Journal of Universal Computer Science, vol. 29, no. 2 (2023), 100-117
submitted: 3/6/2022, accepted: 12/12/2022, appeared: 28/2/2023 CC BY-ND 4.0

Search Algorithms for the Combinatorial Generation of Bordered Box Repetition-Free Words

Trienko Grobler
(Stellenbosch University, Stellenbosch, South Africa
https://orcid.org/0001-5274-0105, tlgrobler@cs.sun.ac.za)

Manfred Habeck
(Stellenbosch University, Stellenbosch, South Africa
https://orcid.org/0000-0002-8106-2654, 18217052@sun.ac.za)

Lynette van Zijl
(Stellenbosch University, Stellenbosch, South Africa
https://orcid.org/0000-0001-5735-0448, lvzijl@sun.ac.za)

Jaco Geldenhuys
(Amazon Web Services (AWS), USA
https://orcid.org/0000-0002-5636-6656, jgeldenh@amazon.com)

Abstract: A bordered box repetition-free word is a finite word w where any given factor of the form asa, with $a \in \Sigma$ and $s \in \Sigma^*$, occurs at most once. Four existing search algorithms are adapted to search for long bordered box repetition-free words over a given alphabet, giving an empirical result on the upper bound of the length of these words. Two algorithms use a tree-based search space, whilst the other two use a graph-based search space. For larger alphabets, the search space rapidly becomes intractable for the tree-based algorithms. In the case of the graph-based algorithms, we show a unique graph representation of bordered boxes that makes it possible to find long bordered box repetition-free words over a larger alphabet. The effectiveness of the four search algorithms are compared, and their respective worst case time complexities are compared against their performance in practice.

Keywords: Combinatorial generation, Combinatorics on words, Repetition, Upper bound
Categories: G.2.1, H.3.3
DOI: 10.3897/jucs.87330

1 Introduction

Combinatorics of words is the study of symbols and the sequences formed by these symbols [Lothaire 1997, Berstel and Karhumäki 2003, Berstel and Perrin 2007]. Early work [Thue 1906, Thue 1912] included the study of square-free words, which is a word with no two adjacent identical factors. More generally, a square can be seen as a pattern to be avoided. This led to a large body of work which addresses the avoidance of certain patterns in words and to the realization that some patterns are unavoidable [Bean et al. 1979, Zimin 1982, Baena-Garcia et al. 2010]. In contrast, less emphasis fell on the study of words that do not contain repeating factors – that is, the situation where specified factors may occur in a word, but only once.

An initial investigation into words that contain only one occurrence of a specified pattern, is that of Martin [Martin 1934]. He considered permutations of length k over an alphabet with n symbols that do not repeat in a word, and gave a greedy algorithm to generate a finite word w where no permutation of length k repeats. Note that Martin focused, as is the case in this article, on the generation of *finite* words. More recently, Carpi and De Luca [Carpi and De Luca 2001] studied repetitiveness (rather than non-repetitiveness) in finite words, where a box is the shortest non-repeating prefix or suffix of a word w. That is, a box is a factor of the form asb, where a, b are symbols from the alphabet, whilst s is a factor which occurs at least twice in w preceeded by different symbols and followed by different symbols. Again, the issue is to generate finite words, given the boxes.

In this article, we focus on the case where factors are bordered by the same symbol, rather than different symbols [Habeck 2022]. That is, we consider factors of the form axa rather than axb, with x any string from the alphabet. In addition, we concern ourselves rather with words that contain **no** repetitions of the form axa. We call these factors *bordered boxes*, or bboxes for short. The reader should note that bboxes are explicitly allowed to overlap, similar to the factors in De Bruijn words [Annexstein 1997].

A combinatorial generation algorithm of words over an alphabet with n symbols, which would contain no repeating bbox factors of any length, is not straightforward. We first develop some theoretical results, which we then utilize to adapt existing search algorithms so that these algorithms can find words with no bbox repetitions.

In these search algorithms, the search space can be represented as either a tree or a graph. Depth-first search (DFS) can then be utilized to find words that are bbox repetition-free. Clearly, to iterate over all possible tree nodes or paths in a graph is not computationally feasible. However, we show a graph construction which makes it possible to find long bbox repetition-free words via DFS in practice. We associate the vertices of the graph with specific bboxes of certain lengths, which in turn result in the indegree and the outdegree of the vertices to decrease as the length of the bboxes increase. DFS can then be modified to exploit this property. The order in which the vertices are explored is chosen so that the vertices associated with longer bboxes are explored first, preserving more unexplored edges within the graph. This simple modification dramatically reduces the run-time of the graph based search algorithm and enables the algorithm to find long bbox repetition-free words.

The novelty of this paper therefore does not lie in the search algorithms per se, but rather in their use in this specific problem. In addition, we compare the various search algorithms with one another quantitatively as well as qualitatively in the context of the problem at hand.

In the next section, we give the necessary definitions for bboxes, and the relevant combinatorial properties of bboxes. In Section 3 we elaborate on the maximum length of a bbox repetition-free word for any alphabet of size n. The generation-by-search algorithms that were developed are presented in Section 4, and the empirical results follow in Section 5.

2 Background Definitions and Properties

We assume that the reader has a general background in combinatorial generation, such as in [Charalambides 2002, Ruskey 2003, Stanley 1986] and others.

An alphabet is a finite set Σ of symbols, and Σ^* is the Kleene closure of Σ. The number of symbols in Σ is indicated by $|\Sigma|$. For clarity, if Σ contains n symbols, we also write Σ_n or Σ_n^*. A word w is a sequence of symbols from Σ; that is, $w \in \Sigma^*$. In this work, only finite words are considered. The length $|w|$ of a word w is the number of symbols in w. A word $x \in \Sigma^*$ is a factor of w if $w = uxv$ for some words $u, v \in \Sigma^*$.

Given a word w, a factor x of w is called a *bbox* if the first and last symbols of x are identical. That is, factor $x = x_1x_2 \cdots x_m$ is a bbox if $x_1 = x_m$. It is assumed that the shortest possible bbox is a word of length two, when x is of the form $x = aa$, for $a \in \Sigma$. Therefore, neither the empty string nor single symbols are considered to be bboxes. A bbox is called an *inner bbox* if it does not contain another bbox. That is, for a bbox $x = as_1s_2 \cdots s_ka$, it holds that x is an inner bbox if $s_i \neq a$ for $1 \leq i \leq k$, and $s_i \neq s_j$ for $1 \leq i, j \leq k$, for $i \neq j$. Note that, for a bbox of length greater than two, this means that none of its cyclic shifts are bordered, as all the symbols s_i are unique and different from the border characters a. On the other hand, if bbox x contains another bbox, it is called an outer bbox.

