n$, but only a subset of its edges. Specifically, for $i \in \{1, 2\}$ and any pair of trees (T, T') with $T \in \mathcal{T}_{n,i}^*$ and $T' = \tau_i(T)$ we only add an edge directed from the set of trees containing T to the set of trees containing T' if $\text{IsFlipTree}_i(T) = \texttt{true}$, where $\text{IsFlipTree}_1()$ and $\text{IsFlipTree}_2()$ are as defined in Section 2.5.

As each node of $\mathcal{G}_n$ and $\mathcal{H}_n$ is a set $\{T_1, T_2, \ldots, T_k\}$ of ordered rooted trees that differ by rotation, i.e., we have $T_{i+1} = \text{rot}(T_i)$ for $i = 1, 2, \ldots, k$, we can identify this node with the plane tree $\text{plane}(T_1) = \cdots = \text{plane}(T_k) \in \mathcal{T}_n$, see the example in Figure 6 for $n = 6$. Also, we can interpret the result of the operations τ_1 and τ_2 that define the edges of $\mathcal{G}_n$ and $\mathcal{H}_n$ in terms of the corresponding plane trees. Note e.g. that τ_1 increases the number of leaves by one, and τ_2 decreases the number of leaves by one, implying that neither $\mathcal{G}_n$ nor $\mathcal{H}_n$ has any loops. This effect can be seen in Figure 6, where all τ_1-edges go from right to left, and all τ_2-edges from left to right. The graph $\mathcal{G}_n$ may have multiple edges between nodes, see Figure 6. The τ_1- and τ_2-edges added to the graph G_n are also described in Figure 5 for the smaller example $n = 4$.

By Lemma 8, the nodes of $\mathcal{G}_n$ and $\mathcal{H}_n$ correspond to the cycles of the 2-factor $\mathcal{C}_{2n+1}$ defined in (4). Applying Lemma 6 with boolean functions $\text{IsFlipTree}_1()$ and $\text{IsFlipTree}_2()$ that always return $\texttt{true}$ shows that each edge of $\mathcal{G}_n$ corresponds to a flippable pair of paths from the set $\mathcal{X}_{2n}(n, n+1)$ for which the paths are contained in the corresponding cycles. In other words, the edges of $\mathcal{G}_n$ capture all potential flipping operations that could be performed to modify the 2-factor $\mathcal{C}_{2n+1}$. Applying Lemma 6 with the boolean functions $\text{IsFlipTree}_1()$ and $\text{IsFlipTree}_2()$ defined in Section 2.5 shows that each edge of $\mathcal{H}_n$ corresponds to a flippable pair of paths from $\mathcal{P}_{2n}(n, n+1)$ that are actually replaced by the corresponding flipped paths from $\widetilde{\mathcal{P}}_{2n}(n, n+1)$ in the first set on the right hand side of (4) by our Hamilton cycle algorithm HamCycle(). By Lemma 7 the edges of $\mathcal{H}_n$ therefore indicate which pairs of cycles from the 2-factor $\mathcal{C}_{2n+1}$ the algorithm joins to a single cycle. Consequently, to complete the correctness proof for the algorithm HamCycle(), it suffices to show the following lemma.

Lemma 9. *For any $n \geq 1$, the graph $\mathcal{H}_n$ is a spanning tree of $\mathcal{G}_n$.*

Proof. For $n = 1$ and $n = 2$ the graph $\mathcal{G}_n$ consists only of a single node and no edges, and we have $\mathcal{G}_n = \mathcal{H}_n$, so in these cases the claim is trivially true. For $n = 3$ the graph $\mathcal{G}_n$ consists of two nodes, connected by a τ_1-edge, and we again have $\mathcal{G}_n = \mathcal{H}_n$, proving the lemma also in this case. For the rest of the proof we assume that $n \geq 4$.

Let $\mathcal{H}_n'$ denote the graph obtained from $\mathcal{H}_n$ by removing all τ_2-edges. We first show that $\mathcal{H}_n'$ is a spanning forest of $\mathcal{G}_n$.

