

Structural and Combinatorial Properties of 2-swap Word Permutation Graphs

Duncan Adamson^a, Nathan Flaherty^{b,c}, Igor Potapov^c, Paul G. Spirakis^c

^a*Department of Computer Science, University of St Andrews, North Haugh, KY18
9SX, United Kingdom*

^b*Leverhulme Research Centre, University of Liverpool, 51 Oxford St, Liverpool, L7
3NY, United Kingdom*

^c*Department of Computer Science, University of Liverpool, Ashton Street, Liverpool, L69
3BX, United Kingdom*

Abstract

In this paper, we study the graph induced by the 2-*swap* permutation (also known as a transposition) on words with a fixed Parikh vector. Informally, a 2-swap is a permutation which swaps exactly two symbols in a word, leaving all others unchanged. With these permutations, we define the *Configuration Graph*, $G(P)$ for a given Parikh vector. Each vertex in $G(P)$ corresponds to a unique word with the Parikh vector P , with an edge between any pair of words v and w if there exists a 2-swap s such that $v \circ s = w$. We provide several key combinatorial properties of this graph, including the exact diameter of this graph, the clique number of the graph, and the relationships between subgraphs within this graph. Additionally, we show that for every vertex in the graph, there exists a Hamiltonian path starting at this vertex. Finally, we provide an algorithm enumerating these paths from a given input word of length n with a delay of at most $O(\sigma \log n)$ between outputting edges, requiring $O(n \log n)$ preprocessing.

Keywords:

Combinatorics on words, Parikh vector, Graph algorithms, Permutation

1. Introduction

In information theory and computer science, there are several well-known edit distances between strings which are based on insertions, deletions and

substitutions of single characters or various permutations of several characters, including swaps of adjacent or non-adjacent characters, shuffling, etc. [1, 2, 3].

These operations are well motivated by problems in physical science, for example, the biological swaps which occur at a gene level are non-adjacent swap operations of two symbols (mutation swap operator) representing gene mutations [4]. In recent work on Crystal Structure Prediction the swap operation on a pair of symbols in a given word representing layers of atomic structures was used to generate new permutations of those layers, with the aim of exploring the configuration space of crystal structures [5]. In computer science string-to-string correction has been studied for adjacent swaps [6] and also in the context of sorting networks [7], motion on graphs and diameter of permutation groups [8]. In group theory, the distance between two permutations (the Cayley distance) measures the minimum number of transpositions of elements needed to turn one into the other [9].

A *configuration graph* is a graph where words (also known as strings) are represented by vertices and operations by edges between the strings. For example, one may define the operations as the standard suite of edits (insertions, deletions, and substitutions), with each edge corresponding to a pair of words at an edit distance of one. In such a graph, the distance between any pair of words corresponds to the edit distance between these words. In this paper, we study the structural properties of such graphs defined by swap operations of two symbols on a given word (2-swap permutations), a permutation defined by a pair of indices (i, j) and changing a word w by substituting the symbol at position i with that at position j , and the symbol at position j with that at position i . As the number of occurrences of each symbol in a given word can not be changed under this operation, we restrict our work to only those words with a given Parikh vector¹. We focus on studying several fundamental properties of the structure of these graphs, most notably the diameter, clique number, number of cliques, and the Hamiltonicity of the graph. Similar problems have been heavily studied for Cayley graphs [9], and permutation graphs [10]. It has been conjectured that the diameter of the symmetric group of degree n is polynomially bounded in n , where only recently the exponential upper bound [11] was replaced by a quasipolynomial

¹The Parikh vector of a word w denotes a vector with the number of occurrences of the symbols in the word w .

upper bound [12]. The diameter problem has additionally been studied with respect to a random pair of generators for symmetric groups [13]. In general, finding the diameter of a Cayley graph of a permutation group is NP-hard and finding the distance between two permutations in directed Cayley graphs of permutation groups is PSPACE-hard [14].

We also build upon previous work done into Combinatorial Gray Codes which, in general, give an ordering of some objects with two consecutive objects differ by a “small change”, The original Gray codes lists strings with subsequent strings having an edit distance of one. An extensive introduction to Combinatorial Gray codes can be found in [15]. In [16] Takoaka provides a combinatorial Gray code for multiset permutations (i.e. words) with constant delay however we wish to start from any word which is not a feature of their algorithm. Further, in Section 13.2.4 of [17] there is a $O(n)$ time algorithm for generating a combinatorial Gray code of multiset permutations with transpositions/2-swaps however this is provided without formal proof.

To develop efficient exploration strategies for these graphs it is essential to investigate its structural and combinatorial properties. As mentioned above the problem is motivated by problems arising in chemistry regarding Crystal Structure Prediction (CSP) which is computationally intractable in general [18, 19]. In current tools [5, 20], chemists rely on representing crystal structures as a multiset of discrete blocks, with optimisation performed via a series of permutations, corresponding to swapping blocks. Understanding reachability properties under the swap operations can help to evaluate and improve various heuristic space exploration tools and extend related combinatorial toolbox [21, 22].

1.1. Our Results

We provide several key combinatorial properties of the graph defined by 2-swap permutations over a given word. First, we show that this graph is *locally isomorphic*, that is, the induced subgraph of radius r centred on any pair of vertices w and u are isomorphic. We strengthen this by providing an exact diameter on the graph for any given Parikh vector. Finally, we show that, for every vertex v in the graph, there is a Hamiltonian path starting at v . We build upon this by providing a novel algorithm for enumerating the Hamiltonian path starting at any given vertex v in a binary graph with at most $O(\log n)$ delay between outputting the swaps corresponding to the transitions made in the graph, after $O(n \log n)$ preprocessing. We extend this to general alphabets, providing an algorithm enumerating the Hamiltonian

path with $O(\sigma \log n)$ delay between outputs after $O(n \log n)$ preprocessing. Our enumeration results correlate well with the existing work on the enumeration of words. This includes work on explicitly outputting each word with linear delay [23, 24], or outputting an implicit representation of each word with either constant or logarithmic delay relative to the length of the words [25, 26, 27, 28, 29]. The surveys [30, 31] provide a comprehensive overview of a wide range of enumeration results.

2. Preliminaries

Let $\mathbb{N} = \{1, 2, \dots\}$ denote the set of natural numbers, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. We denote by $[n]$ the set $\{1, 2, \dots, n\}$ and by $[i, n]$ the set $\{i, i + 1, \dots, n\}$, for all $i, n \in \mathbb{N}_0, i \leq n$. An *alphabet* Σ is an ordered, finite set of symbols. Tacitly assume that the alphabet $\Sigma = [\sigma] = \{1, 2, \dots, \sigma\}$, where $\sigma = |\Sigma|$. We treat each symbol in Σ both as a symbol and by the numeric value, i.e. $i \in \Sigma$ represents both the symbol i and the integer i . A *word* is a finite sequence of symbols from a given alphabet. The length of a word w , denoted $|w|$, is the number of symbols in the sequence. The notation Σ^n denotes the set of n -length words defined over the alphabet Σ , and the notation Σ^* denotes the set of all words defined over Σ .

For $i \in [|w|]$, the notation $w[i]$ is used to denote the i^{th} symbol in w , and for the pair $i, j \in [|w|], w[i, j]$ is used to denote the sequence $w[i]w[i + 1] \dots w[j]$, such a sequence is called a *factor* of w . We abuse this notation by defining, for any pair $i, j \in [|w|]$ such that $j < i$, $w[i, j] = \varepsilon$, where ε denotes the empty string.

Definition 1 (2-swap). Given a word $w \in \Sigma^n$ and pair $i, j \in [n], i < j$ such that $w[i] \neq w[j]$, the *2-swap* of w by (i, j) , denoted $w \circ (i, j)$, returns the word

$$w[1, i - 1]w[j]w[i + 1, j - 1]w[i]w[j + 1, n].$$

Example 1. Given the word $w = 11221122$ and pair $(2, 7)$, $w \circ (2, 7) = 12221112$.