A word w is a bbox repetition-free (BRF) word if every bbox in w is unique. A word w is a maximal bbox repetition-free (MRF) word if it is bbox repetition-free, and no symbol can be added at the end of w without causing a repeating bbox. A word w is a longest maximal bbox repetition-free (LMRF) word if w is MRF and it has the longest possible length for a given n; that is, for any MRF word v over Σ_n^*, it holds that $|w| \geq |v|$.

Example 1. Let $n = 2$ and $\Sigma = \{a, b\}$. The word $w = abaabba$ contains amongst others bboxes $aba, abaa, aa, baab$ and bb. Here, aba, aa and bb are inner bboxes, while $abaa$ and $baab$ are outer bboxes. Also, w is bbox repetition-free, but is not MRF, as the word wb is bbox repetition-free.

A few obvious but useful properties of inner bboxes follow directly from the definitions.

P1: Let $w \in \Sigma_n^*$ be an inner bbox. Then w is of the form asa, with $a \in \Sigma_n$ and $s \in \Sigma_n^*$. It must then hold that $2 \leq |w| \leq n+1$, because the maximum length of s can only be $n-1$ if all the symbols in s are unique and are not a. If the symbols are not unique, then w cannot be an inner bbox.

P2: Let w' be any outer bbox in a word w; that is, $w' = a \ldots bxb \ldots a$. If the outer bbox w' repeats, then its associated inner bbox bxb must also repeat – this implies that in the search for repetitions, one need only consider the repetitions of inner bboxes to decide whether a word w is BRF.

A brute force search for repetitions of bboxes needs to find all repetitions of bboxes of any length. However, by property P2, the search space can be reduced by only checking for repetitions of inner bboxes.

Given the total number of unique inner bboxes, and their respective lengths, a first upper bound on the length of an LMRF word can be calculated as the sum of the lengths of all possible inner bboxes. The number of inner bboxes of length k over an alphabet Σ_n is given by $J(n,k) = \frac{n!}{(n-k+1)!}$. This follows directly, as for any inner bbox $t = asa$ with $|t| = k$, s is some permutation of $k-2$ unique symbols, where $2 \leq k \leq n+1$. Hence, the total number of all inner bboxes over Σ_n is then the sum over all k, which is simply $J(n) = \Sigma_{i=1}^{n} \frac{n!}{(n-i)!}$. Expressed recursively, $J(n) = n(J(n-1)+1)$. The sum $K(n)$ of the lengths of all inner bboxes is therefore the sum of the number of all inner bboxes of length k, times k. Therefore, $K(n) = \Sigma_{j=1}^{n} \frac{(n-j+2)n!}{(j-1)!} = n(K(n-1)+1) + J(n)$.

Since inner bboxes can overlap, the bound $K(n)$ will not necessarily be tight. If one considers any BRF, and look at any factor with two successive inner bboxes, it is possible that there is no overlap between the two inner bboxes. For example, the factor $\cdots aabcc \cdots$ contains the successive inner bboxes aa and cc, which are separated by the symbol b. On the other hand, the factor $\cdots abcab \cdots$ contains two successive inner bboxes of length four ($abca$ and $bcab$), but the length of the factor is five and not eight, because $abca$ and $bcab$ overlap by three symbols (bca). Since we are interested in generating the longest possible MRF over a given alphabet, it is of interest to know what the longest factor is that can be formed by any two successive inner bboxes. Lemma 1 establishes this maximum length.

Lemma 1. *The longest BRF word $w \in \Sigma_n^*$ which can be formed by two successive inner bboxes b_1 and b_2, has length $n + 2$.*

Proof. Let $w \in \Sigma_n^*$ be a BRF word which consists of two successive inner bboxes b_1 and b_2. Without loss of generality, one can write $b_1 = s_1 v_1 s_1 = s_1 s_2 \cdots s_{m-1} s_1$, where $s_1 \neq s_i \neq s_j$, for $2 \leq i, j \leq m - 1$ (for $i \neq j$), since b_1 is an inner bbox. Similarly, $b_2 = s_h v_2 s_h = s_h \cdots s_k s_h$, where $s_h \neq s_i \neq s_j$, for $h + 1 \leq i, j \leq k$ (for $i \neq j$), since b_2 is an inner bbox.

First consider the case where b_1 and b_2 show no overlap:

$$w = \overbrace{\mathbf{s_1} s_2 \cdots s_{m-1} \mathbf{s_1}}^{b_1} s_m \cdots \underbrace{\mathbf{s_j} \cdots s_k \mathbf{s_j}}_{b_2}$$

None of the symbols s_m up to s_k can occur in b_1, as that would form a new inner bbox which would contradict the requirement that b_2 is the first inner bbox that occurs after b_1. Therefore, s_m up to s_k are unique, and the maximum value of k is the number of symbols in the alphabet, so that $k \leq n$.

Simple arithmetic shows that $|b_1| = m$, and $|b_2| = k - j + 2 \leq n - j + 2$. The number of non-overlapping symbols between b_1 and b_2 is $(j - m + 1) - 1 = j - m$. Therefore, $|w| = m + k - j + 2 + j - m = k + 2 \leq n + 2$ as is required.

For the second case, where b_1 and b_2 overlap, the argument is similar. Let

$$w = \rlap{$\overbrace{\phantom{\mathbf{s_1} s_2 \cdots \mathbf{s_j} \cdots s_{m-1} \mathbf{s_1}}}^{b_1}$}\mathbf{s_1} s_2 \cdots \underbrace{\mathbf{s_j} \cdots s_{m-1} \mathbf{s_1} s_m s_{m+1} \cdots s_k \mathbf{s_j}}_{b_2}$$

where $1 \leq j \leq m - 1$. Again, $|b_1| = m$, and $|b_2| = k - j + 2 \leq n - j + 2$. The number of overlapping symbols is now $(m - 1 - j + 1) + 1 = m - j + 1$. The result follows. $\square$

Lemma 1 is referred to as the $n + 2$-rule throughout the remainder of this article. Note that the value given in Lemma 1 is a maximum value, and is not necessarily reached for any two successive inner bboxes b_1 and b_2. Nevertheless, for any inner bbox b_1, there does exist an inner bbox b_2 such that b_1 followed by b_2 adheres to the $n + 2$-rule. For algorithmic purposes, the $n + 2$-rule is called an extension of inner bbox b_1 to b_2. This leads to Lemmas 2 and 3 below, which narrow down these extensions of inner bboxes.