By the definition of the function $\text{IsFlipTree}_1()$ from Section 2.5, the outdegree of every node $T \in \mathcal{T}_n$ in $\mathcal{H}_n'$ is at most 1. More precisely, if T has a thin leaf, then the outdegree is 1, as there is a way to root T such that it is contained in the set $\mathcal{T}_{n,1}^*$ (let u denote a thin leaf, u' the neighbor of u and v the neighbor of u', then this rooted tree is given by $\text{Root}(T, (v, u'))$, see the left hand side of Figure 3), and only for the lexicographically smallest such tree the function returns $\texttt{true}$. On the other hand, it T has no thin leaves, then the outdegree is 0, as there is no way to root T such that it is contained in $\mathcal{T}_{n,1}^*$, so the function returns $\texttt{false}$ for all these trees (see Figure 6). Moreover, as
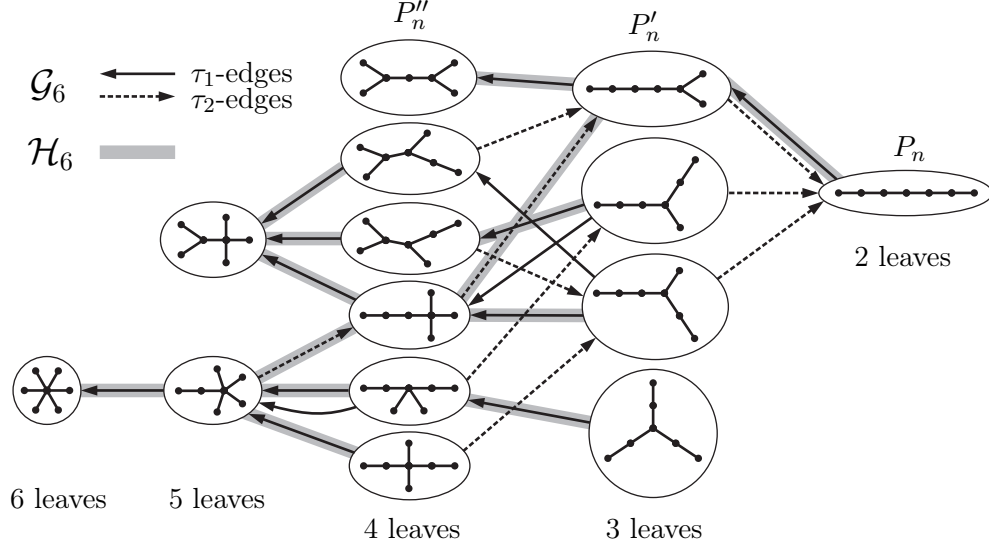
FIGURE 6. The graphs $\mathcal{G}_6$ and $\mathcal{H}_6$ with nodes are arranged in levels according to the number of leaves of the corresponding plane trees. The figure shows for each node, i.e., for each set of ordered rooted trees that differ by rotation, the plane tree corresponding to these trees. In the figure, τ_1-edges are drawn as solid lines, τ_2-edges as dashed lines, and the edges of $\mathcal{H}_6$ are highlighted in grey.

τ_1 increases the number of leaves by one, the graph $\mathcal{H}'_n$ is acyclic. An acyclic directed graph with outdegree at most 1 at every node is a forest, proving that $\mathcal{H}'_n$ is indeed a spanning forest of $\mathcal{G}_n$.

In the following we argue that adding the τ_2-edges from $\mathcal{H}_n$ to $\mathcal{H}'_n$ turns the spanning forest into a spanning tree of $\mathcal{G}_n$.

By what we said before the components (=trees) of the forest $\mathcal{H}'_n$ can be characterized precisely: Every component has a unique sink, i.e., a node with no outgoing edges, which corresponds to a plane tree with no thin leaves. Also, the distance of a node $T \in \mathcal{T}_n$ from the sink of the component it belongs to equals the number of thin leaves of T. Moreover, we obtain that the number of components of $\mathcal{H}'_n$ is given by the number of plane trees from $\mathcal{T}_n$ that have no thin leaves. In Figure 6, $\mathcal{H}'_n$ is obtained from $\mathcal{H}_n$ by removing two dashed τ_2-edges, resulting in three components whose sinks appear as the leftmost node of each component.