Given a word $w \in \Sigma^n$, the *Parikh vector* of w , denoted $P(w)$ is the σ -length vector such that the i^{th} entry of $P(w)$ contains the number of occurrences of symbol i in w , formally, for $i \in [\sigma]$, $P(w)[i] = |\{j \in [n] \mid w[j] = i\}|$, where $n = |w|$. For example, the word $w = 11221122$ has Parikh vector

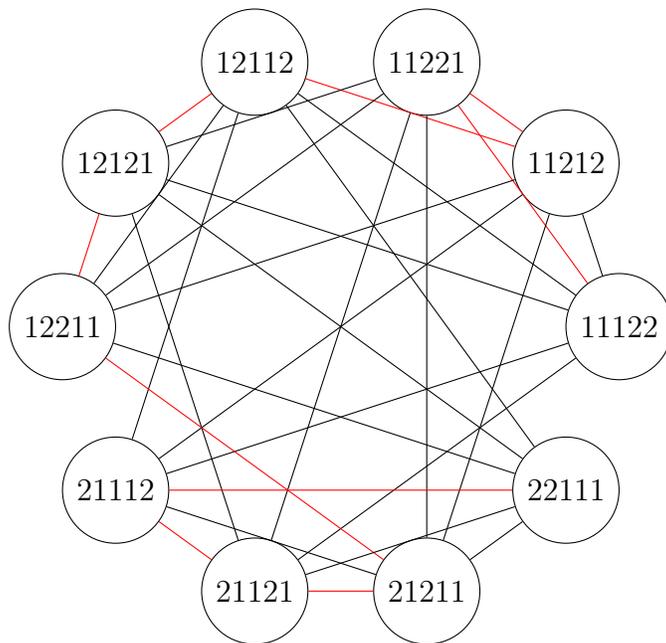


Figure 1: The configuration graph $G(3, 2)$ with Hamiltonian path shown in red.

(4, 4). The set of words with a given Parikh vector P over the alphabet Σ is denoted $\Sigma^{*|P}$, formally $\Sigma^{*|P} = \{w \in \Sigma^* \mid P(w) = P\}$. Unless stated otherwise we define $n := \sum_{i \in [\sigma]} P[i]$. It is notable that $|\Sigma^{*|P}| = \frac{n!}{\prod_{i \in [\sigma]} P[i]!}$ since it means the configuration graph (defined below) is of exponential size in n .

Definition 2. For a given alphabet Σ and Parikh vector P , the *configuration graph* of $\Sigma^{*|P}$ is the undirected graph $G(P) = \{V(P), E(P)\}$ where:

- $V(P) = \{v_w \mid w \in \Sigma^{*|P}\}$.
- $E(P) = \{\{v_w, v_u\} \in V(P) \times V(P) \mid \exists i, j \in [n] \text{ s.t. } w \circ (i, j) = u\}$.

Informally, the configuration graph for a given Parikh vector P is the graph with each vertex corresponding to some word in $\Sigma^{*|P}$, and each edge connecting every pair of words $w, u \in \Sigma^{*|P}$ such that there exists some 2-swap transforming w into u . Figure 1 provides an example of the configuration graph when $P = (3, 2)$.

A *path* (also called a *walk*) in a graph is an ordered set of edges such that the second vertex in the i^{th} edge is the first vertex in the $(i + 1)^{\text{th}}$ edge,

i.e. $p = \{(v_1, v_2), (v_2, v_3), \dots, (v_{|p|}, v_{|p|} + 1)\}$. Note that a path of length i visits $i + 1$ vertices. A path p *visits* a vertex v if there exists some edge $e \in p$ such that $v \in e$. A *cycle* (also called a *circuit*) is a path such that the first vertex visited is the same as the last. A *Hamiltonian* path p is a path visiting each vertex exactly once, i.e. for every $v \in V$, there exists at most two edges $e_1, e_2 \in p$ such that $v \in e_1$ and $v \in e_2$. A cycle is Hamiltonian if it is a Hamiltonian path and a cycle. A path p *covers* a set of vertices V if, for every $v \in V$, there exists some $e \in p$ such that $v \in e$. Note that a Hamiltonian path is a path cover of every vertex in the graph. and a Hamiltonian cycle is a cycle cover of every vertex in the graph. We use the notation $\max_{<i} A$ to denote the largest value in A which is less than i . And similarly $\min_{>i} A$ denotes the smallest value in A which is greater than i .

The *distance* between a pair of vertices $v, u \in V$, denoted $D(v, u)$ in the graph G is the smallest value $d \in \mathbb{N}_0$ for which there exists some path p of length d covering both v and u , i.e. the minimum number of edges needed to move from v to u . If $v = u$, then $D(v, u)$ is defined as 0. The *diameter* of a graph G is the maximum distance between any pair of vertices in the graph, i.e. $\max_{v, u \in V} D(v, u)$.

Given two graphs $G = (V, E)$ and $G' = (V', E')$, G is *isomorphic* to G' if there exists a bijective mapping $f : V \mapsto V'$ such that, for every $v, u \in V$, $(v, u) \in E$ if and only if $(f(v), f(u)) \in E'$. The notation $G \cong G'$ is used to denote that G is isomorphic to G' , and $G \not\cong G'$ to denote that G is not isomorphic to G' . A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. A *clique* $G' = (V', E')$ is a subgraph, $G' \subseteq G$ which is complete (i.e. for all $u, v \in G'$, $(u, v) \in E'$). And the clique number ω of a graph G is the size of the largest clique in G .

3. Basic Properties of the Configuration Graph

In this section, we provide a set of combinatorial results on the configuration graph. We first show that every subgraph, $G_r(v)$, of the configuration graph $G(P) = (V, E)$ with the vertex set $V'(v) = \{u \in V \mid D(v, u) \leq \ell\}$ for $v \in V$, and edge set $E' = (V' \times V') \cap E$ are isomorphic. We build on this by providing a tight bound on the diameter of these graphs. We start by considering some local structures within the graph.

Lemma 1. *Given a Parikh vector P with associated configuration graph $G(P)$, each vertex $v \in V(P)$ belongs to $\sum_{j \in \Sigma} \prod_{i \in \Sigma \setminus \{j\}} P[i]$ maximal cliques, with the size of each such clique being in $\{P[i] + 1 \mid i \in \Sigma\}$.*

PROOF. Consider first the words with Parikh vector $P = (k, 1)$. Note that every word in $\Sigma^{*|P}$ consists of k copies of the symbol 1, and one copy of the symbol 2. Therefore, given any pair of words $w, u \in \Sigma^{*|P}$ such that $w[i] = u[j] = 2$, the 2-swap (i, j) transforms w into u and hence there exists some edge between w and u . Hence $G(P)$ must be a complete graph of size $k + 1$.

In the general case, consider the word $w \in \Sigma^{*|P}$ where $P = (k_1, k_2, \dots, k_\sigma)$, and $n = \sum_{k_i \in P} k_i$. Let $\text{Pos}(w, i) = \{j \in [n] \mid w[j] = i\}$. Given $i, j \in [\sigma], i \neq j$, let $i_1, i_2 \in \text{Pos}(w, i)$ and $j_1 \in \text{Pos}(w, j)$ be a set of indices. Let $v_1 = w \circ (i_1, j_1)$ and $v_2 = w \circ (i_2, j_1)$. Then, $v_1[i_1] = v_2[i_2], v_2[i_2] = v_1[i_1]$, and $v_1[j_1] = v_2[j_1]$. Further, for every $\ell \in [n]$ such that $\ell \notin \{i_1, i_2, j_1\}$, $v_1[\ell] = v_2[\ell]$ as these positions are unchanged by the swaps. Therefore, $v_1 = v_2 \circ (i_1, i_2)$, and hence these words are connected in $G(P)$. Further, as this holds for any $j_1 \in \text{Pos}(w, j)$, the set of words induced by the swaps (j, ℓ) , for some fixed $\ell \in \text{Pos}(w, i)$ correspond to a clique of size $P[j] + 1$. Therefore, there exists $\prod_{i \in \Sigma \setminus \{j\}} P[i]$ cliques of size $P[j] + 1$ including w , for any $j \in \Sigma$.

We now show that the cliques induced by the set of swaps $S(i, j) = \{(i', j) \mid i' \in \text{Pos}(w, w[i])\}$ are maximal. Let $C(i, j, w) = \{w\} \cup \{w \circ (i', j) \mid (i', j) \in S(i, j)\}$, i.e. the clique induced by the set of swaps in $S(i, j)$. Consider a set of swaps, $(i_1, j_1), (i_2, j_1), (i_1, j_2)$ and (i_2, j_2) , where $i_1, i_2 \in \text{Pos}(w, i)$ and $j_1, j_2 \in \text{Pos}(w, j)$. Let $v_{1,1} = w \circ (i_1, j_1), v_{2,1} = w \circ (i_2, j_1), v_{1,2} = w \circ (i_1, j_2)$ and $v_{2,2} = w \circ (i_2, j_2)$. Note that $\{w, v_{1,1}, v_{2,1}\} \subseteq C(i, j_1), \{w, v_{1,1}, v_{1,2}\} \subseteq C(j, i_1), \{w, v_{2,1}, v_{2,2}\} \subseteq C(i, j_2)$ and $\{w, v_{2,1}, v_{2,2}\} \subseteq C(j, i_2)$.

