Determining Pedan Lurik Weaving Motif Variations using Recursive Functions and Search Trees with Time Complexity Analysis

Natan Danuarta Ariel Wicaksana - 13525143
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: arielwicaksana25@gmail.com , 13525143@std.stei.itb.ac.id

***Abstract*— Combinatorics offers a mathematical method to calculate and structure traditional textile patterns. This paper aims to determine and generate the vast motif variations of Pedan Lurik weaving from Klaten, Indonesia. We utilize recursive algorithms to formulate the thread combinations, visualize the execution process through search trees, and measure the computational efficiency using asymptotic time complexity analysis. Our findings demonstrate a clear computational trade-off, as the algorithm must explore the entire search space to map all possible weaving patterns.**

Keywords—Combinatorics, Pedan Lurik, Recursive algorithm, Search tree, Time complexity

I. INTRODUCTION

Combinatorics is an essential branch of discrete mathematics that focuses on counting, arranging, and structuring objects. One of its primary functions is to calculate the total number of possible arrangements within a given set without the need for exhaustive manual enumeration. This mathematical framework provides powerful tools to model complex discrete structures and solve various combinatorial problems. In the context of computational design, combinatorics enables researchers to systematically quantify configurations that arise from multiple interacting variables, forming the theoretical foundation for algorithmic generation.

A compelling real-world application of this concept can be observed in the creation of traditional textiles, specifically the Pedan Lurik weaving from Klaten, Indonesia. This fabric is characterized by its distinct geometric motifs, which are structurally formed through the perpendicular intersection of threads. The anatomy of the woven cloth consists of warp yarns running vertically and weft yarns running horizontally across the loom. By assigning different color palettes to these intersecting elements, weavers can produce an extensive array of intricate visual patterns.

Fig. 1.1 Structural anatomy of Pedan Lurik weaving
(Source: https://lurikprasojo.id/warna-lurik-cocok-untuk-outfit-resmi/)

As the complexity of the textile design increases, a computational approach becomes highly necessary. When the number of thread strands and the available color selections grow, the resulting motif combinations expand at a rapid pace. Exploring and mapping every single pattern variation manually is practically impossible and highly susceptible to human error. Without an automated mechanism to systematically compute these permutations, documenting the complete spectrum of the traditional weaving designs becomes an insurmountable task.

Therefore, this paper aims to address this challenge by computationally modeling the Pedan Lurik motif variations. We propose an algorithm based on recursive functions to systematically generate all possible color and thread configurations. Furthermore, this study will visualize the algorithmic execution and traversal using a search tree structure to provide a clear understanding of the generative process. Finally, we evaluate the computational efficiency of our proposed method through asymptotic time complexity analysis, ensuring a comprehensive assessment of the algorithm's performance as the input size scales.

II. THEORETICAL FOUNDATION

A. Combinatorial Mathematics

Let $A_1, A_2, \ldots, A_n$ be finite sets. The cardinality of their Cartesian product is mathematically formulated as:

$$|A_1 \times A_2 \times \ldots \times A_n| = \prod_{i=1}^{n} |A_i| = |A_1| \cdot |A_2| \cdot \ldots \cdot |A_n|$$

This equation formalizes the combinatorial Rule of Product. It states that if a sequential process consists of n independent events, and each i-th event can result in $|A_i|$ possible outcomes, the total number of distinct combinations for the entire process is the product of their individual cardinalities. This mathematical model is utilized to calculate the exact search space of the Pedan Lurik weaving motifs. Assuming the fabric configuration consists of n thread segments (representing both warp and weft) and there are m available color choices for each specific segment, the total number of possible motif variations equals exactly m^n. This theoretical formulation provides a quantitative baseline before initiating the algorithmic generation process.