We now consider a component of $\mathcal{H}'_n$ that will play a special role in the following. Let $P_n \in \mathcal{T}_n$ denote the path with n edges, and let $(v_1, v_2, \ldots, v_{n+1})$ be the sequence of vertices along this path. We denote by $P'_n \in \mathcal{T}_n$ the graph obtained from P_n by replacing the edge (v_n, v_{n+1}) by the edge (v_{n-1}, v_{n+1}). Furthermore, we denote by $P''_n \in \mathcal{T}_n$ the graph obtained from P'_n by replacing the edge (v_1, v_2) by the edge (v_1, v_3). Clearly, $\mathcal{H}'_n$ has a component consisting of the three nodes P_n, P'_n and P''_n, with a τ_1-edge directed from P_n to P'_n and a τ_1-edge directed from P'_n to the sink P''_n, see the top part of Figure 6. We denote this component of $\mathcal{H}'_n$ by $\mathcal{S}_n$. Note that the definition of $\mathcal{S}_n$ is meaningful only for $n \geq 4$.

We proceed to show the following: (i) In every component of $\mathcal{H}'_n$ except $\mathcal{S}_n$ there is exactly one node that has an outgoing τ_2-edge in $\mathcal{H}_n$, and the component $\mathcal{S}_n$ has no node with an outgoing τ_2-edge. (ii) The τ_2-edges of $\mathcal{H}_n$ form an acyclic graph on the components of $\mathcal{H}'_n$ (in particular, every τ_2-edge of $\mathcal{H}_n$ starts and ends in a different component of $\mathcal{H}'_n$). Combining our earlier observation that $\mathcal{H}'_n$ is a spanning forest of $\mathcal{G}_n$ with claims (i) and (ii) clearly proves the lemma, so it remains to prove (i) and (ii).
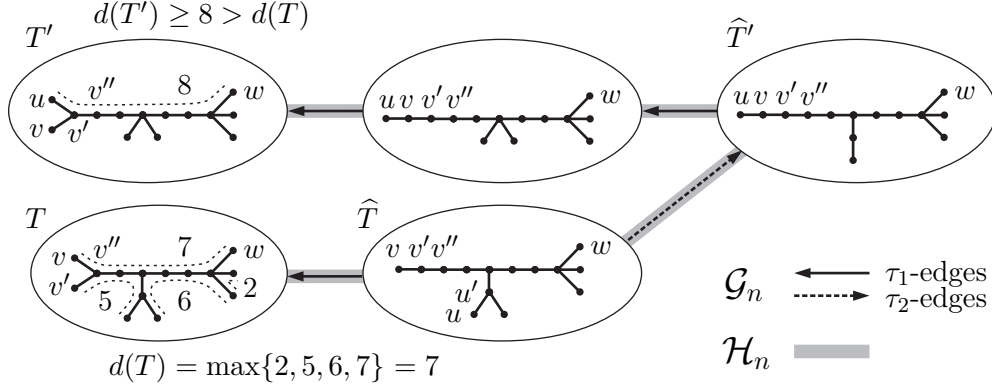
FIGURE 7. Illustration of the notations used in the proof of Lemma 9.

Proof of (i): Consider a component of $\mathcal{H}'_n$ with sink $T \in \mathcal{T}_n$, and let $N(T)$ be the set of all nodes in distance 1 from T in $\mathcal{H}'_n$. As T has no thin leaves, it can be rooted so that it is an image of τ_1: let u and u' be two leaves that have a common neighbor v such that u' is the counterclockwise-next leaf from u, then this rooted tree is given by $\textsc{Root}(T, (v, u))$. See the left hand side of Figure 3. The preimage is a tree with exactly one thin leaf, which corresponds to a node $T' \in \mathcal{T}_n$ with a τ_1-edge directed from T' to T that is present in the graph $\mathcal{H}'_n$. It follows that the set $N(T)$ is nonempty. Note that the function $\textsc{IsFlipTree}_2()$ returns $\texttt{true}$ only if the given tree has a single thin leaf, so the only outgoing τ_2-edges of $\mathcal{H}_n$ that start from a node in the same component as T start at a node in $N(T)$. Moreover, as any tree in $N(T)$ can be rooted uniquely so that it is a preimage of τ_2 (let v denote the thin leaf, u the counterclockwise-next leaf from v, u' the neighbor of u and r the neighbor of u' next to u in the counterclockwise ordering of the neighbors of u', then this rooted tree is given by $\textsc{Root}(T, (r, u'))$, see the right hand side of Figure 3), every node in $N(T)$ has exactly one outgoing τ_2-edge in $\mathcal{G}_n$, see Figure 6. If the component containing T is different from $\mathcal{S}_n$, then the function $\textsc{IsFlipTree}_2()$ returns $\texttt{true}$ for exactly one of these rooted trees: The function first computes a rooted version of T for each of them, thus consistently assigning an integer value to each of the trees (the integers are derived from distances between certain vertices of T, but the origin of these values is irrelevant for this part of the argument), and returns $\texttt{true}$ only for the tree that was assigned the largest value and that is lexicographically smallest among all trees with the same value. However, if the component containing T is $\mathcal{S}_n$ (i.e., $T = P_n$), then $N(T) = \{P'_n\}$ and the unique way to root P'_n such that it is a preimage of τ_2 is the tree $1^{n-1} \circ 0^{n-2} \circ 100$, for which the function $\textsc{IsFlipTree}_2()$ exceptionally returns $\texttt{false}$. This exceptional rooted tree is not encountered in any component other than $\mathcal{S}_n$.

