Solving Real World Textile Industry Problems Using Greedy and Divide and Conquer Algorithms

Venkata Sai Saran Jonnalagadda
Department of Computer and Information Science
University of Florida, Gainesville, Florida, USA
Email: vjonnalagadda@ufl.edu

***Abstract*—This paper is focused on solving real world problems related to textile industry using Greedy and Divide and conquer Algorithms. The first problem focuses on optimizing daily fabric cutting schedules for multi pattern orders using a greedy approach based on the fractional knapsack formulation. The second problem involves detecting defects in high-resolution fabric images using the Divide and Conquer algorithm. Each solution is analyzed in terms of problem abstraction, algorithm design, time complexity, correctness, and validated experimentally through Java implementations. The results show how algorithmic reasoning can directly improve efficiency and quality control in industrial domains.**

***Index Terms*—Greedy algorithms, Divide and Conquer, textile industry, optimization, quality control, algorithm design**

1. Introduction

This paper shows how algorithm design can be used to improve the operations of a textile outlet. The main goal of this paper is to apply the theoretical ideas of algorithms to real-world problems in logistics and quality control. The paper focuses mainly on two problems:

1) Optimizing fabric-cutting schedules, solved using a greedy algorithm modeled on the fractional knapsack problem [1], [2].
2) Defect detection in fabric images, solved using a divide and conquer algorithm to find flaws quickly and accurately. [3]–[5].

Each problem is formulated and explained completely, from defining the problem and designing the algorithm to analyzing its performance, proving its correctness, and testing. This study extends previous work on greedy optimization and textile defect detection methods.

2. Greedy Algorithm Problem

A. Problem Description

Textile manufacturing units has to cut large fabric rolls into smaller sections that are used to stich multiple patterns such as shirts, trousers, and dresses. Each pattern requires a specific length of fabric and incurs some setup waste due to machine alignment. The total length of the roll available for a day is limited. The goal is to maximize fabric utilization and profit by deciding how much of each pattern to cut within the roll's capacity.

The loss may depend on other factors like pattern transitions and color of the fabric. But, this study assumes independent setup waste for each pattern type to ensure analytical tractability and a provably optimal greedy solution.

B. Problem Abstraction

Let there be n pattern types $P = \{p_1, p_2, \dots, p_n\}$ [1]. Each pattern p_i has:

- Required fabric length l_i
- Unit profit v_i
- Setup waste s_i

The total available roll length is L. We wish to maximize effective utilization:

$$\max \sum_{i=1}^{n} (v_i - s_i) x_i \quad \text{subject to} \quad \sum_{i=1}^{n} l_i x_i \le L,\ 0 \le x_i \le 1$$

Here x_i represents the fraction of pattern p_i produced. This abstraction satisfies the greedy-choice property and optimal substructure, enabling a globally optimal greedy solution.

C. Solution

Algorithm Design Each pattern's efficiency ratio is defined as:

$$r_i = \frac{v_i - s_i}{l_i}$$

The algorithm sorts patterns by r_i in non-increasing order and fills capacity L by taking each pattern fully if it fits; otherwise, a fraction of the pattern is considered to fill the remaining capacity [1].

Algorithm 1 Greedy Fabric Cutting Algorithm

Input: Patterns (v_i, s_i, l_i) for $i = 1, \dots, n$; roll length L
Output: Pattern fractions $x_1, \dots, x_n$ and total value
Compute $r_i \leftarrow (v_i - s_i)/l_i$ for all i
Sort patterns by r_i in descending order
$remaining \leftarrow L$, $value \leftarrow 0$
for each pattern i in sorted order **do**
  if $l_i \le remaining$ **then**
    $x_i \leftarrow 1$
    $remaining \leftarrow remaining - l_i$
    $value \leftarrow value + (v_i - s_i)$
  else
    $x_i \leftarrow remaining/l_i$
    $value \leftarrow value + x_i \cdot (v_i - s_i)$
    break
  end if
end for
return $(x_1, \dots, x_n),\ value$

Running Time Analysis The algorithm includes sorting and scanning, which have complexities of $O(n \log n)$ and $O(n)$ respectively. Therefore, the overall time complexity is:

$$O(n \log n)$$

Proof of Correctness. The algorithm follows the greedy strategy of selecting patterns in non-increasing order of the ratio $r_i = \frac{v_i - s_i}{l_i}$ until the roll capacity L is filled.