B. Recursive Functions

A recursive function $f(n)$ can be mathematically modeled as a piecewise formulation consisting of a base case and a recurrence relation:

$$f(n) = \begin{cases} c, & \text{if } n = 0 \\ g\big(n, f(n-1)\big), & \text{if } n > 0 \end{cases}$$

In this mathematical construct, c represents a constant return value for the base condition, which effectively halts the execution to prevent infinite loops. Conversely, the function $g\big(n, f(n-1)\big)$ dictates the recurrence step, wherein the operation invokes itself with a strictly smaller argument until the base case is reached. The procedural generation of the weaving patterns will be computationally executed using this exact recurrence model. The program will construct the fabric motif incrementally. When the remaining sequence length $n > 0$, the algorithm assigns a specific color and recursively calls itself to process the subsequent thread. Once the full length is generated ($n = 0$), the base case terminates that particular branch and yields one completely formed thread combination.

C. Tree Data Structures and m-ary Trees

Let $G = (V, E)$ be a directed graph where V denotes a set of vertices and E signifies a set of edges. G is formally defined as a rooted tree if it is a connected, acyclic graph containing a uniquely distinguished root vertex. Furthermore, a full m-ary tree satisfies a specific mathematical property relating its internal vertices k and the total number of vertices v:

$$v = m \cdot k + 1$$

This equation dictates that in a fully structured m-ary tree, every internal node must branch into exactly m children, establishing a strict hierarchical relationship from the root down to the terminal leaves. The algorithmic execution of the aforementioned recursive function can be directly visualized as an m-ary search tree. Within this structural representation, each internal branch node symbolizes a computational decision point corresponding to the m available thread color selections. Consequently, the directed paths traversing from the root to the leaves denote the generative sequences, and the terminal leaves correspond directly to the final, synthesized textile motifs.

D. Asymptotic Algorithm Complexity

Let $T(n)$ and $f(n)$ be functions mapping positive integers to positive real numbers. The time complexity $T(n)$ is defined to be in $O\big(f(n)\big)$ if there exist positive constants C and n_0 such that:

$$T(n) \leq C \cdot f(n), \quad \forall n \geq n_0$$

This formal definition establishes the asymptotic upper bound of an algorithm's execution time as the input size n scales towards infinity, isolating the core growth rate by disregarding lower-order terms and constant coefficients. Because the proposed exhaustive search approach is designed to generate every single textile motif combination, the algorithm must inherently traverse the entire combinatorial search space. Therefore, this asymptotic evaluation will be employed to mathematically quantify the efficiency of the generative algorithm. Due to the brute-force nature of the search, the algorithm is theoretically expected to yield an exponential time complexity bound, mapping directly to $O(m^n)$.

III. IMPLEMENTATION AND ANALYSIS

A. Combinatorial Analysis of the Weaving Motif

Before implementing the algorithmic generator, it is crucial to analytically quantify the combinatorial search space of the textile patterns. Let n denote the total number of individual thread strands constructing a specific fabric segment, and let m represent the total number of available base colors (e.g., black, white, and brown) in the weaver's palette. Because the color of each thread strand is chosen independently, the total number of valid motif configurations can be formulated by applying the mathematical Rule of Product. The total volume of variations, denoted as V, is expressed through the following equation:

$$V = m^n$$

To illustrate the magnitude of this search space, consider a simplified textile arrangement consisting of exactly 10 thread strands that utilize an $m = 3$ color palette. The total number of valid permutations would be calculated as $3^{10} = 59{,}049$ unique weaving motifs. As the length of the thread sequence grows linearly, the size of the corresponding search space expands exponentially. Table I demonstrates this rapid computational growth, holding the color palette constant at $m = 3$ while progressively increasing the thread strand count.

TABLE I
EXPONENTIAL GROWTH OF THE COMBINATORIAL SEARCH SPACE

Thread Strands (n)	Color Palette (m)	Total Motif Variations ($V = m^n$)
2	3	9
5	3	243
10	3	59.049
15	3	14.348.907
20	3	3.486.784.401

This mathematical formulation proves that an exhaustive enumeration of large textile motifs is highly resource-intensive. Consequently, the succeeding algorithmic design must be carefully structured to systematically handle the expansive state space without logic errors.