Example 2. Consider $\Sigma_3 = \{a, b, c\}$. The inner bbox aba can be extended by inner bbox $bacb$, to form the string $abacb$ of length $n + 2$. On the other hand, if inner bbox aba is extended by bab, then the resulting string $abab$ has length less than $n + 2$.

Lemma 2. *Let $w \in \Sigma_n^*$ be an inner bbox. Then w can be extended by a suffix $p \in \Sigma_n^*$, with $|wp| \leq n+1$, so that the only bbox in wp is w.*

Proof. Let $w = s_1 s_2 \cdots s_{m-1} s_1$, with $|w| = m$. From property P1, it follows that $m \leq n+1$. If $m = n+1$, let $p = \epsilon$, and the lemma holds trivially. If $m < n+1$, choose p to be any permutation of $\Sigma_n \backslash \{s_1, \ldots, s_{m-1}\}$ so that $|p| = k$, where $0 < k < n-m+1$. Clearly, wp still only contains the bbox w, as none of the symbols in w is used in p. Moreover, $|wp| = m + k \leq m + (n-m) + 1 = n+1$. □

Corollary 1. *Let $w \in \Sigma_n^*$ be an inner bbox. Then w can be extended to the left by a prefix $p \in \Sigma_n^*$, with $|pw| \leq n+1$, so that the only bbox in pw is w.*

Example 3. Let $\Sigma_3 = \{a, b, c\}$, and consider an inner bbox $w = aba$. Then prepending a prefix c to w gives the word $caba$. Here, $|cw| = 4 = n+1$, and cw is bbox repetition-free. Note that Corollary 1 does not allow the use of any symbols other than c in the prefix.

Lemma 2 is easily extended to hold for any BRF word wq, with q the last inner bbox at the end of the word.

Lemma 3. *Let $w \in \Sigma_n^*$ be bbox repetition-free, so that $w = tb_k$, where b_k denotes the last inner bbox contained in w. Then wp will remain bbox repetition-free if w is extended with a suffix p as in Lemma 2 above, with $|b_k p| \leq n+1$.*

Proof. From Lemma 2, there exists a suffix p such that $b_k p$ will not contain any other bbox than b_k, with $|b_k p| \leq n+1$. Any symbol contained in p will create at least one outer bbox x when appended to w, but since w is BRF, this outer bbox cannot cause a repetition. □

As a direct corollary, the same result holds for a prefix prepended to a BRF word.

Given the $n+2$-rule and the idea of extensions of an inner bbox, one can capture the length of words with the properties required for the search algorithms.

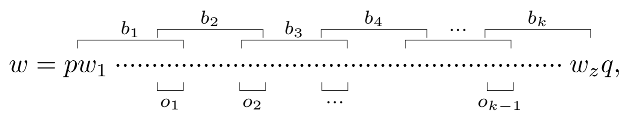
Theorem 1. *If $w \in \Sigma_n^*$ such that w*

1. *is bbox repetition-free,*
2. *contains all inner bboxes over Σ_n,*
3. *consists of consecutive inner bboxes that adhere to the $n+2$ rule, and*
4. *has been extended as per Lemma 3,*

then $|w|$ is equal to

$$B(n) = (n+2)[J(n) - 1] - K(n) + 2(n+1). \tag{1}$$

Proof. Assume that w contains k inner bboxes over Σ_n. Let the overlap lengths between successive inner bboxes be $o_1, o_2, \ldots, o_{k-1}$ such that bbox b_i and bbox b_{i+1} share o_i symbols. The word w will be of the following form:



where p and q, respectively, denote the bbox-free prefix and suffix used to extend w in accordance with Lemma 3. The length of $|w|$ is the sum of the lengths of all the inner bboxes, minus the sum of the lengths of the overlaps between the inner bboxes, plus the lengths of the prefix p and the suffix q:

$$|w| = \sum_{i=1}^{k} |b_i| - \sum_{i=1}^{k-1} o_i + |p| + |q|.$$

All consecutive inner bboxes adhere to the $n + 2$ rule, and hence

$$\begin{aligned}
|w| &= \sum_{i=1}^{k} |b_i| - \sum_{i=1}^{k-1} o_i + |p| + |q| \\
&= \sum_{i=1}^{k-1} (|b_i| - o_i) + |p| + |b_k q| \\
&= \sum_{i=1}^{k-1} (n + 2 - |b_{i+1}|) + |p| + |b_k q| \\
&= (k-1)(n+2) - \sum_{j=2}^{k} |b_j| + |p| + |b_k q| \\
&= (k-1)(n+2) - \sum_{j=1}^{k} |b_j| + |pb_1| + |b_k q| \\
&= (n+2)[J(n) - 1] - K(n) + 2(n+1)
\end{aligned}$$

□

Note that Lemma 3 and Theorem 1 do not provide existence proofs. The only guarantee is that, if w exists, then its length would be bounded by Equation 1.

From an algorithmic point of view, it is helpful to note that $B(n)$ can be expressed recursively as $B(n) = n \cdot (B(n-1) + 2)$. The relationship amongst $B(n)$ and $J(n)$ follow directly, as $B(n) = 2J(n)$.

Finally, note a restriction on the suffix of an LMRF word.

Theorem 2. *If $w \in \Sigma_n^*$ is MRF, then w ends with a permutation of the symbols of Σ_n.*

Proof. Let w be an MRF word with $|w| = k$, where

$$w = w_1 w_2 \cdots w_i \cdots w_j \cdots w_{k-1} w_k,$$

and $j = k - n + 1$ (that is, w_j to w_k are the last n symbols in w). The proof is by contradiction. Assume that the last n symbols in w is not a permution of the symbols in Σ_n. Then there is at least one symbol $a \in \Sigma$ which does not appear in $w_j \cdots w_k$. But a must appear in w, since w is MRF. Suppose that the last occurrence of a in w is at symbol w_i such that $i < j$, with $1 \leq i, j \leq k$. Then, by the pigeonhole principle, at least one of the symbols in $w_j \cdots w_k$ must repeat, thus forming one or more bboxes. Pick any of these bboxes and call it z.