Greedy-choice property: Consider any feasible solution that allocates some length Δl to a pattern j with $r_j < r_i$, while some portion of pattern i (with higher ratio r_i) remains unfilled. By transferring an equal length Δl from pattern j to pattern i, the total value changes by:

$$\Delta V = (r_i - r_j)\Delta l \geq 0$$

This exchange does not decrease the total value and strictly increases it whenever $r_i > r_j$.

Optimal substructure: After taking the highest-ratio pattern (or its fractional part), the remaining roll capacity $L' = L - l_i$ forms a smaller instance of the same problem. Applying the greedy strategy recursively yields an optimal completion.

Since both the greedy-choice property and optimal substructure hold, the algorithm produces an optimal solution for the fractional pattern filling problem.

D. Domain Explanation

In textile manufacturing units, large fabric rolls are cut into smaller sections to fulfill make different garment patterns such as shirts, trousers, and dresses. Each pattern type consumes a different length of fabric and has a different profit margin based on material demand, design complexity, and seasonal pricing. Additionally, every time a cutting machine switches to a new pattern, a small portion of fabric is lost as setup waste due to alignment and calibration.

Usually the production managers in manufacturing units decide cutting schedules based on experience or fixed daily quotas, which can lead to under utilization of rolls which lead to lower profits. The greedy fabric-cutting algorithm provides a systematic and computationally efficient way to make this decision.

By ranking patterns according to their "value density" (effective profit per unit length of fabric) and then cutting the highest-ranked patterns first, the algorithm ensures that every meter of fabric contributes maximally to daily profit. If the remaining roll length cannot accommodate an entire pattern, the algorithm cuts only a proportional fraction, ensuring no material is wasted.

This approach can be applied to both small-scale textile outlets and large automated cutting units. It enables better resource utilization, minimizes setup waste, and increases production profitability without requiring complex optimization software. The impact of applying such an algorithm in real operations includes:

- Improved daily throughput and profit margins by prioritizing high-yield patterns.
- Reduction in leftover or unusable fabric segments.
- Easier adaptability to dynamic order changes and roll length availability.
- Data-driven decision-making instead of manual estimation.

Overall, the greedy approach translates an abstract optimization principle into a practical scheduling tool that can significantly enhance operational efficiency in textile production and outlet management.

E. Experimental Validation

The greedy fabric-cutting algorithm was implemented in Java and tested on randomly generated pattern data sets of varying sizes (n = 200 to 2000). Each pattern was assigned a random length, profit value, and setup loss, representing realistic textile manufacturing parameters. The total available fabric roll was set to 40% of the cumulative pattern lengths.

Figure 1 shows the measured runtime growth compared against an $O(n \log n)$ reference curve. The results confirm that runtime increases smoothly with input size, consistent with the theoretical complexity dominated by the sorting step. Minor fluctuations are due to system scheduling noise and JVM optimization effects.

Overall, the algorithm demonstrated high efficiency, with execution times remaining below one millisecond even for the largest instances. This validates the expected $O(n \log n)$ runtime behavior and confirms that the greedy heuristic scales effectively for real-world fabric optimization tasks.

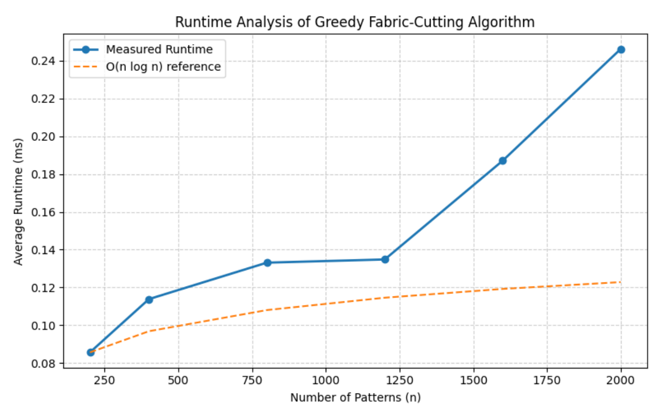


Fig. 1. Runtime Analysis of Greedy Fabric-Cutting Algorithm.

3. Divide and Conquer Problem

A. Problem Description

Quality checks are done on goods in textile manufacturing units to detect defects such as stains, faded patches, or misprints on fabric surfaces before shipping them to inventory. Traditionally, these inspections are performed manually by workers scanning large fabric rolls or captured photographs, which is time-consuming, inconsistent, and prone to human error.