B. *Recursive Algorithm Design*

To navigate the previously modeled search space, we employ an exhaustive brute-force search strategy. The core objective of this approach is to systematically generate every single viable color combination and document the complete set into an array data structure. The algorithm achieves this through a recursive mechanism, sequentially traversing each thread index and branching out for every available color option until a full motif is successfully constructed. The formal logic of this generative process is documented in Algorithm 1.

Algorithm 1 Recursive Generation of Lurik Motifs

```
1:  procedure generate_motif(current_thread_index, n, colors, current_motif)
2:      if current_thread_index > n then
3:          save_to_list(current_motif)
4:      else
5:          for each c in colors do
6:              current_motif[current_thread_index] <- c
7:              generate_motif(current_thread_index + 1, n, colors, current_motif)
8:          end for
9:      end if
10: end procedure
```

The generate_motif procedure takes four parameters: the positional index of the thread currently being evaluated, the sequence boundary n, the set of available color choices, and the active array storing the partially built motif. The execution relies on a strict control flow governed by a base case and a recurrence step.

The conditional statement in Line 2 establishes the base case. It verifies whether the current_thread_index has exceeded the predefined length limit n. If this condition evaluates to true, it indicates that all thread segments have been properly assigned a color. The completely formed configuration is then appended to a global collection on Line 3, effectively terminating that specific branch of execution.

Conversely, if the pattern remains incomplete, the algorithm proceeds to the recurrence block starting at Line 4. The procedure iterates through every single color element within the available palette (Line 5). During each iteration, a selected color c is assigned to the current array position (Line 6). Immediately following this assignment, the function recursively invokes itself (Line 7), incrementing the current_thread_index by 1 to dictate the transition to the subsequent thread. This continuous cycle of assignment and self-invocation guarantees that the algorithm deeply explores every mathematical possibility, successfully generating the entire m^n collection of Pedan Lurik weaving motifs.

C. *Search Tree Visualization*

The algorithmic execution of generate_motif can be directly mapped to an m-ary search tree structure. In this structural representation, the root node symbolizes the initial state where the thread motif array is entirely empty. Every internal node denotes a partial state of the motif configuration, while the directed branches emanating from these nodes represent the computational decision of selecting a specific thread color. Because the algorithm evaluates m distinct color options at every step, each internal vertex branches into exactly m children. The hierarchical depth of the tree correlates directly to the total number of thread strands n. Consequently, the terminal leaves positioned at depth n represent the fully synthesized pattern arrays.

To clearly illustrate this behavior, consider a scaled-down generative instance characterized by $n = 3$ threads and a binary color palette consisting of $m = 2$ options (Color A and Color B). The recursive traversal forms a complete binary tree, visually constructed as follows:

This traversal confirms that an exhaustive generation process mandates navigating every possible path from the root. The algorithm continuously descends into the tree depth, backtracks upon reaching a leaf, and explores unvisited subtrees until all m^n terminal vertices are evaluated and recorded.

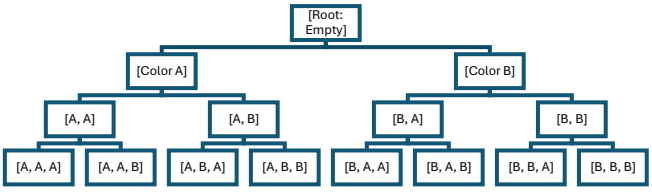

Fig. 3.1 Binary search tree of the generate_motif

D. *Asymptotic Time Complexity Analysis*

To evaluate the computational efficiency of the recursive pattern generator, we formulate its recurrence relation

mathematically. Let $T(n)$ represent the time required to compute the remaining combinations for a motif of length n, and let m denote the constant number of available colors. Assuming the assignment operation and the base case evaluation require a constant cost $O(1)$, denoted algebraically as c, the recurrence relation is defined as:

$$T(n) = m \cdot T(n-1) + c$$

$$T(0) = c$$

To isolate the growth function, we utilize the iterative substitution method. By continuously substituting the equation into itself, we extract a geometric progression model:

$$T(n) = m(m \cdot T(n-2) + c) + c = m^2 T(n-2) + mc + c$$

$$T(n) = m^3 T(n-3) + m^2 c + mc + c$$

Generalizing this pattern for k iterations yields:

$$T(n) = m^k T(n-k) + c \sum_{i=0}^{k-1} m^i$$

The recursion halts when it reaches the base case, meaning$n - k = 0$, or $k = n$. Substituting k with n provides the complete sequence formula. By utilizing the standard sum of a geometric series, the equation transforms into:

$$T(n) = m^n T(0) + c\left(\frac{m^n - 1}{m - 1}\right)$$

Because $T(0) = c$ and $m \geq 2$, the term m^n dictates the highest order of growth. Consequently, by dropping the lower-order terms and constant coefficients, the asymptotic upper bound is strictly proven to be:

$$T(n) \in O(m^n)$$

This derivation proves that the algorithm exhibits exponential time complexity. The brute-force mechanism enforces an exhaustive search over every single node within the generated m-ary tree. As the motif length n increases, the total volume of recursive calls scales exponentially. To validate this theoretical derivation empirically, Table II compares the input size against the exact number of recursive function calls evaluated during program execution, utilizing a constant $m = 3$ palette.

TABLE II
EMPIRICAL COMPUTATION OF RECURSIVE FUNCTION CALLS

Input Size (n)	**Theoretical Search Space (m^n)**	**Total Recursive Calls ($\frac{m^{n+1}-1}{m-1}$)**
2	9	13
5	243	364
10	59.049	88.573
15	14.348.907	21.523.360

The empirical results confirm the mathematical bounds; the recursive operations invariably outnumber the final generated permutations due to the overhead of exploring intermediate tree nodes. This dynamic underscores the critical limitation of exhaustive combinatorial searches when applied to large-scale textile matrices.

E. C Implementation of the Recursive Generator

To bridge the gap between theoretical algorithm design and practical system execution, we translate the aforementioned pseudocode into a concrete C language implementation. This low-level programming approach allows for direct memory manipulation and precise performance measurement. The complete recursive formulation, including timing mechanisms, is documented in the source code below.

Code Snippet 1: C Implementation of generate_motif

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void save_to_list(int* motif, int n) {
}
void generate_motif(int current_thread_index, int n, int*
colors, int m, int* current_motif) {
    if (current_thread_index == n) {
        save_to_list(current_motif, n);
    }
    else {
        for (int i = 0; i < m; i++) {
            current_motif[current_thread_index] = colors[i];
             generate_motif(current_thread_index + 1, n, colors,
m, current_motif);
        }
    }
}

int main() {
    int n = 15;
    int m = 3;
    int colors[] = {1, 2, 3};

    int* current_motif = (int*)malloc(n * sizeof(int));
    if (current_motif == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }
    clock_t start_time = clock();
    generate_motif(0, n, colors, m, current_motif);
    clock_t end_time = clock();
     double time_spent = ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000.0;
```