We now claim that there exists no swap transforming $v_{1,1}$ into $v_{2,2}$. Observe first that, for every $\ell \in [|w|]$ such that $\ell \notin \{i_1, i_2, j_1, j_2\}$, $v_{1,1}[\ell] = v_{2,2}[\ell]$. As $v_{1,1}[i_1] = v_{2,2}[j_1]$, and $v_{1,1}[j_1] = v_{2,2}[i_2]$, exactly two swaps are needed to transform $v_{1,1}$ into $v_{2,2}$. Therefore, for any pair of swaps $(i_1, j_1), (i_2, j_2) \in \text{Pos}(i, w) \times \text{Pos}(j, w)$, such that $i_1 \neq i_2$ and $j_1 \neq j_2$, the words $w \circ (i_1, j_1)$ and $w \circ (i_2, j_2)$ are not adjacent in $G(v)$. Similarly, given a set of indices $i' \in \text{Pos}(i, w), j' \in \text{Pos}(j, w)$ and $\ell' \in \text{Pos}(w, w)$ and swaps $(i', j'), (i', \ell')$, observe that as $w[j'] \neq w[\ell']$, the distance between $w \circ (i', j')$ and $w \circ (i', \ell')$ is 2. Therefore, every clique induced by the set of swaps $S(i, j) = \{(i', j) \mid i' \in \text{Pos}(w, w[i])\}$ is maximal.

Corollary 1. *Let $v \in V(P)$ be the vertex in $G(P)$ corresponding to the word $w \in \Sigma^{*|P}$. Then, v belongs only to the maximal cliques corresponding to the set of words $\{w \circ (i, j) \mid i \in \text{Pos}(w, x)\}$ for some fixed symbol $x \in \Sigma$ and position $j \in [|w|], w[j] \neq x$, where $\text{Pos}(w, x) = \{i \in [|w|] \mid w[i] = x\}$.*

Now since the edges in each maximal clique only swap two types of symbols we have the following corollary for the number of cliques.

Corollary 2. *There are $\sum_{(i,j) \in \Sigma \times \Sigma} \frac{(\sum_{k \in \Sigma \setminus \{i,j\}} P[k])!}{\prod_{k \in \Sigma \setminus \{i,j\}} P[k]!}$ maximal cliques in $G(p)$.*

Corollary 3. *The clique number $\omega(G(P))$ is equal to $\max_{i \in \Sigma} P[i] + 1$.*

Lemma 2. *Let $G_r(v)$ be the subgraph of $G(P)$ induced by all vertices of distance at most r away from a given vertex v . Then, for any pair of vertices $u, v \in V$ and given any $r \in \mathbb{Z}^+$, $G_r(u) \cong G_r(v)$.*

PROOF. Let $\pi \in S_n$ be the permutation such that $u \circ \pi = v$. We use the permutation π to define an isomorphism $f : G_r(u) \rightarrow G_r(v)$ such that $f(w) = w \circ \pi$. In order to show that f is an isomorphism we need to show that it preserves adjacency. We start by showing that for every word, $w \in G_1(u)$, $f(w) \in G_1(v)$.

Let $\tau = (\tau_1, \tau_2)$ be the 2-swap such that $w = u \circ \tau$. We now have 3 cases for how π and τ interact, either none of the indices in τ are changed by π , just one of τ_1 or τ_2 are changed by π , or both τ_1 and τ_2 are changed by π . In the first case, $f(w)$ is adjacent to v as $v \circ \tau = f(w)$. In the second case, let (τ_1, τ_2) be a swap that that $\pi[\tau_1] = \tau_1$, i.e. τ_1 is not changed by the permutation π . We define a new swap τ' such that $v \circ \tau' = f(w)$. Let $x, y \in [n]$ be the positions in v such that $\pi[x] = \tau_2$ and $\pi[\tau_2] = y$. Now, let $\tau' = (\tau_1, y)$. Observe that $v[y] = w[\tau_2]$, and $v[\tau_1] = w[\tau_1]$. Therefore, the word $v \circ \tau' = u \circ \tau \circ \pi$. Note that as the ordering of the indices in the swap does not change the swap, the same argument holds for the case when $\pi[\tau_2] = \tau_2$. In the final case, let $\tau' = (\pi[\tau_1], \pi[\tau_2])$. Note that by arguments above, $u[\pi[\tau_1]] = v[\tau_1]$ and $u[\pi[\tau_2]] = v[\tau_2]$, and hence $v \circ \tau' = u \circ \tau \circ \pi$. Repeating this argument for each word at distance $\ell \in [1, r]$ proves this statement.

We now provide the exact value of the diameter of any configuration graph $G(P)$. Theorem 3 states the explicit diameter of the graph, with the remainder of the section dedicated to proving this result.

Theorem 3. *The diameter of the Configuration Graph, $G(P)$ for a given Parikh vector P is $n - \max_{i \in \Sigma} P[i]$.*

Theorem 3 is proven by first showing that the upper bound matches $n - \max_{i \in \Sigma} P[i]$ (Lemma 4). We then show that the lower bound on the diameter matches the upper bound (Lemma 6), concluding our proof of Theorem 3.

Lemma 4 (Upper Bound of Diameter). *The diameter of the Configuration Graph, $G(P)$ for a given Parikh vector P is at most $n - \max_{i \in \Sigma} P[i]$.*

PROOF (PROOF OF UPPER BOUND). This claim is proven by providing a procedure to determine a sequence of $n - \max_{i \in [\sigma]} P[i]$ swaps to transform any word $w \in \Sigma^{*|P}$ into some word $v \in \Sigma^{*|P}$. We assume, without loss of generality, that $P[1] \geq P[2] \geq \dots \geq P[\sigma]$. The procedure described by Algorithm 1 operates by iterating over the set of symbols in Σ , and the set of occurrences of each symbol in the word. At each step, we have a symbol $x \in [2, \sigma]$ and index $k \in [1, P[x]]$. The procedure finds the position i of the k^{th} appearance of symbol x in w , and the position j of the k^{th} appearance of x in v . Formally, i is the value such that $w[i] = x$ and $|\{i' \in [1, i-1] \mid w[i'] = x\}| = k$ and j the value such that $v[j] = x$ and $|\{j' \in [1, j-1] \mid v[j'] = x\}| = k$. Finally, the algorithm adds the swap (i, j) to the set of swaps, and then moves to the next symbol.

Algorithm 1 Procedure to select 2-swaps to generate a path from w to v .

```

1:  $S \leftarrow \emptyset$  ▷ Set of 2-swaps
2: for  $x \in \Sigma \setminus \{1\}$  do
3:   for  $1 \leq k \leq P_x$  do
4:      $i \leftarrow$  index of  $k^{\text{th}}$  occurrence of  $x$  in  $w$ 
5:      $j \leftarrow$  index of  $k^{\text{th}}$  occurrence of  $x$  in  $v$ 
6:      $S \leftarrow S \cup (i, j)$ 
7:   end for
8:   Apply all 2-swaps in  $S$  to  $w$  and set  $S \leftarrow \emptyset$ 
9: end for

```

This procedure requires one swap for each symbol in w other than 1, giving a total of $n - \max_{i \in [\sigma]} P[i]$ swaps. Note that after each swap, the symbol at position j of the word is the symbol $v[j]$. Therefore, after all swaps have been applied, the symbol at position $j \in \{i \in [1, |w|] \mid v[i] \in \Sigma \setminus \{1\}\}$ must equal $v[j]$. By extension, for any index i such that $v[i] = 1$, the symbol at position i must be 1, and thus equal $v[i]$. Therefore this procedure transforms w into v .

In order to prove the lower bound on the diameter (i.e. that $\text{diam}(G) \geq n - \max_{i \in [\sigma]} P[i]$) we introduce a new auxiliary structure, the *2-swap graph*. Informally, the 2-swap graph, defined for a pair of words $w, v \in \Sigma^{*|P}$ and denoted $G(w, v) = \{V(w, v), E(w, v)\}$ is a directed graph such that the edge

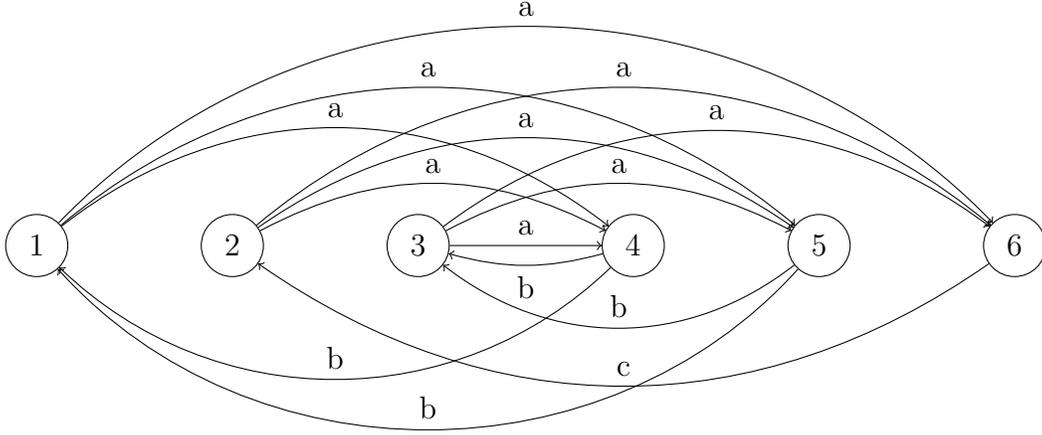


Figure 2: The graph $G(aaabbc, bcbaaa)$ with the edge (i, j) labelled by the symbol $w[i](=v[j])$.