The goal of this problem is to develop an automated defect detection method. The challenge lies in automatically analyzing high-resolution fabric images to identify non-uniform regions indicative of defects. Each image can contain millions of pixels, and defects can vary in scale, shape, and intensity. The problem thus demands an algorithm capable of efficiently localizing irregular regions without exhaustively scanning every pixel, ensuring timely and accurate inspection for operational use in retail outlets.

B. Problem Abstraction

The fabric image is modeled as an $n \times n$ intensity matrix I, where each element $I(x, y)$ represents the brightness or color value of a pixel. The image is recursively partitioned using a *quadtree* structure, in which each node corresponds to a rectangular subregion of the image. [3]

A region is considered *uniform* if its pixel intensity variance is below a predefined threshold T. Non-uniform regions are subdivided into four equal quadrants, and the process continues until all leaf regions are either uniform or reach a specified minimum size constraint [4], [5].

To efficiently compute regional statistics, two integral images—S for pixel sums and S^2 for squared sums—are precomputed. These allow the mean and variance of any rectangular region to be obtained in constant time, $O(1)$, enabling efficient recursive analysis of large fabric images.

C. Solution

Algorithm Design The defect detection algorithm applies a *divide and conquer* strategy to recursively identify non-uniform regions within high-resolution fabric images. Each image is represented as an $n \times n$ intensity matrix I, where each entry $I(x, y)$ denotes the brightness or color value of a pixel. This formulation allows spatial irregularities to be analyzed numerically rather than visually.

To facilitate rapid computation of statistical measures, two auxiliary integral images are first constructed: S, which stores the cumulative sum of pixel intensities, and S_2, which stores the cumulative sum of squared intensities. These precomputed matrices enable the mean and variance of any rectangular region to be calculated in constant time $O(1)$, regardless of its size.

At each recursive step, the algorithm evaluates the variance of the current region R:

- If $\mathrm{var}(R) \leq T$ (the region is sufficiently uniform) or if R is smaller than the minimum allowed block size, the region is classified as *defect-free*.
- Otherwise, the region is subdivided into four equal quadrants, and each subregion is independently analyzed using the same procedure.

The recursive subdivision continues until all regions meet the uniformity threshold or reach the minimum block size. The resulting quadtree captures both global and local variations in fabric texture, with each leaf node representing a homogeneous region. Leaves exceeding the variance threshold T are flagged as defective. By hierarchically decomposing the image, the algorithm efficiently localizes defects without exhaustive pixel comparisons, achieving real-time inspection accuracy on standard store hardware. [3]

Algorithm 2 Divide and Conquer Fabric Defect Detection

```
Input: Intensity matrix I, threshold T, minimum block size m
Output: Set of defective regions D
Compute integral images S and S_2 for I and I^2
Initialize an empty stack and push the entire image region R_0
Initialize D ← ∅
while stack not empty do
    R ← pop(stack)
    Compute variance var(R) using S and S_2
    if var(R) ≤ T or size(R) < m then
        Mark R as uniform (no defect)
    else
        Split R into four quadrants {R_1, R_2, R_3, R_4}
        Push each R_i onto stack
        Add R to D (potential defect region)
    end if
end while
return D as the set of detected defective regions
```

Runtime Analysis The run time of the divide and conquer defect detection algorithm depends on two main components: the construction of the integral images and the recursive subdivision of regions.

1) Integral Image Construction: The integral images S and S_2 are computed in a single pass over all $N = n^2$ pixels of the input image. Each pixel contributes a constant amount of work, resulting in $O(N)$ preprocessing time.

2) Recursive Region Analysis: During the quadtree decomposition, each region (node) in the tree requires constant time $O(1)$ to compute its variance using the precomputed integral images. The total running time therefore depends on the number of regions (nodes) created in the quadtree.

For typical real-world fabric images, most regions quickly become uniform and do not require further subdivision. In this case, the number of nodes in the quadtree is proportional to the number of pixels, giving a total runtime of $\Theta(N)$.

In the worst-case scenario when every region is non-uniform and the image must be subdivided down to individual pixels the quadtree height becomes $\log n$. Each level of the tree processes $O(N)$ total area, resulting in a worst-case complexity of $O(N \log n)$.