Now consider appending a to w:

$$w' = w_1 \cdots w_i \overbracket{a\, w_{i-1} \cdots \underbracket{\; w_j \;}_{z} \cdots w_{k-1} w_k\, a}^{y}$$

Since w is MRF, w' now contains a repeating bbox y which appears earlier in w' and therefore also in w. But that means that z also appears at least twice in w and w is not bbox repetition-free and not MRF. This is a contradiction, and therefore a must appear in the last n symbols of w. This is true for all symbols of Σ_n. Therefore, every symbol of Σ_n occurs somewhere in the last n symbols of w and the last n symbols form a permutation of Σ_n. □

Corollary 2. *If w is LMRF, then w starts and ends with a permutation of the symbols of Σ_n.*

3 Properties of LMRF Words

Recall that the sum of the lengths of all inner bboxes over Σ_n^* provides an upper bound on the length of an LMRF word, but due to bboxes that overlap, this bound is not tight. This section demonstrates that these overlaps between inner bboxes display a noticeable pattern. Consider the example below.

Example 4. Let $\Sigma_3 = \{a, b, c\}$ and

$$w = \overbracket{ab\mathbf{ca}}\ \overbracket{\mathbf{c}bc}\ abacabbcaabcbacbabccabc\,. \tag{2}$$

An exhaustive enumeration of all MRFs over Σ_3 shows that w is indeed LMRF. Now consider all the inner bboxes in w, with their corresponding overlaps. For example, the first three inner bboxes in w are $abca$, cac and cbc. Here, $abca$ and cac overlap by two positions, and cac and cbc overlap by one position. Use the notation $b_1(-k)b_2$ to refer to the overlap when the inner bbox $b_1 \in \Sigma_n^*$ is extended by $b_2 \in \Sigma_n^*$, by letting the last k symbols from b_1 overlap with the first k symbols of b_2. Similarly, $b_1(+k)b_2$ indicates that b_1 extended with b_2 has no overlap, and k symbols appear between b_1 and b_2, where those k symbols are not already present in b_1 and b_2. Rewriting w into its constituent inner bboxes in this fashion, gives

$$\begin{aligned} w = abca(-2)cac(-1)cbc(-2)bcab(-2)aba(-1)aca(0)bb(+1)aa(0)bcb(-2)\\ cbac(-3)bacb(-3)acba(-2)bab(0)cc(-1)cabc\,. \end{aligned}$$

By inspection, w contains all inner bboxes over Σ_3. Moreover, the length of any $b_1(-k)b_2$ and $b_1(+k)b_2$ is equal to $n+2$ (in this case, $n+2=5$). In other words, $b_1(-k)b_2$ and $b_1(+k)b_2$ have been extended maximally and as such adhere to the $n+2$-rule (see Lemma 1). These two observations (henceforth referred to as the LMRF assertions) can be used to compute the length of the example LMRF in Equation 2, by employing Theorem 1, with $n = 3$:

$$\begin{aligned} |w| &= \overbrace{[4-2+3]+[3-1+3]+\cdots+[2-1+4]}^{(n+2)[J(n)-1]} - \overbrace{(4+3+3+\cdots+2+4)}^{K(n)} \\ &\quad + \overbrace{(4+4)}^{2(n+1)} \\ &= \overbrace{5+5+\cdots+5}^{5\cdot 14=70} - \overbrace{(4+3+3+\cdots+2+4)}^{48} + \overbrace{(4+4)}^{8} \\ &= 30. \end{aligned} \tag{3}$$

An empirical verification was performed over arbitrarily generated LMRFs, up to $n = 4$. It demonstrated that if $w \in \Sigma_n^*$ is LMRF, then it contains all inner bboxes over Σ_n, and all factors in w of the form $b_1(-k)b_2$ or $b_1(+k)b_2$ have a length of $n+2$. These LMRFs were generated without employing the LMRF assertions in the generation process. It is therefore conjectured that

Conjecture 1. *If $w \in \Sigma_n^*$ is LMRF, then $|w|$ is equal to*

$$B(n) = (n+2)[J(n)-1] - K(n) + 2(n+1).$$

Given the properties of MRF words developed above, we turn our attention to specific search algorithms for LMRF words.

4 Generating LMRF Words

Four search algorithms to find LMRF words over Σ_n are presented in this section. The first two algorithms are adaptations of standard tree-based search algorithms, whilst the latter two have been developed based on a graph representation of inner bboxes.

4.1 Backtracking

The simplest way to search for MRF words is to employ a backtracking algorithm. That is, use a recursive depth-first search algorithm that systematically finds longer BRF words. The pseudo-code for this backtracking algorithm is depicted in Algorithm 1. The algorithm starts with the word w, which is initially empty. It then appends a symbol $a \in \Sigma_n$ to w to form w'. If w' is a bbox repetition-free word, the recursive search continues (the process starts anew and w' becomes the new initial starting string). If w' is not BRF, an alternative character from Σ_n is appended. If there are no more characters in Σ_n left to append, the algorithm terminates (after exploring all possible subtrees through backtracking).

Example 5. Algorithm 1 generates the search tree depicted in Figure 1 when $n = 2$. Every path in this tree is a valid MRF word. The path that generates the LMRF $baabbaba$ is highlighted in bold.

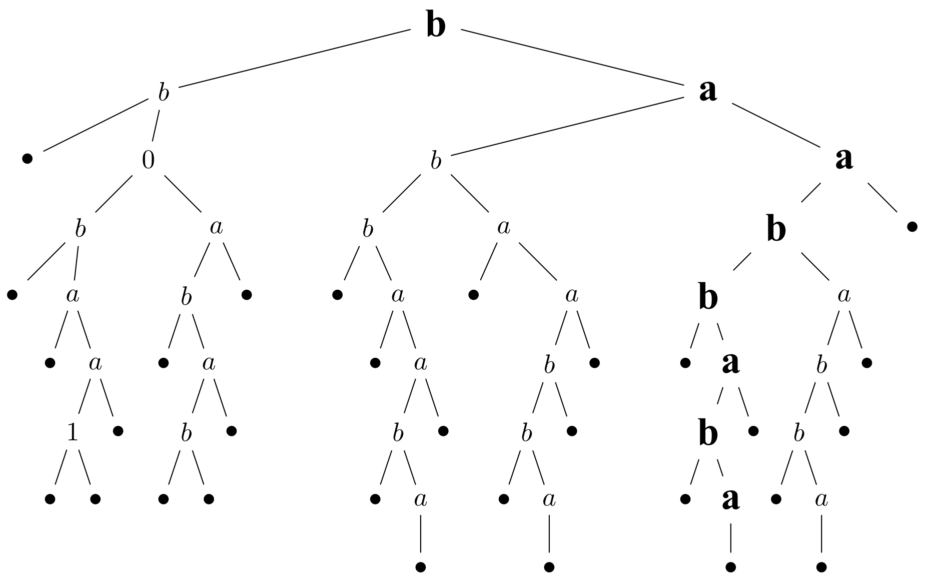


Figure 1: A tree of all possible valid BRFs, starting with symbol b, over $\Sigma = \{a, b\}$.