Proof of (ii): For the reader's convenience, this part of the proof is illustrated in Figure 7. For any plane tree $T \in \mathcal{T}_n$, we define $d(T)$ as the maximum distance between any two leaves v and w that satisfy the following conditions: w is the clockwise-next leaf from v and the counterclockwise-next leaf v' from v has distance 2 from v. In other words, the leaves v and v' are adjacent to a common vertex v''; see the bottom left part of Figure 7. Claim (ii) is an immediate consequence of the following observation: For any τ_2-edge from $\mathcal{H}_n$ directed from a component of $\mathcal{H}'_n$ with sink $T \in \mathcal{T}_n$ to a component of $\mathcal{H}'_n$ with sink $T' \in \mathcal{T}_n$, we have $d(T) < d(T')$. In particular, the edge starts and ends in a different component of $\mathcal{H}'_n$. To see this let $\widehat{T} \in \mathcal{T}_n$ be the node in distance 1 of T in $\mathcal{H}'_n$ that is the starting node of a τ_2-edge of $\mathcal{H}_n$ (this node exists and is unique by claim (i)), and let $\widehat{T}' \in \mathcal{T}_n$ be the end node of this τ_2-edge. As $\widehat{T}$ has exactly one thin leaf, and as τ_2 preserves one thin leaf and possibly creates one additional thin leaf, $\widehat{T}'$ has distance 1 or 2 from T'. So in $\mathcal{H}_n$ there is a

τ_1-edge directed from $\widehat{T}$ to T, a τ_2-edge directed from $\widehat{T}$ to $\widehat{T}'$ and either a single τ_1-edge or a path consisting of two τ_1-edges directed from $\widehat{T}'$ to T', see Figure 7.

By the definition of $\text{IsFlipTree}_2()$ and of the parameter $d(T)$, the tree T has two leaves v and w in distance $d(T)$ such that w is the clockwise-next leaf from v and the counterclockwise-next leaf v' from v has distance 2 from v, implying that v and v' have a common adjacent vertex v''. Furthermore $\widehat{T}$ is obtained from T by replacing the edge (v, v'') by (v, v'). The function $\text{IsFlipTree}_2()$ of course computes a rooted version of T from a rooted version of $\widehat{T}$.

By the definition of τ_2, the tree $\widehat{T}'$ is obtained from $\widehat{T}$ by considering the counterclockwise-next leaf u from v, its neighbor u' in $\widehat{T}$, and by replacing the edge (u, u') by the edge (u, v). By the definition of τ_1, T' is obtained from $\widehat{T}'$ by replacing the edge (u, v) by (u, v') plus possibly another application of τ_1 if $\widehat{T}'$ has two thin leaves. It follows that the distance between u and w in T' is $d(T) + 1$, implying that $d(T') > d(T)$, as claimed.

This completes the proof. □

Remark 10. Here is how our algorithm can be modified to compute another Hamilton cycle in the middle levels graph: The key is to modify the functions $\text{IsFlipTree}_1()$ and $\text{IsFlipTree}_2()$ such that they define a spanning tree of $\mathcal{G}_n$ that is different from the 'canonical' spanning tree $\mathcal{H}_n$. Each different spanning tree of $\mathcal{G}_n$ clearly yields a different Hamilton cycle in the middle levels graph $Q_{2n+1}(n, n+1)$ (cf. [Müt16, Lemma 6]). The only limitation an efficient algorithm must obey is that the spanning tree must be locally computable, i.e., given information about the current node $T \in \mathcal{T}_n$ of $\mathcal{G}_n$, the algorithm must decide based on a polynomially sized neighborhood of T which edges of $\mathcal{G}_n$ to include and which to exclude from a spanning tree. Note that the degree of the nodes of $\mathcal{G}_n$ is $\mathcal{O}(n)$ and the total number of nodes is $|\mathcal{T}_n| = \Theta(4^n \cdot n^{-5/2})$.