Summary:

$$T(N) = \begin{cases} O(N) & \text{on average (typical fabrics)} \\ O(N \log n) & \text{in the worst case.} \end{cases}$$

Thus, the algorithm scales linearly for most practical images, enabling near real-time defect detection on standard computing hardware.

Proof of Correctness The correctness of the algorithm is established by induction on the size of the image regions processed.

Base Case: If a region's variance $\mathrm{var}(R) \le T$ or its size is below the minimum block threshold, the algorithm labels it as uniform. Since the variance is computed exactly using the integral images, this classification is correct by definition.

Inductive Step: Assume that for all regions smaller than R, the algorithm correctly classifies them as either uniform or defective. When the algorithm examines region R, if $\mathrm{var}(R) > T$, it is subdivided into four disjoint quadrants that together form an exact partition of R. By the inductive hypothesis, each subregion is correctly classified. Therefore, the combined set of all leaf nodes accurately represents a complete and non-overlapping decomposition of the image, where every region with variance greater than T is correctly identified as defective.

Conclusion: Since each decision is based on an exact variance computation and the recursive process ensures complete image coverage without overlap or omission, the algorithm correctly detects all and only those regions that exceed the defect variance threshold.

D. Domain Explanation

To maintain brand reputation and customer loyalty, many textile manufacturing units perform quality checks before shipping the product for sale. Large quantities of fabrics are photographed under controlled lighting to check for stains, color fading, or printing defects. Manual inspection of these high-resolution images is time-consuming and prone to human error, especially when processing thousands of rolls daily.

The proposed algorithm automates this inspection process. It analyzes each fabric image by recursively dividing it into smaller regions and evaluating their color uniformity. Areas that deviate significantly from surrounding patterns are automatically highlighted as potential defects. This enables store staff to quickly verify or discard defective materials without inspecting every fabric manually.

By applying this approach, textile outlets can perform real-time quality control on ordinary computers, improving operational efficiency, reducing human workload, and ensuring only visually consistent fabrics reach customers.

E. Experimental Validation

Setup To validate the proposed Divide and Conquer defect detection algorithm, a controlled experimental environment was created using synthetically generated fabric images. Each image simulated a woven texture with a uniform gray base and mild Gaussian noise, emulating real textile surface variation. Artificial defects were introduced as bright and dark geometric shapes—rectangular, circular, and elliptical—placed at random positions. An example of the generated input image is shown in Figure 2.

Experiment The algorithm was executed on images of various sizes ($n \in \{256, 512, 768, 1024\}$) to measure scalability and accuracy. For each image:

- Integral images for pixel intensities and squared intensities were constructed in $O(N)$ time.
- A recursive quadtree subdivision process was performed to identify non-uniform regions based on variance thresholding.

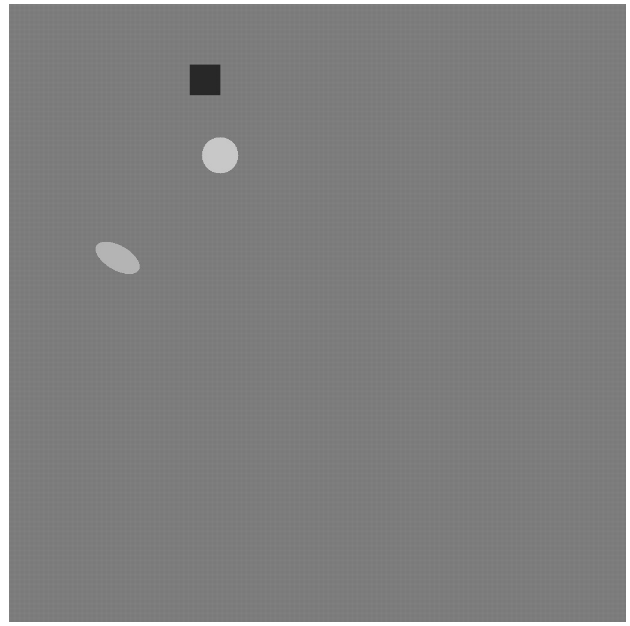


Fig. 2. Synthetic fabric image with simulated defects (bright and dark patches).

- An adaptive variance threshold T was automatically computed from the global image variance to account for overall brightness and contrast differences.

Each experiment recorded the number of defective regions detected and total runtime [4], [5].