$(u_i, u_j) \in V(w, v) \times V(w, v)$ if and only if $w[i] = v[j]$. Note that this definition allows for self-loops.

Definition 3 (2-swap Graph). Let $w, v \in \Sigma^{*|P}$ be a pair of words. The 2-swap graph $G(w, v) = \{V(w, v), E(w, v)\}$ contains the vertex set $V(w, v) = \{u_1, u_2, \dots, u_{|w|}\}$ and edge set $E(w, v) = \{(v_i, v_j) \in V(w, v) \times V(w, v) \mid w[i] = v[j] \text{ or } v[i] = w[j]\}$. The edge set, E , is defined as follows, for all $i, j \in \Sigma$ there exists an edge $(i, j) \in E$ if and only if $w[i] = v[j]$.

An example of the 2-swap graph is given in Figure 2.

Lemma 5. Let $G(w, v)$ be a graph constructed as above for transforming w into v using 2-swaps. Then, there exists a procedure to convert any cycle cover of $G(w, v)$, \mathcal{C} , into a $w - v$ path in $G(P)$

PROOF. Let $C \in \mathcal{C}$ be a cycle where $C = (e^1, e^2, \dots, e^{|C|})$ and $e_2^i = e_1^{i+1 \bmod |C|}$. The $w - v$ path (i.e. a sequence of 2-swaps) is constructed as follows. Starting with $i = 1$ in increasing value of $i \in [|C| - 1]$, the 2-swap (e_1^i, e_2^i) is added to the set of 2-swaps S . Where e_1^i and e_2^i are the endpoints of edge e^i for each i .

Assume, for the sake of contradiction, that S does not correspond to a proper set of 2-swaps converting w into v . Then, there must exist some symbol at position i such that the symbol $w[i]$ is placed at some position j such

that $w[i] \neq v[j]$. As w_i must be placed at some position that is connected to node i by an edge, there must be an edge between i and j , hence $w_i = v_j$, contradicting the construction of $G(w, v)$. Therefore, S must correspond to a proper set of 2-swaps.

Corollary 4. *Let \mathcal{C} be a cycle cover of $G(w, v)$. Then there exists a set of $\sum_{c \in \mathcal{C}} |c| - 1$ 2-swaps transforming w into v .*

Corollary 5. *Let S be the smallest set of 2-swaps transforming w into v , then S must correspond to a vertex disjoint cycle cover of $G(w, v)$.*

PROOF. For the sake of contradiction, let S be the smallest set of 2-swaps transforming w into v , corresponding to the cycle cover \mathcal{C} where \mathcal{C} is not vertex disjoint. Let $c_1, c_2 \in \mathcal{C}$ be a pair of cycles sharing some vertex u . Then, following the construction above, the symbol w_u must be used in two separate positions in v , contradicting the assumption that v can be constructed from w using 2-swaps. Hence S must correspond to a vertex disjoint cycle cover.

Corollary 6. *Given a pair of words $w, v \in \Sigma^{*|P}$, the minimum set of 2-swaps transforming w into v S corresponds to the vertex disjoint cycle cover of $G(w, v)$ maximising the number of cycles.*

Lemma 6 (Lower Bound). *The diameter of $G(p)$ is at least $n - \max_{i \in \Sigma} P[i]$.*

PROOF. We assume w.l.o.g. that $P[1] \geq P[2] \geq \dots \geq P[\sigma]$. Let $w, v \in \Sigma^{*|P}$ satisfy:

$$w = (123 \dots \sigma)^{P[\sigma]} (123 \dots \sigma - 1)^{P[\sigma-1]-P[\sigma]} \dots 1^{P[1]-P[2]}$$

and

$$= (23 \dots \sigma 1)^{P[\sigma]} (23 \dots (\sigma - 1) 1)^{P[\sigma-1]-P[\sigma]} \dots 1^{P[1]-P[2]},$$

i.e. w is made up of $P[1]$ subwords, each of which are of the form $12\dots k$, and v is made up of the same subwords as w but each of them has been cyclically shifted by one (for example when $P = (3, 2, 1)$ we have $w = 123121$ and $v = 231211$). Following Corollary 6, the minimum number of 2-swaps needed to convert w into v can be derived from a vertex disjoint cycle cover of $G(w, v)$ with the maximum number of cycles.

Observe that any occurrence of symbol σ must have an outgoing edge in $G(w, v)$ to symbol 1, and an incoming edge from symbol $\sigma - 1$. Repeating

this logic, each instance of σ must be contained within a cycle of length σ . Removing each such cycles and repeating this argument gives a set of $P[1]$ cycles, with $P[\sigma]$ cycles of length σ , $P[\sigma - 1] - P[\sigma]$ cycles of length $\sigma - 1$, and generally $P[i] - P[i + 1]$ cycles of length i . This gives the number of 2-swaps needed to transform w to v being a minimum of $n - P[1] = n - \max_{i \in \Sigma} P[i]$.

Theorem 3 follows from Lemmas 4 and 6.

4. Hamiltonicity

In this section, we prove that the configuration graph contains Hamiltonian paths and we provide an efficient algorithm for enumerating the vertices of a Hamiltonian path. We first show that every configuration graph of a Parikh Vector over a binary alphabet is Hamiltonian. This is then generalised to alphabets of size σ , using the binary case to build Hamiltonian paths with alphabets of size σ .

4.1. Binary Alphabets

For notational conciseness, given a symbol a , in a binary alphabet Σ , the notation \bar{a} is used to denote $\bar{a} \in \Sigma, a \neq \bar{a}$, i.e. if $a = 1$, then $\bar{a} = 2$. We prove Hamiltonicity via a recursive approach that forms the basis for our enumeration algorithm. Our proof works by taking an arbitrary word in the graph w , and constructing a path starting with w . At each step of the path, the idea is to find the shortest suffix of w such that both symbols in Σ appear in the suffix. Letting $w = ps$, the path is constructed by first forming a path containing every word ps' , for every $s' \in \Sigma^{*|P(s)}$, i.e. a path from w transitioning through every word formed by maintaining the prefix p and permuting the suffix s . Once every such word has been added to the path, the algorithm repeats this process by performing some swap of the form $(|p|, i)$ where $i \in [|p| + 1, |w|]$, i.e. a swap taking the last symbol in the prefix p , and replacing it with the symbol $\bar{w}[|p|]$ from some position in the suffix.

This process is repeated, considering increasingly long suffixes, until every word has been covered by the path. Using this approach, we ensure that every word with the same prefix is added to the path first, before shortening the prefix. The algorithm HAMILTONENUMERATION outlines this logic within the context of the enumeration problem, where each transition is output while constructing the path.

Theorem 7. *For every Parikh vector $P \in \mathbb{N}_0^2$ and word $w \in \Sigma^{*|P}$, there exists a Hamiltonian path starting at w in the configuration graph $G(P) = (V(P), E(P))$.*

As stated in [32] the binary reflected Gray code gives an ordering for the words over a Binary alphabet restricted to a certain Parikh Vector $(k, n - k)$ with a single 2-swap between each subsequent word. This does indeed prove Theorem 7 by providing a Hamiltonian Circuit for a given binary Parikh vector (it is worth noting that following a Hamiltonian Circuit starting at w gives a Hamiltonian path from w .) However, we also present our own inductive proof for this case to provide an explanation of our enumeration algorithm.

PROOF. We prove this statement in a recursive manner. As a base case, consider the three vectors of length 2 as our Parikh Vector, namely $(2, 0)$, $(0, 2)$ and $(1, 1)$. Note that there exists only a single word with the Parikh vectors $(2, 0)$ or $(0, 2)$, and thus the graph must, trivially, be Hamiltonian. For the Parikh vector $(1, 1)$, there exists only the words 12 and 21, connected by the 2-swap $(1, 2)$ and therefore is also a Hamiltonian path and it can be found starting at either word.