Algorithm 1: Backtracking algorithm for generating BRF words

```
$w \leftarrow \varepsilon$
Function explore(w):
    for $a \in \Sigma$ do
        $w' = w + a$
        if valid($w'$) then
            explore($w'$)
            record($w'$)
        end
    end
```

The backtracking algorithm is simple to implement, but resource intensive. It is enumerative and not a search algorithm per se. Hence, as expected, even with many programming optimisations, the backtracking implementation was inefficient. It could not find an LMRF for $n > 3$ in reasonable time. For $n = 4$, it found an MRF of length 98, which is shorter than the known LMRF length of 128.

A more sophisticated search algorithm can be produced by adapting the well-known Monte Carlo tree search algorithm. Where the backtracking algorithm generates all LMRFs over a given alphabet, the Monte Carlo tree search approach attempts to find an example of one LMRF, by selecting favorable branches to traverse in the decision tree.

4.2 Monte Carlo tree search

Monte Carlo tree search (MCTS) is often used in game theory to find optimal moves in a two-player game [Nijssen 2013]. Finding LMRF words can be viewed as a deterministic single player game, where the goal of the game is to stay alive as long as possible. The game starts with an empty word w. During each move of the game, a player can append one symbol from Σ_n to w. If the player adds a symbol that causes a repeating bbox, the game ends. If the word generated by the player is LMRF, the player wins, or else the player loses. The progress of the game is tracked by a tree, where each node represents a move by the player.

To use MCTS to search for LMRF words, consider an existing BRF word w. Then each node in the tree represents the addition of another alphabet symbol in the construction of w, so that the word w can be constructed by traversing the tree from the root node to the current node. To make a move in the game from a currently selected node, all its children are created (the expansion phase). Then random walks are initiated from each child, to investigate possible paths in the tree (the simulation phase). A score is kept in each node (see below), and when the simulation is done, all the node scores are updated backwards up to the root node (the backpropagation phase). The scores are then used to select a new node based on the scores (the selection phase), and the process repeats.

The success of MCTS in searching for LMRF words depends to a large degree on the score calculated for each node – one wants to select nodes with the biggest chance of leading to an LMRF word. The upper confidence bounds applied to trees (UCT) scoring method [Browne et al. 2012] was adapted for use in our experiments. For every node i, let S_i be the total length of all the BRF words found by random walks initiated from node i or any of its subsequent children, and let v_i represent the total number of simulations that were initiated from node i or any of its subsequent child nodes. A win rate X_i can then be calculated as $\frac{S_i}{v_i \cdot B(n)}$, with the value of $B(n)$ as given in Theorem 1. The quantity X_i, therefore, represents the average normalized search depth reachable from node i.

The calculation of the UCT score for any node i, with parent node s, is then given by

$$\mathrm{UCT}_i = X_i + C\sqrt{\frac{\ln v_s}{v_i}}, \tag{4}$$

where C is a hyper-parameter which determines to what degree the search algorithm is allowed to explore poorly explored subtrees (v_i and per implication v_s have already been explored). Thus, the UCT score uses X_i for exploitation; this value is high for nodes from which one can reach deeper search depths (that is, generate longer MRFs), while the second term in UCT_i corresponds to exploration; it is high for nodes from which few simulations were conducted.

As MCTS is a stochastic algorithm, multiple experiments were run. Surprisingly, MCTS was able to find an LMRF for $n = 4$, but not for $n = 5$.

To find a better search algorithm, the inner bboxes for a given value of n were organised into a graph, and two search algorithms were developed, based on the traversal of the graph. Both search algorithms make use of the LMRF assertions, as developed in Section 3. All LMRFs generated by these two approaches were tested and were in fact bbox repetetition free.

4.3 Node search

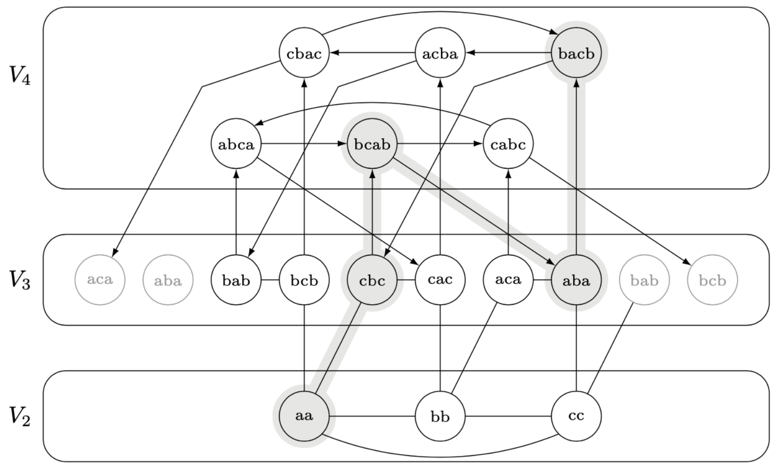
A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where an edge (v_1, v_2) connects vertices v_1 and v_2. A path in G is a sequence of edges. A Hamiltonian path in G is any sequence of edges of G which passes through all the vertices in V exactly once [Diestel 2017]. Given alphabet Σ_n, consider the graph $G_n = (V, E)$, where V is the set of all possible inner bboxes over Σ_n, and $(b_1, b_2) \in E$ if and only if b_2 is a maximal inner bbox extension of b_1. That is, b_1 extended by b_2 adheres to the $n+2$-rule as in Lemma 1.