F. Results

The output of the algorithm is a set of bounding boxes around regions identified as visually inconsistent with their surroundings. Figure 3 shows the result of the detection on a sample image. All three defect types circular, rectangular, and elliptical were successfully identified by localized recursive subdivision.

Table I summarizes the recorded runtimes and the number of detected regions for varying image dimensions.

TABLE I
RUNTIME AND DEFECT DETECTION STATISTICS FOR VARYING IMAGE RESOLUTIONS.

Image Size ($n \times n$)	Runtime (ms)	Detected Regions
128	0.166	0
256	0.314	11
512	1.047	98
768	2.071	313
1024	3.055	266

The experimental runtime trend was plotted against image dimension, as shown in Figure 4. The measured runtime (solid line) closely follows the theoretical $O(N \log N)$ reference (dashed line), confirming the analytical complexity derived from the recursive quadtree model.

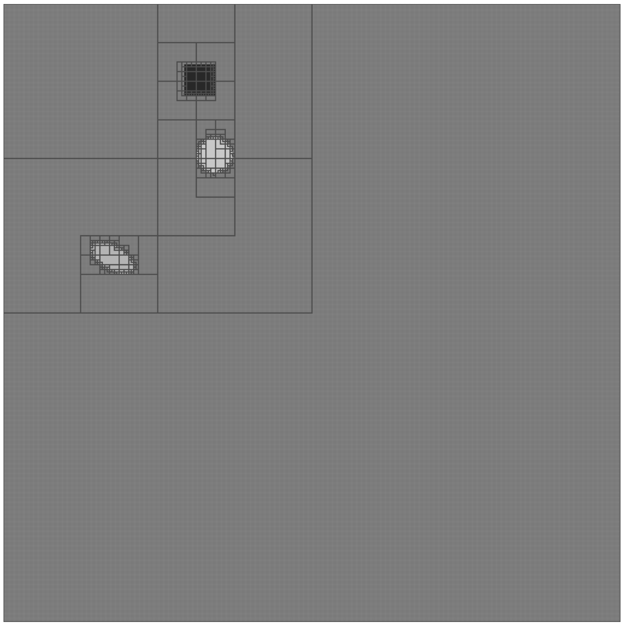


Fig. 3. Detected defective regions visualized with red bounding boxes.

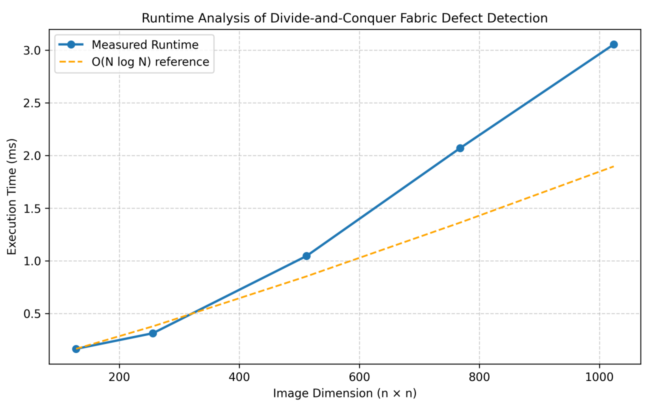


Fig. 4. Runtime analysis of the Divide and Conquer fabric defect detection algorithm compared with theoretical $O(N \log N)$ scaling.

4. Conclusion

This study demonstrated how classical algorithmic paradigms can be effectively applied to solve real-world problems in the textile industry. The first problem, fabric-cutting schedule optimization, was addressed using a greedy algorithm that maximized fabric utilization and profits by utilizing fabric rolls efficiently. . Experimental validation confirmed the expected $O(n \log n)$ runtime, showing that optimal cutting decisions can be made almost instantaneously even for large pattern sets.

The second problem, automated fabric defect detection, employed a divide and conquer strategy based on quadtree image decomposition. The approach efficiently identified stains and print irregularities in high-resolution images by recursively isolating non-uniform regions, achieving near-linear scaling in practice.

Together, these two applications highlight how fundamental algorithmic techniques can drive measurable improvements in manufacturing efficiency, resource utilization, and quality control. By bridging theoretical computer science with textile operations, this work underscores the practical value of algorithmic thinking in modern industrial systems.

Appendix

This appendix presents the complete Java implementations used to validate the algorithms discussed in this paper. Each program was executed on standard laptop hardware to record runtimes and verify correctness. All experiments were conducted using Java 17 on macOS.