In the general case, assume that for every Parikh vector $P' = (P'_1, P'_2)$ with $P'_1 + P'_2 < \ell$, the graph $G(P')$ contains a Hamiltonian path, and further there exists such a path starting at every word in $\Sigma^{*|P'}$. Now, let $P = (P_1, P_2)$ be an arbitrary Parikh vector such that $P_1 + P_2 = \ell$. Given some word $w \in \Sigma^{*|P}$, observe that there must exist some Hamiltonian path starting at the word $w[2, \ell]$ in the subgraph $G'(P) = (V'(P), E'(P))$ where $V'(P) = \{u \in V(P) \mid u[1] = w[1]\}$ and $E'(P) = (V'(P) \times V'(P)) \cap E(P)$. Let w' be the last word visited by the Hamiltonian path in $G'(P)$, and let i be some position in w' such that $w'[i] = \overline{w[1]}$. Note that there must exist Hamiltonian path starting at $(w \circ (1, i))[2, \ell]$ in the subgraph $G''(P) = (V''(P), E''(P))$ where $V''(P) = \{u \in V(P) \mid u[1] = \overline{w[1]}\}$ and $E''(P) = (V''(P) \times V''(P)) \cap E(P)$. As every vertex in $G(P)$ is either in the subgraph $G'(P)$ or $G''(P)$, the Hamiltonian paths starting at w in $G'(P)$ and at $w' \circ (1, i)$ in $G''(P)$ cover the complete graph. Further, as these paths are connected, there exists a Hamiltonian path starting at the arbitrary word $w \in \Sigma^{*|P}$, and therefore the Theorem holds.

Enumeration. We now provide our enumeration algorithm. Rather than output each word completely, we instead maintain the current state of the

word in memory and output the swaps taken at each step, corresponding to the edges traversed in the path. This way, at any given step the algorithm may be paused and the current word fully output, while the full path can be reconstructed from only the output. There are two key challenges behind this algorithm. First is the problem of deciding the next swap to be taken to move from the current word in the graph to the next word. Second, is the problem of minimising the worst-case delay in the output of these swaps, keeping in mind that the output is of constant size.

High-Level Idea.

From a given word w with Parikh vector, P , the algorithm works by first finding the shortest suffix s of w such that there exists some pair of indices i, j for which $s[i] \neq s[j]$. Using this suffix and letting $w = us$, we find a path through every vertex in $G(P)$ with the prefix u . Note that following the same arguments as Theorem 7, such a path must exist. Once every word in $G(P)$ with the prefix u has been visited by the path, the algorithm then enumerates every word with the prefix $u[1, |u| - 1]$, extending the current path. When adding every word with the prefix $u[1, |u| - 1]$ to the path, note that every word with the prefix u has already been added, thus all that is left is to add those words with the prefix $u[1, |u| - 1]\overline{u[|u|]}$, which is achieved via the same process as before.

The swaps are determined as follows. From the initial word w , let R_1 be the last occurrence of the symbol 1 in w , and let R_2 be the last occurrence of 2 in w . The first swap is made between $\min(R_1, R_2)$ and $\min(R_1, R_2) + 1$, with the algorithm then iterating through every word with the Parikh vector $P[w[\min(R_1, R_2), |w|]] - P[\overline{w[\min(R_1, R_2)]}]$.

In the general case, a call is made to the algorithm with a Parikh vector $P = (P_1, P_2)$, with the current word w fixed, and the assumption that no word with the prefix $w[1, |w| - (P_1 + P_2)]$ has been added to the path other than w . The algorithm, therefore, is tasked with iterating through every word with the current prefix. Let R_1 be the last occurrence of the symbol 1, and R_2 be the last occurrence of the symbol 2 in the current word. The algorithm first enumerates every word with the prefix $w[1, \min(R_1, R_2) - 1]$. Noting that there exists only a single word with the prefix $w[1, \min(R_1, R_2)]$, it is sufficient to only enumerate through those words with the prefix $w[1, \min(R_1, R_2) - 1]\overline{w[\min(R_1, R_2)]}$. The first swap made by this algorithm is $(\min(R_1, R_2), \min(R_1, R_2) + 1)$, allowing a single recursive call to be made to

Algorithm 2 Algorithm for enumerating a Hamiltonian path in the configuration graph defined by a Parikh vector P . Note that the pointer $last_state$ is used to return at the state in the stack at position $last_state$ to avoid needless recursion. The function $CurrentState()$ is used to get the current state in the stack. The trees T_1 and T_2 are balanced binary search trees such that every node in T_1 corresponds to a position of symbol 1 in the current state of w , and every node in T_2 corresponds to a position of the symbol 2 in w .

```

1: Global Variables:
2: Word  $w \in \Sigma^n$ 
3: Balanced Binary Search Tree  $T_1$ 
4: Balanced Binary Search Tree  $T_2$ 
5: function HAMILTONIANENUMERATION( $(P_1, P_2) \in \mathbb{N}_0 \times \mathbb{N}_0$ , pointer
    $last\_state$ )
6:   if  $P_1 = 0$  or  $P_2 = 0$  then Return To  $last\_state$ 
7:   else if  $(P_1, P_2) = (1, 1)$  then
8:     Output:  $(n - 1, n)$ 
9:     Remove $(T_{w[n-1]}, n - 1)$ , Insert $(T_{w[n-1]}, n)$ 
10:    Remove $(T_{w[n]}, n)$ , Insert $(T_{w[n]}, n - 1)$ 
11:     $w \leftarrow w \circ (n - 1, n)$ 
12:    ReturnTo  $last\_state$ 
13:   else % Note that  $P_1 + P_2 \geq 3$ 
14:      $R_1 \leftarrow \max(T_1)$ 
15:      $R_2 \leftarrow \max(T_2)$ 
16:     for  $i \in \min(R_1, R_2), \min(R_1, R_2) - 1, \dots, n - (P_1 + P_2 - 1)$  do
17:        $j \leftarrow \min_{j' \in [i+1, n]} \overline{T_{w[j'()]}}$ 
18:       Output:  $(i, j)$ 
19:       Remove $(T_{w[i]}, i)$ , Insert $(T_{w[i]}, j)$ 
20:       Remove $(T_{w[j]}, j)$ , Insert $(T_{w[j]}, i)$ 
21:        $w \leftarrow w \circ (i, j)$ 
22:        $P' \leftarrow (P_1, P_2) - P(w[i])$ 
23:       HamiltonianEnumeration( $P', CurrentState()$ )
24:     end for
25:     % As the Parikh vector must have at least one value for each
   symbol, there is a valid swap from  $n - (P_1 + P_2 - 1)$  to  $i$ , for some
    $i > n - (P_1 + P_2 - 1)$ 
26:      $j \leftarrow \min_{i \in [n - (P_1 + P_2 - 1), n]} \overline{T_{w[i]}}$ 
27:     Output  $(n - (P_1 + P_2 - 1), j)$ 
28:     Remove $(T_{w[n - (P_1 + P_2 - 1)]}, n - (P_1 + P_2 - 1))$ , In-
   sert $(T_{w[n - (P_1 + P_2 - 1)]}, j)$  15
29:     Remove $(T_{w[j]}, j)$ , Insert $(T_{w[j]}, n - (P_1 + P_2 - 1))$ 
30:      $w \leftarrow w \circ (n - (P_1 + P_2 - 1), j)$ 
31:      $P' \leftarrow (P_1, P_2) - P(w[n - (P_1 + P_2 - 1)])$ 
32:     HamiltonianEnumeration( $P', last\_state$ ) % Note that this skips
   over the current state when returning
33:   end if

```

HamiltonianEnumeration($P(w \circ (\min(R_1, R_2), \min(R_1, R_2) + 1))[\min(R_1, R_2) + 1, |w|]$, CURRENT CALL), where CURRENT CALL denotes the pointer to the current call on the stack. From this call the algorithm enumerates every word with the prefix $w[1, \min(R_1, R_2) - 1] \overline{w[\min(R_1, R_2)]}$. As every word with the prefix $w[1, \min(R_1, R_2)]$ has already been output and added to the path, once this recursive call has been made, every word with the prefix $w[1, \min(R_1, R_2) - 1]$ will have been added to the path. Note that the word w is updated at each step, ending at the word w' .

After every word with the prefix $w'[1, \min(R_1, R_2) - 1]$ has been added to the path, the next step is to add every word with the prefix $w'[1, \min(R_1, R_2) - 2]$ to the path. As every word with the prefix $w[1, \min(R_1, R_2) - 1]$ is already in the path, it is sufficient to add just those words with the prefix $w'[1, \min(R_1, R_2) - 2] \overline{w[\min(R_1, R_2)]}$ to the path. This is achieved by making the swap between $\min(R_1, R_2) - 2$, and the smallest value $i > \min(R_1, R_2) - 2$ such that $w[i] \neq w'[\min(R_1, R_2) - 1]$, then recursively enumerating every word with the prefix $w'[1, \min(R_1, R_2) - 2]$. This process is repeated in decreasing prefix length until every word has been enumerated.

To efficiently determine the last position in the current word w containing the symbols 1 and 2, a pair of balanced binary search trees are maintained. The tree T_1 corresponds to the positions of the symbol 1 in w , with each node in T_1 being labelled with an index and the tree sorted by the value of the labels. Analogously, tree T_2 corresponds to the positions of the symbol 2 in w . Using these trees, note that the last position in w at which either symbol appears can be determined in $O(\log n)$ time, and further each tree can be updated in $O(\log n)$ time after each swap.