If the LMRF assertions, as listed in Section 3, are indeed necessary and sufficient conditions for a word w to be LMRF, then it follows that LMRFs can be generated by finding and traversing Hamiltonian paths in G_n. The traversed paths need to be Hamiltonian to guarantee that the words formed by the traversal are repetition-free. As per Lemma 3, a word w formed using this graph traversal approach may still have to be extended with an appropriate prefix p and/or suffix q to make it LMRF.

Example 6. The graph G_3 is depicted below, where V_k indicates the vertices depicting inner bboxes of length k. For readability purposes, not all of the edges in G_3 are shown; in particular, the edges between V_2 and V_4 are not drawn. Edges with arrowheads are directed; all other edges are bi-directional. The light gray vertices of V_3 are simply duplicates to simplify the drawing of the edges. A partial traversal of this graph is depicted in gray. The string formed via this traversal is

$$\begin{aligned} w &= aacbcabacb \\ &= aa(0)cbc(-2)bcab(-2)aba(-2)bacb. \end{aligned}$$

Note that w consists of only inner bboxes, and it adheres to the $n+2$-rule. Moreover, it can easily be verified that the LMRF in Equation 2 can be associated with a Hamiltonian path in G_3.



The adjacency matrix **A** for G_3 is shown below, with $(b_i, b_j) = 1$ if b_j is a maximal extension of b_i.

A	aa	bb	cc	aba	aca	bab	bcb	cac	cbc	$abca$	$acba$	$bacb$	$bcab$	$cabc$	$cbac$
aa	0	1	1	0	0	0	1	0	1	1	1	0	0	0	0
bb	1	0	1	0	1	0	0	1	0	0	0	1	1	0	0
cc	1	1	0	1	0	1	0	0	0	0	0	0	0	1	1
aba	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0
aca	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0
bab	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0
bcb	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1
cac	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0
cbc	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
$abca$	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
$acba$	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1
$bacb$	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0
$bcab$	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0
$cabc$	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0
$cbac$	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0

In general, the indegree and the outdegree of the vertices belonging to V_k decrease as k increases. The number of incoming and outgoing edges per inner bbox size, for G_2 through G_7, is shown in Table 1. An entry is marked with '-' if inner bboxes of that length cannot be constructed. For example, in G_3, there are no inner bboxes of length five. The number of edges is naturally ordered from large to small; the longer the inner bbox, the fewer edges it has.

Inner bbox length	G_2	G_3	G_4	G_5	G_6	G_7
2	2	6	21	88	445	2676
3	2	3	8	27	112	565
4	-	3	4	10	33	136
5	-	-	4	5	12	39
6	-	-	-	5	6	14
7	-	-	-	-	6	7
8	-	-	-	-	-	7

Table 1: The indegree and outdegree per inner bbox size of G_n.

Let I denote the list of inner bboxes ordered lexicographically on length (as in the adjacency matrix **A** above). Let I' denote the reverse of this list. Recall that the search algorithm will attempt to find a Hamiltonian path in G_n.

The simplest approach to search for Hamiltonian paths in G_n is to employ DFS, with a defined order in which the vertices of G_n are to be explored [Diestel 2017]. The pseudo-code for finding all the Hamiltonian paths in an arbitrary graph G via DFS is presented in Algorithm 2. Note that in Algorithm 2 the order in which the nodes are to be explored is not made explicit. Moreover, the algorithm will find all the Hamiltonian paths in an arbitrary graph, and does not terminate when the first path is found. In this work, we used a slighlty modified version of this algorithm in which the program terminates as soon as a Hamiltonian path is found (making it non-enumerative). Two exploration strategies are considered. In the first approach, the next vertex to expand is found via its rank in I. That is, from all the valid inner bboxes that can be expanded at any point in time, the one with the lowest rank in I is chosen. In the second approach, the inner bbox with the lowest rank in I' is chosen. The first approach is called basic node search, and the latter approach reverse node search. The latter approach should intuitively outperform the former – the inner boxes with longer length have fewer edges, and so the reverse search covers fewer edges and leaves more edges open for further exploration within the graph [Seeja 2018] while it searches (see **A** and Table 1 to verify). Note that backtracking is used in both approaches. As soon as a dead end is reached, the algorithms backtrack to the closest vertex which still contains edges connected to vertices that have not been visited yet. As soon as a Hamiltonian path is found, the search is terminated.

Algorithm 2: Finding all Hamiltonian paths in a graph G via DFS.

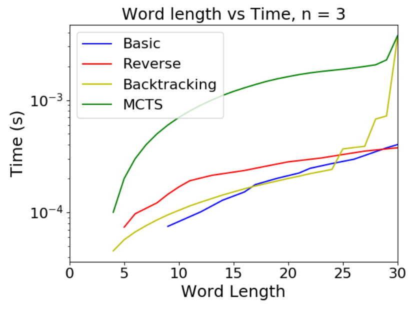
```
Function DFS(Graph G, node V, path P):
    add node $V$ to path $P$
    if path $P$ contains all nodes then
        save($P$)
    end
    for each node $W$ connected to node $V$ do
        if node $W$ is not in path $P$ then
            DFS($G$,$W$,$P$)
        end
    end
    remove node $V$ from path $P$

Function Hamiltonian(Graph G):
    for each node $K$ in $G$ do
        DFS($G$,$K$,$\emptyset$)
    end
```

The results from the basic node search improved upon both the brute force and the MCTS, in the sense that it could find longer MRFs in the same time. Nevertheless, it still failed to find an LMRF for $n = 5$ in a reasonable time. In contrast, the reverse node search could find an LMRF for both $n = 5$ and $n = 6$.

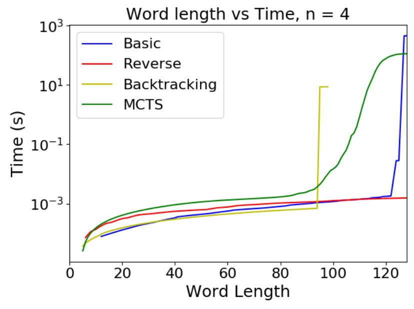
5 Analysis

The four search algorithms have different inherent properties, which influences the comparison of the results obtained. Note that backtracking and MCTS are both tree based search algorithms, while the remaining two algorithms are graph based search algorithms. In addition, MCTS is a stochastic search algorithm; the remaining algorithms are all deterministic.