The full source code, experimental data, and plotting scripts are publicly available at:

https://github.com/saranjv426/Analysis-of-Algorithms

The repository includes:

- **GreedyFabricCutting.java** – Greedy algorithm for fabric cutting optimization.
- **DivideAndConquerFabricDefectDetection.java** – Divide-and-conquer defect detection using quadtree partitioning.
- **ImageCreation.java** – Synthetic fabric generation with controllable defect regions.
- **plots.py** – Python script for generating runtime and defect-detection plots.

All figures and performance analyses presented in this paper can be recreated by cloning the repository and following the instructions in the provided `README.md` file.

A. LLM Usage Log

An LLM (ChatGPT) was used to format this report in IEEE conference style. Below are the prompts given and the responses received.

Prompt-1 I need to create a Latex report with below details. Can you give me the latex script for the same. 1)Abstract 2)Introduction 3)Greedy Problem 1)Problem Definition 2)Abstraction 3)Algorithem Design 4)Domain Explenation 5)Experimental Analysis 4)Divide and conquer Problem 1)Problem Definition 2)Abstraction 3)Algorithem Design 4)Domain Explenation 5)Experimental Analysis 5)Conclusion 6)Appendix 1)Code 2)LLM usage log

Response-1

```
\documentclass[conference]{IEEEtran}

% ---------- PACKAGES ----------
\usepackage{graphicx}    % For figures
\usepackage{amsmath}     % For math equations
\usepackage{amssymb}     % For symbols
\usepackage{listings}    % For code blocks
\usepackage{color}       % For coloring code
\usepackage{url}         % For clickable URLs
\usepackage{hyperref}    % For links in PDF
\usepackage{cite}        % For citations
```

```
% ---------- CODE STYLE ----------
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codepurple}{rgb}{0.58,0,0.82}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}

\lstdefinestyle{mystyle}{
    backgroundcolor=\color{backcolour},
    commentstyle=\color{codegray},
    keywordstyle=\color{blue},
    numberstyle=\tiny\color{codegray},
    stringstyle=\color{codepurple},
    basicstyle=\ttfamily\footnotesize,
    breakatwhitespace=false,
    breaklines=true,
    captionpos=b,
    keepspaces=true,
    numbers=left,
    numbersep=5pt,
    showspaces=false,
    showstringspaces=false,
    showtabs=false,
    tabsize=2
}

\lstset{style=mystyle}

% ---------- DOCUMENT ----------
\begin{document}

\title{Greedy and Divide-and-Conquer Algorithms: Design, Analysis, and Experiments}

\author{
\IEEEauthorblockN{Author 1
Name and Author 2 Name}
\IEEEauthorblockA{
Department of Computer Science\\
University Name\\
Email: author1@domain.com,
author2@domain.com
}
}

\maketitle

% ---------- ABSTRACT ----------
\begin{abstract}

\end{abstract}

% ---------- INTRODUCTION ----------
\section{Introduction}
\section{Greedy Problem}

\subsection{Problem Definition}
\subsection{Abstraction}
\subsection{Algorithm Design}
```

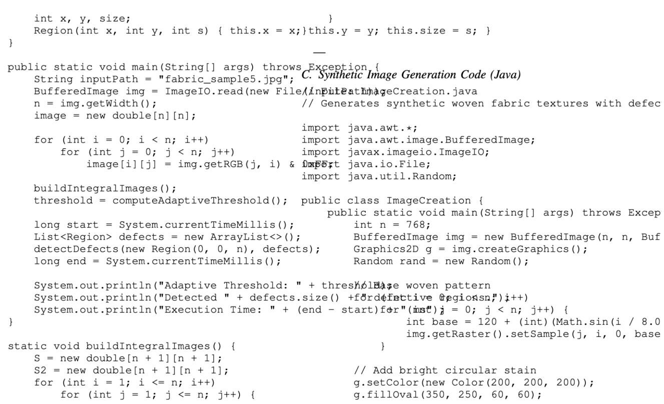
```
\begin{lstlisting}[language=Python,
caption={Example Greedy Algorithm}]
def greedy_algorithm(input):

    return solution
\end{lstlisting}
\subsection{Domain Explanation}
\subsection{Experimental Analysis}
\begin{figure}[h]
\centering
\includegraphics[width=0.45\textwidth]
{greedy_runtime.png}
\caption
\end{figure}
\section{Divide and Conquer Problem}
\subsection{Problem Definition}
\subsection{Abstraction}
\subsection{Algorithm Design}
\begin{lstlisting}[language=Python,
caption={Example Divide and Conquer
Algorithm}]
\subsection{Domain Explanation}
\subsection{Experimental Analysis}
\begin{figure}[h]
\centering
\includegraphics[width=0.45\textwidth]
{divide_conquer_runtime.png}
\caption{Running time analysis of divide
and conquer algorithm.}
\end{figure}

\section{Conclusion}

\appendices

\section{Code}


\begin{lstlisting}[language=Python,
caption={Greedy Algorithm Implementation}]
# Example code
def main():
    ...
\end{lstlisting}

\begin{lstlisting}[language=Python
, caption={Divide and
Conquer Implementation}]
# Example code
def main():
    ...
\end{lstlisting}

\section{LLM Usage Log}
List all instances where LLMs were
used, including prompts and outputs
```

```
as required by the project guidelines.

\begin{itemize}

\end{itemize}

% ---------- REFERENCES ----------
\bibliographystyle{IEEEtran}
\bibliography{references}

\end{document}
```