```
    printf("Successfully evaluated %d thread variations.\n",
n);
    printf("Total Execution Time: %f ms.\n", time_spent);
    free(current_motif);
    return 0;
}
```

The C implementation utilizes a dynamically allocated array, current_motif, instantiated via the malloc function within the main routine. This contiguous memory block acts as the singular state container maintaining the active weaving sequence across all recursive calls. By passing the pointer of this array recursively, the algorithm circumvents the severe computational overhead of instantiating redundant memory copies for each execution branch. Consequently, the space complexity remains strictly bounded to $O(n)$. To accurately assess the raw computational cost of the algorithm, the standard output functions within the base case are intentionally suppressed. Printing outputs to a terminal introduces a significant input/output bottleneck, which would distort the empirical measurement of the CPU execution time recorded by the <time.h> library.

During execution, the generate_motif function relies on the current_thread_index parameter to track its current depth within the sequence. At each recursive level, the procedure iterates through the available colors array. It mutates the state by assigning the selected color to the corresponding array index and immediately invokes itself with an incremented index position (current_thread_index + 1). This recurrence halts exclusively when the index variable satisfies the threshold n, triggering the base case. At this terminal stage, a fully synthesized thread combination resides within the array buffer, which is conceptually recorded before the function terminates and returns control to the previous caller.

From a structural perspective, this programmatic behavior intrinsically executes a Depth-First Search (DFS) traversal upon the m-ary tree model established in the previous section. As the function continuously calls itself, it descends aggressively along a specific branch until reaching a leaf node. Upon returning from the base case, the control flow backtracks to the immediate parent node. The iteration then advances to the subsequent color parameter, overwrites the value at current_thread_index, and initiates a descent down a newly formed branch. This continuous cycle of array mutation and recursive backtracking ensures that the computational system systematically enumerates the entire m^n search space without duplication or omission.

IV. METHODOLOGY

To empirically evaluate the performance of the proposed recursive algorithm, a controlled computational experiment was designed and executed. The brute-force pattern generator was implemented utilizing standard pure C and compiled via the GNU Compiler Collection (GCC). To measure the computational cost, the real execution time, representing the exact CPU processing duration devoid of input/output overhead, was calculated using the standard <time.h> library, with outputs formatted in milliseconds (ms). The benchmarking procedure was conducted systematically by incrementally scaling the input size, specifically the number of thread strands (n), to precisely observe the growth rate of the execution time. All experimental executions were performed on a local machine with the following hardware specifications:

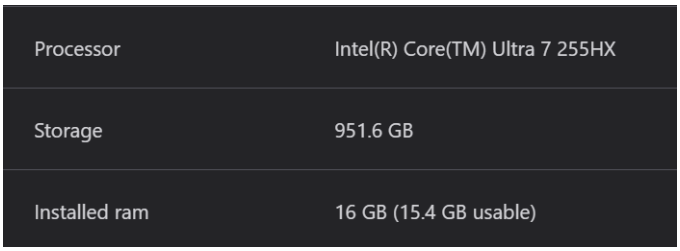

Fig. 4.1 Device specifications

V. EXPERIMENTAL RESULT AND DISCUSSIONS

The empirical data extracted from the algorithmic benchmarking process is documented in Table III. The table compares the incrementing input size against both the theoretical search space volume ($m = 3$) and the actual recorded CPU execution time.

TABLE III
EMPIRICAL PERFORMANCE OF THE RECURSIVE GENERATOR ($m = 3$)

Number of Threads (n)	**Theoretical Search Space (m^n)**	**Real Execution Time (ms)**
5	243	< 1
10	59.049	1
15	14.348.907	99
20	3.486.784.401	15.401

Analyzing the tabulated results reveals a profound demonstration of computational scaling. For constrained input sizes such as $n = 5$ and $n = 10$, the algorithmic execution is virtually instantaneous, completing the combinatorial enumeration in 1 ms or less. At this stage, the processor easily accommodates the relatively small state spaces. However, as the sequence length extends to $n = 15$, the processing time increases to 99 ms to successfully traverse over 14 million unique motif configurations.