Lemma 8. *Let P be a Parikh vector of length n , and let $w \in \Sigma^{*|P}$ be a word. HAMILTONIANENUMERATION outputs a path visiting every word in $\Sigma^{*|P}$ starting at w .*

PROOF. This lemma is proven via the same tools as Theorem 7. Explicitly, we show first that the algorithm explores every suffix in increasing length, relying on the exploration of suffixes of length 2 as a base case, then provide an inductive proof of the remaining cases. We assume that the starting word has been fully output as part of the precomputation. With this in mind, note that there are two cases for length 2 prefixes, either the suffix contains two copies of the same symbol or one copy of each symbol. In the first case, as w has been output, so has every permutation of the length 2 prefix of

w . Otherwise, the algorithm outputs the swap $(n - 1, n)$ and returns to the previous call.

In the general case, we assume that for some $\ell \in [n]$, every permutation of $w[n - \ell + 1, n]$ has been visited by the path. Further, we assume the algorithm can, given any word v , visit every word of the form $v[1, n - \ell]u$, for every $u \in \Sigma^{*|P(v[n - \ell + 1, n])}$, i.e. the algorithm is capable of taking any word v as an input, and visiting every word with the same Parikh vector $P(v)$ and prefix $v[1, n - \ell + 1]$. Note that in the case that $w[n - \ell, n] = w[n - \ell]^\ell$, the algorithm has already visited every word in $\Sigma^{*|P(w)}$ with the prefix $w[1, n - \ell]$. Otherwise, as the algorithm has, by this point, visited every word of the form $w[1, n - \ell + 1]u$, for every $u \in \Sigma^{*|P(v[n - \ell - 1, n])}$, it is sufficient to show that the algorithm visits every word of the form $w[1, \ell - 1]\overline{w[\ell]}u$, for every $u \in \Sigma^{*|P'}$, $P' = P(w[n - \ell, n]) - P(\overline{w[\ell]})$.

Let w' be the last word visited by the algorithm with the prefix $w[1, n - \ell + 1]$. Note that the first step taken by the algorithm is to determine the first position j in $w'[n - \ell + 1, n]$ containing the symbol $\overline{w[\ell]}$. Therefore, by making the swap $(n - \ell, j)$, the algorithm moves to some word with a suffix in $\Sigma^{P'}$, where $P' = P(w[n - \ell, n]) - P(\overline{w[n - \ell]})$. As the algorithm can, by inductive assumption, visit every word with a suffix of length $\ell - 1$, the algorithm must also be able to visit every word with a suffix of length ℓ , completing the proof.

Lemma 9. *Let P be a Parikh vector, and let $w \in \Sigma^{*|P}$ be a word. The path output by HAMILTONENUMERATION does not visit any word in $w \in \Sigma^{*|P}$ more than once.*

PROOF. Note that this property holds for length 2 words. By extension, the length at most 2 path visiting every word with the prefix $w[1, n - 2]$ does not visit the same word twice before returning to a previous call on the stack.

Assume now that, given any input word $v \in \Sigma^{*|P}$, the algorithm visits every word in $\Sigma^{*|P}$ with the prefix $v[1, n - l + 1]$ without repetition, and has only visited words with this prefix. Further, assume that $P(v[n - \ell, n]) \neq (0, \ell - 1)$ or $(\ell - 1, 0)$. Then, after every such word has been visited by the path, the algorithm returns to the previous state, with the goal of enumerating every word with the prefix $v[1, n - \ell]$. As every word in $\Sigma^{*|P}$ with the prefix $v[1, n - \ell + 1]$ has been visited, it is sufficient to show that only those words with the prefix $v[1, n - \ell]\overline{v[n - \ell + 1]}$ are enumerated. The first swap made at this state is between ℓ and the smallest index $j \in [n - \ell + 1, n]$ such that $v[n - \ell] \neq v[j]$, which, as the algorithm has only visited words with the prefix

$v[1, n - l + 1]$, has not previously been visited. After this swap, the algorithm enumerates every word with the prefix $v[1, n - l - 1]v[n - \ell]$, which, by the inductive assumption, is done without visiting the same word. Therefore, by induction, every word with the prefix $v[1, n - \ell]$ is visited by the path output by HAMILTONIANENUMERATION exactly once.

Theorem 10. *Given a Parikh vector $P = (P_1, P_2)$ such that $P_1 + P_2 = n$, and word $w \in \Sigma^{*|P}$, HAMILTONIANENUMERATION outputs a Hamiltonian path with at most $O(\log n)$ delay between the output of each edge after $O(n \log n)$ preprocessing.*

PROOF. Following Lemmas 8 and 9, the path outputted by HAMILTONENUMERATION is Hamiltonian. In the preprocessing step, the algorithm constructs two balanced binary search trees T_1 and T_2 . Every node in T_1 is labelled by some index $i_1 \in [n]$ for which $w[i_1] = 1$, and sorted by the values of the labels. Similarly, every node in T_2 is labelled by some index $i_2 \in [n]$ for which $w[i_2] = 2$, and sorted by the values of the labels. As each of these constructions requires at most $O(n \log n)$ time, the total complexity of the preprocessing is $O(n \log n)$.

During each call, we have one of three cases. If either value of the Parikh vector is 0, then the algorithm immediately returns to the last state without any output. If the Parikh vector is $(1, 1)$, then the algorithm outputs a swap between the two symbols, updates the trees T_1 and T_2 , requiring at most $O(\log n)$ time, then returns to the last state. In the third case, the Parikh vector (P_1, P_2) satisfies $P_1 > 0, P_2 > 0$. First, the algorithm determines the last position in the current state of the word w containing the symbol 1 and the last position containing the symbol 2, i.e. the values $R_1 = \max_{j \in [1, n]} w[i_1] = 1$ and $R_2 = \max_{j \in [1, n]} w[i_2] = 2$. These values can be determined in $O(\log n)$ time using the trees T_1 and T_2 . Using these values, the algorithm iterates through every length from $\min(T_1, T_2)$ to $n - (P_1 + P_2 - 1)$, enumerating every word in $\Sigma^{*|P(w)}$ with the prefix $w[1, n - (P_1 + P_2 - 1)]$. For each $\ell \in [\min(T_1, T_2), n - (P_1 + P_2 - 1)]$, the algorithm outputs the swap (ℓ, j) , where $j \in [n - \ell, n]$ is the largest value for which $w[j] = w[\ell]$. After this has been output, the algorithm updates the trees T_1 and T_2 . Note that both finding the value of j and updating the trees require $O(\log n)$ time. After this swap, the algorithm makes the next call to HAMILTONIANENUMERATION. Note that after this call, HAMILTONIANENUMERATION must either return immediately to the last state or output some swap before either returning

or making the next recursive call. Therefore, ignoring the time complexity of returning to a previous state in the stack, the worst case delay between outputs is $O(\log n)$, corresponding to searching and updating the trees T_1 and T_2 .

To avoid having to check each state in the stack after returning from a recursive call, the algorithm uses tail recursion. Explicitly, rather than returning to the state in the stack from which the algorithm was called, the algorithm is passed a pointer to the last state in the stack corresponding to a length ℓ such that some word with the prefix $w[1, n - \ell]$ has not been output. To do so, after the swap between $n - (P_1 + P_2 - 1)$ and j is made, for the value j as defined above, the algorithm passes the pointer it was initially given, denoted in the algorithm as *last_state* to the call to HAMILTONIANENUMERATION, allowing the algorithm to skip over the current state during the recursion process.

4.2. General Alphabets.

We now show that the graph is Hamiltonian for any alphabet of size $\sigma \geq 2$. The main idea here is to build a cycle based on recursively grouping together sets of symbols. Given a Parikh vector $P = (P_1, P_2, \dots, P_\sigma)$, our proof operates in a set of $\sigma - 1$ recursive phases, with the i^{th} step corresponding to finding a Hamiltonian path in the graph $G(P_i, P_{i+1}, \dots, P_\sigma)$, then mapping this path to one in $G(P)$. The paths in $G(P_i, P_{i+1}, \dots, P_\sigma)$ are generated in turn by a recursive process. Starting with the word w , first, we consider the path visiting every vertex corresponding to a permutation of the symbols $i + 1, \dots, \sigma$ in w . Explicitly, every word v in this path is of the form:

$$v[i] = \begin{cases} w[i] & w[i] \in \{1, 2, \dots, i\} \\ x_i \in \{i + 1, \dots, \sigma\} & w[i] \notin \{1, 2, \dots, i\} \end{cases},$$

where x_i is some arbitrary symbol $\{i, i + 1, \dots, \sigma\}$. Further, every such word is visited exactly once.