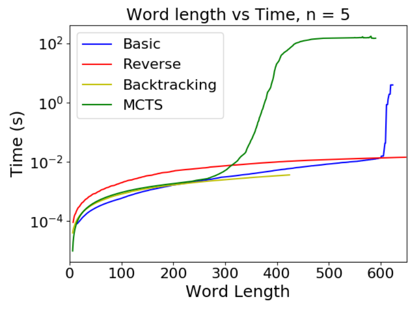
One would expect that the exploitation of domain knowledge would increase the chances of finding LMRF words. This is indeed the case. Backtracking requires no domain knowledge, other than the definition of an LMRF word. MCTS uses partial domain knowledge, as it employs $B(n)$. The two node searches exploit knowledge about the properties of LMRF words; in particular, the LMRF assertions.



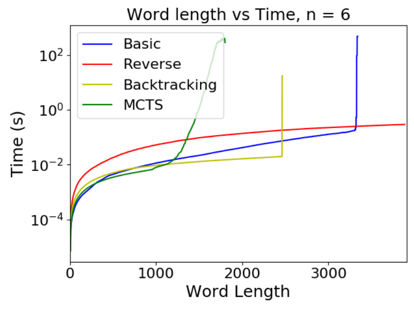
(a) $n = 3$



(b) $n = 4$



(c) $n = 5$



(d) $n = 6$

Figure 2: The average time taken for each search algorithm to find a BRF word of length $k \leq B(n)$, for $n \in \{3, 4, 5, 6\}$.

The average time for each of the four algorithms to find a BRF word of length $k \leq B(n)$ (for $3 \leq n \leq 6$) is presented in Figure 2. The search experiments were repeated multiple times; the average result obtained is plotted. Each algorithm was allowed to search through the allotted search space for 10 minutes at a time (experiments showed that this was a choking point for all the algorithms). All of the experiments were conducted on an Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz with 8 cores.

Most of the curves in Figure 2 are monotonically increasing; as expected, the algorithms generally take longer to find longer BRF words. However, as the value of n increases, the curve for MCTS reaches a non-increasing plateau. Because fewer of the search experiments for MCTS are able to find long BRF words, the measurements reported towards the end of the curves have a higher error associated with them, due to the stochastic nature of the algorithm.

The first words found by the different algorithms are of varying lengths. This is not surprising, as some algorithms search by adding individual symbols to existing words, while others add entire inner bboxes. Moreover, some algorithms, like the node search algorithms, require the addition of a prefix and a suffix as well. The words used to initialise the different algorithms also vary. For backtracking and MCTS the starting word is a permutation of Σ_n (by Theorem 2), while the node search algorithms start with an inner bbox padded with an appropriate prefix and suffix (see Lemma 2).

Most algorithms experience a steady increase in execution time as a function of the word length k. However, at some value of k, most algorithms reach a choking point – a sudden increase in execution time is required to make further progress. Soon after the choking point, most algorithms get stuck and are unable to find longer BRF words. Enumerations of BRF lengths for $n = 3$ and $n = 4$ show that, at the choking points, the allotted search space contains many relatively long BRF words and only a few LMRF words. Interestingly, MCTS experiences a less abrupt increase in execution time at the choking point. This is due to the fact that it explores its search space somewhat intelligently and therefore still has favorable avenues to explore at the choking point. This is exacerbated exacerbated by the fact that MCTS is a stochastic algorithm where only average results are reported. The choking point of the graph based approaches generally appear at a longer word length than for the tree based approaches. This is to be expected, as the tree-based search recurses through branches with a similar prefix. The graph-based approach searches through more varied prefixes. Note that a choking point is not visible in the curves associated with reverse node search. Because the values of n are small, the reverse node search did not have to backtrack in its search – this will however not be true in general for larger values of n.

The execution times of the algorithms are comparable, until each algorithm reaches its choking point. The noteable exception here occurs for MCTS when $n = 3$. This can most likely be attributed to the overheads that are required to set up the MCTS algortihm. Generally speaking, backtracking performs the worst, only reaching the theoretical upper bound for $n = 3$. MCTS performs slightly better, reaching the theoretical limit for both $n = 3$ and $n = 4$. The basic node search also reaches the theoretical limit for $n = 3$ and $n = 4$, while reverse node search reached the theoretical limit for $3 \leq n \leq 6$. As pointed out in Section 4.3, this is due to the fact that its exploration strategy preserves the most unexplored edges in G_n. The algorithm will undoubtedly require backtracking for larger n, as it is dependent on a deterministic algorithm for finding Hamiltonian paths in G_n. Lastly, one should note that backtracking can outperform MCTS. When the search is small, a random walk is an adequate strategy for exploring the search space, but for larger n a more sophisticated strategy is required. This could also be attributed to the need for further optimisation of the hyper-parameters of MCTS.

The experiments also illustrate that the worst case (exponential) time complexity of the algorithms can be alleviated to some degree in practice. Recall that $J(n)$ denotes the total number of inner bboxes that exist over an alphabet Σ_n and that $B(n) = 2J(n)$. Furthermore, the worst case time complexity of bactracking and MCTS will occur if the entire tree is to be generated; that is, if every vertex is to be created. A tree with a branching factor of r and a depth of h has $(r-1)^{-1}(r^h - 1)$ vertices [McConnell 2001]. The tree generated here has a branching factor of n and a depth of $2J(n)$. The worst case time complexity is, therefore, roughly $O(n^{2J(n)})$. In other words, neither backtracking nor MCTS can be executed in polynomial or linear time. It should be noted that the worst case complexity will never occur in practice, since the tree is continually pruned as it is expanded when backtracking is used, while it is explored cleverly in the case of MCTS. Having said this, even with all of these optimisations these two algorithms fail to execute in polynomial time, as the results clearly indicate. Moreover, the execution time of DFS, when used to find the Hamiltonian paths in an arbitrary graph G, has a worst case time-complexity of $O(v!)$, where v denotes the total number of nodes in G [McConnell 2001]. This occurs when the algorithm has to enumerate through all possible paths in G that consist of only unique vertices, of which there are at most $v!$ possibilities. In the case of G_n, therefore, the worst case time complexity is given by $O(J(n)!)$. Given these theoretical upper bounds, neither the tree based nor the graph based algorithms ought to be capable of finding LMRF words for larger n. This is clearly highlighted by the fact that basic node search already fails to find an LMRF word for $n = 5$ within the allotted search time. The reason that reverse node is successful, is due to the unique structure of G_n, where the indegree and the outdegree of V_k deacreases as k increases (see Table 1). By prioritizing the exploration of the longer inner bbox vertices, the reverse node search algorithm maximises the number of unexplored edges that remain in the graph. Hence, the DFS traversal requires almost no backtracking for small values of n. In other words, it immediately finds a Hamiltonian path, and it does so in linear time. Nevertheless, we hypothesize that the algorithm will, for larger n, have to rely on backtracking which will increase its execution time significantly. Exploring this aspect in more detail will be done as part of a future endeavour. Lastly, the astute reader may be wondering why we do not present execution times, but rather kept track of the longest MRF that each of the algorithms could find within 10 minutes. The theoretical time complexity of the algorithms highlight the reason why. Most of the algorithms were not able to make much progress after 10 minutes (they reached a choking point), precisely since the search space grows exponentially for larger n. Most of the algorithms made little progress even for substantial longer periods. In contrast, the reverse node search always found an LMRF within the allotted time for small values of n.