4. Running time and space requirements of the algorithm

In this section we prove that our Hamilton cycle algorithm can be implemented so that it satisfies the runtime and space bounds claimed in the introduction. In the runtime analysis we apply some results about the paths $\mathcal{P}_{2n}(n, n+1)$ that have been established in [MW12].

4.1. **Running time.** A naive implementation of the function $\text{Paths}()$ takes time $\mathcal{O}(n^2)$: To see this observe that the membership tests whether x^- is contained in one of the sets $D^{=0}_{2n-2}(n-1)$, $D^-_{2n-2}(n-1)$ or $D^{>0}_{2n-2}(n)$ in lines P12, P15, P16, P17, P20 and P21 and the application of the mappings $\overline{\text{rev}}$ and π in line P22 take time $\mathcal{O}(n)$ (recall that $\overline{\text{rev}}^{-1} = \overline{\text{rev}}$ and $\pi^{-1} = \pi$), and that the value of n decreases by 1 with each recursive call. In the following we sketch how this can improved so that each call of $\text{Paths}()$ takes only time $\mathcal{O}(n)$. More details can be found in the comments of our C++ implementation [www]. For this we maintain counters c_0 and c_1 for the number of zeros and ones of a given bitstring $x = (x_1, x_2, \ldots, x_{2n})$. Moreover, interpreting the bitstring x as a lattice path (as described in Section 2.1), we maintain vectors c_{00}, c_{01}, c_{10}, c_{11} that count the number of occurences of pairs of consecutive bits (x_{2i}, x_{2i+1}), $i \in \{1, 2, \ldots, n-1\}$, *per height level of the lattice path* for each of the four possible value combinations of x_{2i} and x_{2i+1}. E.g., for the bitstring $x = 1100001010$ the vector c_{10} has a single nonzero entry 1 at height level (=index) 1 for the two bits $(x_2, x_3) = 10$, the vector c_{00} has a single nonzero entry 1 at height level (=index) 0 for the two bits $(x_4, x_5) = 00$, and the vector c_{01} has a single nonzero entry 2 at height level (=index) -1 for the pairs of bits $(x_6, x_7) = (x_8, x_9) = 01$. Using these counters, the three membership tests mentioned before can be performed in constant time. E.g., a bitstring $x = (x_1, x_2, \ldots, x_{2n})$ is contained in $D^{=0}_{2n}(n)$ if and only if $c_0 = c_1$ and $x_1 = 1$ and the entry of c_{00} at height level 0 equals 0 (i.e., the lattice path never moves below the line $y = 0$). Moreover, these counters can be updated in constant time when removing the last two bits of x and when applying the mappings $\overline{\text{rev}}$ and π: Note that $\overline{\text{rev}}$ simply

swaps the roles of c_0 and c_1 and the roles of c_{00} and c_{11} and possibly incurs an index shift, and that π simply swaps the roles of c_{10} and c_{01}. To compute the applications of $\overline{\mathrm{rev}}$ and π in line P22 in constant time, we do not modify x at all, but rather count the number of applications of $\overline{\mathrm{rev}}$ and π and keep track of the middle range of bits of x that is still valid. When removing the last two bits of x, this range shrinks by 2 on one of the sides. Taking into account that multiple applications of $\overline{\mathrm{rev}}$ and π cancel each other out, this allows us to compute the effect of applying those mappings *lazily* when certain bits are queried later on (when testing the values of the last two bits of some substring of x). E.g., by repeatedly removing the last two bits and applying the mappings $\overline{\mathrm{rev}}$ and π, the bitstring $x = (x_1, x_2, \ldots, x_{10})$ is transformed into $(\overline{x_8}, \overline{x_6}, \overline{x_7}, \overline{x_4}, \overline{x_5}, \overline{x_2}, \overline{x_3}, \overline{x_1})$, $(x_2, x_4, x_5, x_6, x_7, x_8)$, $(\overline{x_6}, \overline{x_4}, \overline{x_5}, \overline{x_2})$, and finally into (x_4, x_6).