After this path is output, a single swap corresponding to the first swap in $G(P_i, (P_{i+1} + P_{i+2}, \dots, P_\sigma))$ is made, ensuring that this swap must involve some position in w containing the symbol i . After this swap, another path visiting exactly once every word corresponding to a permutation of the symbols $i + 1, \dots, \sigma$ in w can be output. By repeating this for every swap in $G(P_i, (P_{i+1} + P_{i+2}, \dots, P_\sigma))$, inserting a path visiting exactly once every word

corresponding to a permutation of the symbols $i+1, \dots, \sigma$ in w between each such swap, note that every permutation of the symbols $i, i+1, \dots, \sigma$ in w is output exactly once. In other words, every word $v \in \Sigma^{*|P}$ of the form

$$v[i] = \begin{cases} w[i] & w[i] \in \{1, 2, \dots, i-1\} \\ x_i \in \{i, i+1, \dots, \sigma\} & w[i] \in \{i, i+1, \dots, \sigma\} \end{cases},$$

where x_i is some symbol in $\{i, i+1, \dots, \sigma\}$. Further, each such word is visited exactly once. Using the binary alphabet as a base case, this process provides an outline of the proof of the Hamiltonicity of $G(P)$.

Theorem 11. *Given an arbitrary Parikh vector $P \in \mathbb{N}^\sigma$, there exists a Hamiltonian path starting at every vertex v in the configuration graph $G(P)$.*

PROOF. This is formally proven using the outline above via an inductive argument with the base case of binary alphabets, the Hamiltonicity of which is proven in Theorem 7.

We assume now that there exists, for any Parikh vector $p \in \mathbb{N}_0^{\ell-1}$ and word $w \in \Sigma^{*|P}$, there exists some Hamiltonian path in $G(P)$ starting at w . Let $q = (q_1, q_2, \dots, q_\ell)$ be a Parikh vector, and let $v \in \Sigma^{*|q}$ be an arbitrary word with the Parikh vector q . To construct the Hamiltonian path starting at v in $G(q)$, we first form a Hamiltonian path P_1 in $G(q_2, q_3, \dots, q_\ell)$ starting at the word v' formed by deleting every symbol 1 from v . We assume that we have a table T such that $T[i]$ returns the index in $[1, |v|]$ for which $v'[i] = v[T[i]]$. To avoid any repetition, we require $T[1] < T[2] < \dots < T[|v'|]$. With this table, each swap (s_1, s_2) in the path P_1 can be converted to the swap $(T[s_1], T[s_2])$ in the graph $G(q)$ swapping the same symbols in v as in the reduced word v' . With this conversion, P_1 constructs a path in $G(q)$ visiting exactly once each word where the symbol 1 appears only at the position $\{i \in [1, |v|] \mid v[i] = 1\}$.

Next, we construct a Hamiltonian path P_2 in the graph $G(q_1, q_2 + q_3 + \dots + q_\ell)$ starting at the word v' where $v'[i] = \begin{cases} 1 & v[i] = 1 \\ x & v[i] \neq 1 \end{cases}$, for some new symbol

x . This graph can be seen as an abstraction of $G(q)$, considering only swaps between some position labelled 1 and any position with a different symbol.

The first swap in P_2 is applied to the current word, however, rather than proceeding along this path, a new set of swaps is inserted corresponding to some Hamiltonian path in $G(q_2, q_3, \dots, q_\ell)$ generated in the same manner as

before. Again, this new path corresponds to a permutation of every symbol in the set $\{2, 3, \dots, \sigma\}$, while fixing the positions of the symbol 1 in the word. This is repeated by taking a single swap from the path P_2 , followed by a complete path corresponding to a Hamiltonian path in $G(q_2, q_3, \dots, q_\ell)$. By combining these paths, the new path must visit exactly once every word in $\Sigma^{*|q_2, q_3, \dots, q_\ell}$ where the positions of symbol 1 are fixed, each time a word w with a new permutation of the symbol 1 is visited. Similarly, every word in $\Sigma^{*|q_1, q_2 + q_3 + \dots + q_\sigma}$ is visited exactly once, corresponding to a path through every permutation of the positions of the symbols 1 in the word. Therefore, the output path is Hamiltonian.

Theorem 12. *Given a Parikh vector $P = (P_1, P_2, \dots, P_\sigma)$ such that $\sum_{i \in [1, \sigma]} P_i = n$, there exists an algorithm outputting a Hamiltonian path in $G(P)$ with a delay of at most $O(\sigma \log n)$ between outputting each edge after $O(n \log n)$ preprocessing.*

PROOF. See Algorithm 3 for the full pseudocode of this algorithm.

Our algorithmic results works using the same approach as for the binary case, outlined in Theorem 10. The primary difference between these algorithms is the use of a recursive process to enumerate the swaps between all positions containing some symbol in $(2, 3, \dots, \sigma)$, using the same algorithm to enumerate the Hamiltonian path starting with the word formed by removing every word containing the symbol 1. In the q^{th} -step of the recursion, after a swap is made between some position containing the symbol q , and some position containing some symbol $x \in [q + 1, \sigma]$, a recursive call is made to enumerate the Hamiltonian path starting at the word formed by removing the every copy of the symbol q .

Our algorithm works as follows. We assume, at each step, we have the current word w and σ balanced binary search trees, $T_1, T_2, \dots, T_\sigma$ where T_q stores all positions of the symbol q in w . When making a call to the function, GENERALHAMILTONIANENUMERATION, three parameters are passed, the Parikh vector the the current prefix being considered, $(P_1, P_2, \dots, P_\sigma)$, the smallest symbol q on which swaps are allowed, and a pointer to the return state once this process is exhausted. The swaps are determined in the same way as Algorithm 2, with two key differences. Within each call, we only make swaps between positions in the word w containing the symbol q , and positions containing any symbol in $[q + 1, \sigma]$. The swap is chosen in the same way as in the binary case, swapping the positions R_q , the **R**ightmost position containing q ,

and R_p , the **R**ightmost position containing any symbol greater than q . We note that, in order to determine the value of R_p we determine the rightmost position of each symbol $x \in [q + 1, \sigma]$, the value of which is determined using the trees $T_{q+1}, T_{q+2}, \dots, T_\sigma$, requiring in total $\sigma \log n$ time. Alongside this, we store a set of positions $p_q, p_{q+1}, \dots, p_\sigma$, where p_x stores the rightmost position smaller than $\min(R_q, R_p)$ containing the symbol x , i.e. the value such that $p_x < \min(R_q, R_p)$ and, for every p'_x where $w[p'_x] = x$, either $p'_x \geq \min(R_q, R_p)$ or $p'_x < p_x$. We use these to allow efficient update of the value i denoting the largest position of the next swap. Once a swap has been made, two recursive calls are made. First, to `GENERALHAMILTONIANENUMERATION` with the arguments $P(w), q + 1, \text{current_state}()$, enumerating every permutation of the positions in w containing some symbol $x \in [q + 1, \sigma]$, before returning to this call, then to `GENERALHAMILTONIANENUMERATION` with the arguments $(P_1, P_2, \dots, P_\sigma - P(w[i]), q, \text{current_state}())$, enumerating every suffix of w of length $P_1 + P_2 + \dots + P_\sigma - 1$. Finally, once every permutation of $w[n - P_1 - P_2 - \dots - P_\sigma + 1, n]$ formed by swapping two positions containing distinct symbols from the set $[q, \sigma]$ has been enumerated, one last swap is made between the position $n - P_1 - P_2 - \dots - P_\sigma$ and the symbol at position j where j is the leftmost (smallest) position in $[n - (P_1 + P_2 + \dots + P_\sigma), n]$ that does not contain $w[n - (P_1 + P_2 + \dots + P_\sigma)]$, i.e. the value j such that $w[j] \neq w[n - (P_1 + P_2 + \dots + P_\sigma)]$ and, $\forall j' \in [n - (P_1 + P_2 + \dots + P_\sigma), j - 1]$, $w[j'] = w[n - (P_1 + P_2 + \dots + P_\sigma)]$. Once this swap has been made, the algorithm returns to the state it was passed when called, avoiding unnecessary recursion.

Observe that the correctness of this algorithm follows from Theorems 10 (laying out the correctness of our choice of swaps to make at each step) and 11. Further, the worst case delay of $O(\sigma \log n)$ is due to determining the values of R_p at the start of the function, and j during both the main loop enumerating the permutations of the suffix, and in the final output. As each value is computed at most twice between outputs, we arrive at the stated worst case delay.

5. Conclusion

Following the work on 2-swap, the most natural step is to consider these problems for k -swap based on two variants with exactly k and less or equal to k . Note that a configuration graph for exactly k -swap permutation might not have a single component. We also would like to point to other attractive

directions of permutations on multidimensional words [19] and important combinatorial objects such as necklaces and bracelets [21, 22]. For the 2-swap graph specifically, we leave open the problem of determining the shortest path between two given words w and v . We conjecture that the simple greedy algorithm used to derive the upper bound in Lemma 4 can be used to find the shortest path between any pair of vertices.