The most critical analytical observation occurs at $n = 20$. Despite appending merely five additional thread units to the parameter, the execution time undergoes a dramatic surge, reaching exactly 15,401 ms (approximately 15.4 seconds). This steep escalation accurately confirms the theoretical $O(m^n)$ asymptotic upper bound established mathematically in Chapter III. The empirical timing data perfectly mirrors the characteristic trajectory of an exponential growth curve. Consequently, this proves that while the exhaustive brute-force

method is fundamentally necessary to map the entirety of the Pedan Lurik motif variations, it inherently places an immense processing burden on the CPU, severely restricting its practical execution speed for extensive textile matrices.

VI. CONCLUSION

This paper has successfully modeled and generated the motif variations of traditional Pedan Lurik weaving by applying the foundational principles of combinatorial mathematics and algorithmic design. Utilizing the combinatorial Rule of Product, we analytically established that the expansive search space of the textile patterns strictly scales at a volume of $V = m^n$. To systematically navigate this domain, a recursive algorithm alongside an m-ary search tree structure was developed, successfully generating and visualizing every viable combination of thread colors.

The primary conclusion of this research lies in the computational evaluation of this generative process. The theoretical time complexity analysis mathematically proved that the brute-force enumeration operates within an asymptotic upper bound of $O(m^n)$. This conclusion was definitively validated through empirical experiments using a pure C program, wherein the execution time experienced an exponential surge, exceeding 15 seconds merely to process an input size of $n = 20$. This dramatic performance degradation emphasizes the absolute, unavoidable consequence of employing an exhaustive search algorithm, which strictly mandates the total enumeration of the combinatorial space to document the intricate cultural designs without omission.

APPENDIX

The complete pure C source code utilized for the recursive motif generator, alongside the raw empirical data from the computational experiments, is publicly accessible. Interested readers can review the implementation details and benchmark scripts via our Google Drive folder here: https://drive.google.com/file/d/16OaJZ7JwRFG69e8Ji7Z7qdbu-kfdeao3/view?usp=sharing

ACKNOWLEDGEMENT

The author would like to express profound gratitude to God Almighty for the continuous guidance and blessings throughout the completion of this paper. Special appreciation is extended to Dr. Ir. Rinaldi Munir, M.T., the lecturer of the IF1220 Discrete Mathematics course, for his invaluable teachings, insightful feedback, and academic support.

REFERENCES

[1] Douglas B. West, "Combinatorial Mathematics". Cambridge University Press: Cambridge (United Kingdom). [Online]. Available: Combinatorial Mathematics - Douglas B. West - Google Buku. [Accessed: Jun 9, 2026].

[2] Peter Sanders and Rudolf Fleischer, "Asymptotic Complexity from Experiments? A Case Study for Randomized Algorithms," [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-44691-5_12 [Accessed: May 12, 2026].

[3] Sinta Agustina, "*Tenun Lurik Pedan di Klaten Punya Desain Unik, Dibuat Secara Manual untuk Kualitas Terbaik*," [Online]. Available: https://travel.tribunnews.com/2025/03/25/tenun-lurik-pedan-di-klaten-punya-desain-unik-dibuat-secara-manual-untuk-kualitas-terbaik / [Accessed: Jun 15, 2026].

[4] Emiliana Sadilah. 2009. "KERAJINAN TENUN LURIK PEDAN DI KLATEN." Available: https://d1wqtxts1xzle7.cloudfront.net/96516652/227150527-libre.pdf?1672302363=&response-content-disposition=inline%3B+filename%3DJantra_jurnal_sejarah_dan_budaya_Vol_IV.pdf&Expires=1781881222&Signature=IArJn-hi5NY3uXX94mFMcFuLHKl3N7DfmicjXDufdWSvgnfuwzjzZsVfXfRztA9mP5jWbnqUFrcXPSnr7MUzJKA8Xdjcn3wrob4~FxR8EBwga6alsEyogVbHIqBK1iP2T5ipWD5YpiBmuCW-6GrcKa-9Mzqw8lgeq0lvxZNhx2lygop45VQogV~7EXbgyNOIEvjX~Nd4kAGyo37X9xh7ksG4SGE7FeTyoYi9PSgo44FjHhuNBF6o8Cz5xSl0TbKX8TRT27HTDqyYgQq-o41iZDPWWlEWJTZUvFD5hFGJh9mxV1cM7uLuULaP-xY~TnnMRHvu1ssuZxsZfGTQQdUug__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA#page=42 . [Accessed: Jun 18, 2026]

[5] R. Munir, " Barisan, sumasi, rekursi, dan relasi rekurens (Bagian 1)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/13-Barisan,%20rekursi-dan-relasi-rekurens-(Bagian1)-2026.pdf . [Accessed: Jun 19, 2026].

STATEMENT

Hereby, I declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not a product of plagiarism.

Bandung, 19 June 2026

Natan Danuarta Ariel Wicaksana - 13525143