The function IsFLIPVERTEX() can be implemented to run in time $\mathcal{O}(n^2)$. To see this observe that the result of applying the function h^{-1} in line T1 can be computed in time $\mathcal{O}(n^2)$, and that the functions IsFLIPTREE$_1$() and IsFLIPTREE$_2$() called in lines T2–T5 also need time $\mathcal{O}(n^2)$: For both we need to rotate an ordered rooted tree with n edges (=bitstring of length $2n$) for one full rotation, and each rotation operation takes time $\mathcal{O}(n)$.

It was shown in [MW12, Lemma 9] that the length of any path $P \in \mathcal{P}_{2n}(n, n+1)$ with a first vertex $F(P) =: x \in D_{2n}^{=0}(n)$ is given by the following simple formula: Considering the unique decomposition $x = 1 \circ x_\ell \circ 0 \circ x_r$ with $x_\ell \in D_{2k}^{=0}(k)$ for some $k \geq 0$, the length of P is given by $2|x_\ell| + 2 \leq 2(2n-2) + 2 = 4n - 2$. It follows that the while-loop in line H5 terminates after at most $\mathcal{O}(n)$ iterations, i.e., the initialization phase of HAMCYCLE() (lines H2–H7) takes time $\mathcal{O}(n^2)$.

It was shown in [MW12, Theorem 10] that the distance between any two neighboring vertices of the form $x \circ 0$, $x' \circ 0$ with $x, x' \in D_{2n}^{=0}(n)$ (x and x' are first vertices of two paths $P, P' \in \mathcal{P}_{2n}(n, n+1)$) on a cycle in (4) is exactly $4n + 2$. Comparing the lengths of two paths from $\mathcal{P}_{2n}(n, n+1)$ that form a flippable pair with the lengths of the corresponding flipped paths from the set $\widetilde{\mathcal{P}}_{2n}(n, n+1)$, we observe that either the length of one the paths decreases by 4 and the length of the other increases by 4, or the lengths of the paths do not change. Specifically, the paths P, P' and R, R' defined in (2) and (3) have exactly the length differences -4 and $+4$, and these differences only propagate through the first cases of the PATHS() recursion, but not the last case in lines P19–P22. It follows that every call of HAMCYCLENEXT() for which the condition in line N3 is satisfied and which therefore takes time $\mathcal{O}(n^2)$ due to the call of IsFLIPVERTEX(), is followed by at least $4n - 3$ calls in which the condition is not satisfied, in which case HAMCYCLENEXT() terminates in time $\mathcal{O}(n)$. Consequently, ℓ consecutive calls of HAMCYCLENEXT() take time $\mathcal{O}(n^2 + n\ell)$.

Summing up the time $\mathcal{O}(n^2)$ spent for the initialization phase and $\mathcal{O}(n^2 + n\ell)$ for the actual Hamilton cycle computation, we conclude that the algorithm HAMCYCLE(n, x, ℓ) runs in time $\mathcal{O}(n^2 + n\ell) = \mathcal{O}(n\ell(1 + \frac{n}{\ell}))$, as claimed.

4.2. **Space requirements.** The optimized variant of the algorithm PATHS() discussed in the previous section requires space $\mathcal{O}(n)$, e.g., to store the counting vectors $c_{00}, c_{01}, c_{10}, c_{11}$. The functions IsFLIPTREE$_1$() and IsFLIPTREE$_2$() require storing only a constant number of ordered rooted trees with n edges (=bitstrings of length $2n$) for lexicographic comparisons, so they also require space $\mathcal{O}(n)$. Furthermore, it is readily checked that the additional space required by each of the calling functions up to the top-level algorithm HAMCYCLE() is only $\mathcal{O}(n)$. This proves that the total space required by our algorithm is $\mathcal{O}(n)$.

5. ACKNOWLEDGEMENTS

for numerous valuable comments that helped improving the presentation of this work and the C++ code.

References

[BER76] J. Bitner, G. Ehrlich, and E. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Comm. ACM*, 19(9):517–521, 1976.

[BW84] M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Math.*, 48(2-3):163–171, 1984.