6. Acknowledgements

We wish to thank Torsten Mütze for providing us with vital assistance concerning the literature of combinatorial Gray codes. We would also like to thank the Leverhulme Trust for funding through the Leverhulme Research Centre for Functional Materials Design.

References

- [1] M. Crochemore, W. Rytter, *Jewels of stringology*, World Scientific, 2003.
- [2] M. Ganczorz, P. Gawrychowski, A. Jez, T. Kociumaka, Edit distance with block operations, in: *ESA, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*, 2018.
- [3] H. Maji, T. Izumi, Listing center strings under the edit distance metric, *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9486 (15) (2015) 771–782.
- [4] A. Amir, H. Paryenty, L. Roditty, On the hardness of the consensus string problem, *Information Processing Letters* 113 (10) (2013) 371–374.
- [5] C. Collins, M. Dyer, M. Pitcher, G. Whitehead, M. Zanella, P. Mandal, J. Claridge, G. Darling, M. Rosseinsky, Accelerated discovery of two crystal structure types in a complex inorganic phase field, *Nature* 546 (7657) (2017) 280–284.
- [6] A. Levy, Exploiting Pseudo-locality of Interchange Distance, in: *SPIRE 2021*, Springer, 2021, pp. 227–240.
- [7] O. Angel, A. Holroyd, D. Romik, B. Virág, Random sorting networks, *Advances in Mathematics* 215 (2) (2007) 839–868.

- [8] D. Kornhauser, G. L. Miller, P. G. Spirakis, Coordinating pebble motion on graphs, the diameter of permutation groups, and applications, in: FOCS 1984, IEEE Computer Society, 1984, pp. 241–250.
- [9] E. Konstantinova, Some problems on cayley graphs, *Linear Algebra and its Applications* 429 (11) (2008) 2754–2769, special Issue devoted to selected papers presented at the first IPM Conference on Algebraic Graph Theory.
- [10] W. Goddard, M. E. Raines, P. J. Slater, Distance and connectivity measures in permutation graphs, *Discrete Mathematics* 271 (1) (2003) 61–70.
- [11] L. Babai, A. Seress, On the diameter of permutation groups, *European journal of combinatorics* 13 (4) (1992) 231–243.
- [12] H. A. Helfgott, Á. Seress, On the diameter of permutation groups, *Annals of mathematics* (2014) 611–658.
- [13] H. A. Helfgott, Á. Seress, A. Zuk, Random generators of the symmetric group: Diameter, mixing time and spectral gap, *Journal of Algebra* 421 (2015) 349–368.
- [14] M. R. Jerrum, The complexity of finding minimum-length generator sequences, *Theoretical Computer Science* 36 (1985) 265–289.
- [15] T. Mütze, Combinatorial gray codes — an updated survey, *The Electronic Journal of Combinatorics* 1000 (Jul. 2023). doi:10.37236/11023. URL <http://dx.doi.org/10.37236/11023>
- [16] A. Aggarwal, C. P. Rangan, T. Takaoka, An $o(1)$ time algorithm for generating multiset permutations, in: *Algorithms and Computation: 10th International Symposium, ISAAC’99 Chennai, India, December 16–18, 1999 Proceedings* 10, Springer, 1999, pp. 237–246.
- [17] J. Arndt, *Matters Computational: Ideas, Algorithms, Source Code*, 201. doi:10.1007/978-3-642-14764-7.
- [18] D. Adamson, A. Deligkas, V. Gusev, I. Potapov, On the hardness of energy minimisation for crystal structure prediction, *Fundamenta Informaticae* 184 (3) (2021) 181–203.

- [19] D. Adamson, A. Deligkas, V. V. Gusev, I. Potapov, The Complexity of Periodic Energy Minimisation, in: MFCS 2022, Vol. 241 of LIPIcs, 2022, pp. 8:1–8:15.
- [20] C. Collins, G. R. Darling, M. J. Rosseinsky, The Flexible Unit Structure Engine (FUSE) for probe structure-based composition prediction, Faraday discussions 211 (2018) 117–131.
- [21] D. Adamson, A. Deligkas, V. V. Gusev, I. Potapov, The k -centre problem for classes of cyclic words, in: SOFSEM 2023, Vol. 13878 of LNCS, Springer, 2023, pp. 385–400.
- [22] D. Adamson, V. V. Gusev, I. Potapov, A. Deligkas, Ranking Bracelets in Polynomial Time, in: CPM 2023, Vol. 191 of LIPIcs, 2021, pp. 4:1–4:17.
- [23] D. Adamson, Ranking and unranking k -subsequence universal words, in: WORDS, Springer Nature Switzerland, 2023, pp. 47–59.
- [24] F. Ruskey, C. Savage, T. Min Yih Wang, Generating necklaces, Journal of Algorithms 13 (3) (1992) 414–430.
- [25] M. Ackerman, E. Mäkinen, Three new algorithms for regular language enumeration, in: COCOON 2009, Springer, 2009, pp. 178–191.
- [26] M. Ackerman, J. Shallit, Efficient enumeration of words in regular languages, Theoretical Computer Science 410 (37) (2009) 3461–3470.
- [27] E. Mäkinen, On lexicographic enumeration of regular and context-free languages, Acta Cybernetica 13 (1) (1997) 55–61.
- [28] M. L. Schmid, N. Schweikardt, Spanner evaluation over slp-compressed documents, in: PODS’21, ACM, 2021, pp. 153–165.
- [29] M. L. Schmid, N. Schweikardt, Query evaluation over slp-represented document databases with complex document editing, in: PODS ’22, ACM, 2022, pp. 79–89.
- [30] T. Mütze, Combinatorial Gray codes - an updated survey, arXiv preprint arXiv:2202.01280 (2022).
- [31] K. Wasa, Enumeration of enumeration algorithms, arXiv e-prints (2016) arXiv:1605.

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

Algorithm 3 Enumeration algorithm for general alphabets

```
1: Global Variables:
2:  $w \in \Sigma^n$ 
3: Balanced Binary Search Trees  $T_1, T_2, \dots, T_\sigma$ 
4: function GENERALHAMILTONIANENUMERATION( $P \in \mathbb{N}^\sigma, q \in [\sigma]$ ,,
   pointer  $last\_state$ )
5:   if  $q = 2$  then
6:     HAMILTONIANENUMERATION( $(P_{\sigma-1}, P_\sigma), CurrentState()$ )
7:     Return To:  $last\_state$ 
8:   else if  $P_q = 0$  then
9:     Return To:  $last\_state$ 
10:  else
11:     $R_q \leftarrow \max(T_q)$ 
12:     $R_p \leftarrow \max_{x \in [q+1, \sigma]} T_x$ 
13:     $i \leftarrow \min(R_q, R_p)$ 
14:     $p_x \leftarrow \max_{< i} T_x, \forall x \in [q, \sigma]$ 
15:    while  $i > n - (P_1 + P_2 + \dots + P_\sigma)$  do
16:       $j \leftarrow 0$ 
17:      if  $w[i] = q$  then
18:         $j \leftarrow \min_{x \in [q+1, \sigma]} \min_{\geq i+1} T_x$ 
19:      else
20:         $j \leftarrow \min_{\geq i+1 - (\sum_{y \in [1, \sigma]} P_y)} T_q$ 
21:      end if
22:      Output:  $(i, j)$ 
23:      Remove $(T_{w[i], i})$ , Insert $(T_{w[i], j})$ 
24:      Remove $(T_{w[j], j})$ , Insert $(T_{w[j], i})$ 
25:       $w \leftarrow w \circ (i, j)$ 
26:      GENERALHAMILTONIANENUMERATION( $P(w), q$            +
   1,  $current\_state()$ )
27:       $P' \leftarrow (P_1, P_2, \dots, P_q) - P(w[i])$ 
28:      GENERALHAMILTONIANENUMERA-
   TION( $P', q, current\_state()$ )
29:       $i, x \leftarrow \max_{x \in [q, \sigma]} p_x$ 
30:       $p_x \leftarrow \max_{< i} T_x$ 
31:    end while
32:     $m \leftarrow n - (P_1 + P_2 + \dots + P_\sigma)$ 
33:     $j \leftarrow \min_{x \in [q, \sigma] \setminus w[m]} \min_{\geq m} T_x$ 
34:    Output  $(m, j)$ 
35:    Remove $(T_{w[j], j})$ , Insert $(T_{w[j], m})$ 
36:    Remove $(T_{w[m], m})$ , Insert $(T_{w[m], j})$ 
37:     $w \leftarrow w \circ (m, j)$ 
38:    GENERALHAMILTONIANENUMERATION( $P(w), q$            +
   1,  $current\_state()$ )
39:     $P' \leftarrow (P_1, P_2, \dots, P_n) - P(w[m])$ 
40:    GENERALHAMILTONIANENUMERATION( $P', q, last\_state$ )
41:  end if
```