The results are summarised in Table 2.

	$B(n)$	**Backtracking**	**MCTS**	**Basic node**	**Reverse node**
$n = 3$	30	30	30	30	30
$n = 4$	128	98	128	128	128
$n = 5$	650	424	588	623	650
$n = 6$	3912	2468	1802	3341	3912

Table 2: Maximum BRF word length found for each search algorithm.

6 Conclusion

In this paper some theoretical properties of longest bbox repetition-free words were derived and utilized to modify search algorithms to generate such words. Based on theory and an empirical analysis, a conjectured tight upper bound for the length of the longest maximal bbox repetition-free word was established as $B(n) = n(2 + B(n - 1))$, for an alphabet size of n (see Theorem 1 and Conjecture 1).

The theoretical properties of MRF words made it clear that the associated search space could be represented as either a tree (see Section 4.1 and Section 4.2) or a graph (see Section 4.3). DFS can then be utilized to traverse the search spaces. In both cases the search space becomes intractable and it becomes impossible to find LMRF words for large n (see Table 2).

Construction of a unique graph G_n, which can be traversed using a modified DFS, allowed for finding LMRF words for larger values of n. The vertices of G_n can be grouped into sets of vertices V_k, depending on the length of the bbox that a vertex is associated with. The larger k becomes, the smaller the indegree and the outdegree of vertices belonging to V_k are. When DFS is executed, the vertices with the lowest indegree and outdegree are explored first. This maximizes the number of unexplored edges that remain within the graph while the search progresses. This strategy (called reverse node search) makes it possible to find LMRF words up to $n = 6$ (see Table 2). To summarise, the graph-based search methods, based on the LMRF assumptions, outperformed backtracking and MCTS.

Future work includes a theoretical proof for the upper bound on the length of an LMRF, and determining the number of LMRF words for each n.

Acknowledgements

This work is based on research supported in part by the National Research Foundation of South Africa (grant number: 111732).

References

[Annexstein 1997] Annexstein, F.S.: "Generating De Bruijn sequences: an efficient implementation"; IEEE Trans. on Computers 46, 2 (1997), 198-200.

[Baena-Garcia et al. 2010] Baena-Garcia, M., Morales-Bueno, R., Carmona-Cejudo, J., Castillo, G.: "Counting word avoiding factors"; Electronic J. of Math. and Tech. 4, 3 (2010), 251.

[Bean et al. 1979] Bean, D., Ehrenfeucht, A., McNulty G.: "Avoidable patterns in strings of symbols"; Pacific J. Math. 85, (1979), 261-294.

[Berstel and Karhumäki 2003] Berstel, J. and Karhumäki, J.: "Combinatorics on words: a tutorial"; Bull. of the EATCS 79, (2003), 178-228.

[Berstel and Perrin 2007] Berstel, J., Perrin, D.: "The origins of combinatorics on words"; European J. of Combinatorics. 28, 3 (2007), 996-1022.

[Browne et al. 2012] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S. Perez Liebana, D., Samothrakis, S., Colton, S.: "A survey of Monte Carlo tree search methods"; IEEE Trans. on Comp. Intelligence and AI in Games 4, 1 (2012), 1-43.

[Carpi and De Luca 2001] Carpi, A., De Luca, A.: "Words and special factors"; Theor. Comp. Sci. 259 (2001), 145-182.

[Charalambides 2002] Charalambides, C.A.: "Enumerative Combinatorics"; CRC Press, Boca Raton, USA.

[Diestel 2017] Diestel, R.: "Graph Theory"; Springer, Berlin.

[Habeck 2022] Habeck, M.: "The Generation of Maximum Length Box Repetition-free Words"; MSc thesis, 2002, Stellenbosch University, Stellenbosch, South Africa.

[Lothaire 1997] Lothaire, M.: "Combinatorics on Words"; Encyclopedia of Mathematics and its Applications, Vol. 17, Cambridge University Press.

[Martin 1934] Martin, M.: "A problem in arrangements"; Bull. of the American Math. Soc. 40, 12 (1934), 859-864.

[McConnell 2001] McConnell, J.J.: "Analysis of Algorithms"; Jones and Bartlett, Massachusetts, USA.

[Nijssen 2013] Nijssen, P.: “Monte-Carlo Tree Search for Multi-Player Games”; Ph.D. thesis, University of Maastricht, The Netherlands.

[Pearl 1982] Pearl, J.: “The solution for the branching factor of the alpha-beta pruning algorithm and its optimality”; Comm. of the ACM 25, 8 (Aug 1982), 559–564.

[Ruskey 2003] Ruskey, F.: “Combinatorial Generation”; University of Victoria, Canada.

[Seeja 2018] Seeja, K.: “HybridHAM: a novel hybrid heuristic for finding Hamiltonian cycle”; J. of Optimization (2018), 1-10.

[Stanley 1986] Stanley, R.P.: “What is enumerative combinatorics?”; In: Enumerative Combinatorics, Springer, Berlin, 1-63.

[Thue 1906] Thue, A.: “Über unendliche Zeichenreihen”; Norske Vid. Selsk. Skr. I Math-Nat. Kl. Chris. 7, (1906), 1-22.

[Thue 1912] Thue, A.: “Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen”; Norske Vid. Selsk. Skr. I Math-Nat. Kl. Chris. 1, (1912), 1–67.

[Zimin 1982] Zimin, A.: “Blocking sets of terms”; Mat. Sb. 119, 3 (1982), 363–375.