[Che08] Y. Chen. The Chung–Feller theorem revisited. *Discrete Math.*, 308(7):1328–1329, 2008.

[Cum66] R. Cummins. Hamilton circuits in tree graphs. *Circuit Theory, IEEE Transactions on*, 13(1):82–90, Mar 1966.

[DG12] P. Diaconis and R. Graham. *Magical mathematics*. Princeton University Press, Princeton, NJ, 2012. The mathematical ideas that animate great magic tricks, With a foreword by Martin Gardner.

[DKS94] D. Duffus, H. Kierstead, and H. Snevily. An explicit 1-factorization in the middle of the Boolean lattice. *J. Combin. Theory Ser. A*, 65(2):334–342, 1994.

[DSW88] D. Duffus, B. Sands, and R. Woodrow. Lexicographic matchings cannot form Hamiltonian cycles. *Order*, 5(2):149–161, 1988.

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

[EHR84] P. Eades, M. Hickey, and R. Read. Some Hamilton paths and a minimal change algorithm. *J. ACM*, 31(1):19–29, January 1984.

[EM84] P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Information Processing Letters*, 19(3):131–133, 1984.

[FT95] S. Felsner and W. Trotter. Colorings of diagrams of interval orders and α-sequences of sets. *Discrete Math.*, 144(1-3):23–31, 1995. Combinatorics of ordered sets (Oberwolfach, 1991).

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

[GŠ10] P. Gregor and R. Škrekovski. On generalized middle-level problem. *Inform. Sci.*, 180(12):2448–2457, 2010.

[Hav83] I. Havel. Semipaths in directed cubes. In *Graphs and other combinatorial topics (Prague, 1982)*, volume 59 of *Teubner-Texte Math.*, pages 101–108. Teubner, Leipzig, 1983.

[HH72] C. Holzmann and F. Harary. On the tree graph of a matroid. *SIAM Journal on Applied Mathematics*, 22(2):187–193, 1972.

[HKRR05] P. Horák, T. Kaiser, M. Rosenfeld, and Z. Ryjáček. The prism over the middle-levels graph is Hamiltonian. *Order*, 22(1):73–81, 2005.

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

[Joh04] R. Johnson. Long cycles in the middle two layers of the discrete cube. *J. Combin. Theory Ser. A*, 105(2):255–271, 2004.

[Knu11] D. Knuth. *The Art of Computer Programming, Volume 4A*. Addison-Wesley, 2011.

[KT88] H. Kierstead and W. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988.

[Luc87] J. Lucas. The rotation graph of binary trees is Hamiltonian. *Journal of Algorithms*, 8(4):503–535, 1987.

[LvBR93] J. Lucas, D. R. van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *Journal of Algorithms*, 15(3):343–366, 1993.

[MN17] T. Mütze and J. Nummenpalo. A constant-time algorithm for middle levels Gray codes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2238–2253, 2017.

[Müt16] T. Mütze. Proof of the middle levels conjecture. *Proc. London Math. Soc.*, 112(4):677–713, 2016. All references are to the unabridged version at *arXiv:1404.4442*.

[MW12] T. Mütze and F. Weber. Construction of 2-factors in the middle layer of the discrete cube. *J. Combin. Theory Ser. A*, 119(8):1832–1855, 2012.

[Rus88] F. Ruskey. Adjacent interchange generation of combinations. *Journal of Algorithms*, 9(2):162–180, 1988.

[SA11] M. Shimada and K. Amano. A note on the middle levels conjecture. *arXiv:0912.4564*, September 2011.

[Sav93] C. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35-A:97–108, 1993.

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

[Sed77] R. Sedgewick. Permutation generation methods. *ACM Comput. Surv.*, 9(2):137–164, June 1977.

[SS99] I. Shields and C. Savage. A Hamilton path heuristic with applications to the middle two levels problem. In *Proceedings of the Thirtieth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, Florida, 1999)*, volume 140, pages 161–178, 1999.

[SSS09] I. Shields, B. Shields, and C. Savage. An update on the middle levels problem. *Discrete Math.*, 309(17):5271–5277, 2009.

[Sta99]   R. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999. With a foreword by Gian-Carlo Rota and appendix 1 by Sergey Fomin.

[SW95]   C. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995.

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

[www]   currently `http://www.math.tu-berlin.de/~muetze` and `http://people.inf.ethz.ch/njerri`.