Prompt-2
How do I add LaTex code as text in overleaf editor.
Response-2
Wrap the entire LaTeX script inside:

```
\begin{verbatim}\\
... your LaTeX code here ...
```

Everything between verbatim will be displayed exactly as written.
Prompt-3 The code is getting overlapped, can you give me ways how to fix it.



```
    int x, y, size;                              }
    Region(int x, int y, int s) { this.x = x;}this.y = y; this.size = s; }
}

public static void main(String[] args) throws Exception {
    String inputPath = "fabric_sample5.jpg";  C. Synthetic Image Generation Code (Java)
    BufferedImage img = ImageIO.read(new File(inputPath));ImageCreation.java
    n = img.getWidth();                       // Generates synthetic woven fabric textures with defec
    image = new double[n][n];
                                              import java.awt.*;
    for (int i = 0; i < n; i++)               import java.awt.image.BufferedImage;
        for (int j = 0; j < n; j++)           import javax.imageio.ImageIO;
            image[i][j] = img.getRGB(j, i) & 0xFF;rt java.io.File;
                                              import java.util.Random;
    buildIntegralImages();
    threshold = computeAdaptiveThreshold();  public class ImageCreation {
                                                  public static void main(String[] args) throws Excep
    long start = System.currentTimeMillis();          int n = 768;
    List<Region> defects = new ArrayList<>();         BufferedImage img = new BufferedImage(n, n, Buf
    detectDefects(new Region(0, 0, n), defects);      Graphics2D g = img.createGraphics();
    long end = System.currentTimeMillis();            Random rand = new Random();

    System.out.println("Adaptive Threshold: " + threshold);se woven pattern
    System.out.println("Detected " + defects.size() + " defective regions.");++)
    System.out.println("Execution Time: " + (end - start) + " ms"); = 0; j < n; j++) {
}                                                         int base = 120 + (int)(Math.sin(i / 8.0
                                                          img.getRaster().setSample(j, i, 0, base
static void buildIntegralImages() {                   }
    S = new double[n + 1][n + 1];
    S2 = new double[n + 1][n + 1];                    // Add bright circular stain
    for (int i = 1; i <= n; i++)                      g.setColor(new Color(200, 200, 200));
        for (int j = 1; j <= n; j++) {                g.fillOval(350, 250, 60, 60);
```

Fig. 5.

Response-3
Add this to your preamble

```
\usepackage{listings}
\usepackage{tcolorbox}
\tcbuselibrary{listings, skins}

% Define code style
\lstdefinestyle{javaStyle}{
  language=Java,
  basicstyle=\ttfamily\footnotesize,
  keywordstyle=\color{blue}\bfseries,
  commentstyle=\color{gray}\itshape,
  stringstyle=\color{orange!90!black},
  numbers=left,
  numberstyle=\tiny\color{gray},
  stepnumber=1,
  frame=none,
  breaklines=true,
  tabsize=2,
  showstringspaces=false,
  captionpos=b
}

% Define boxed listing environment
\newtcblisting{javaListing}[2][]{
  listing only,
  listing options={style=javaStyle},
  colback=gray!5,
  colframe=black,
  title=#2,
  fonttitle=\bfseries,
  sharp corners,
  boxrule=0.8pt,
  enhanced,
  #1
}
```

Then, in your Appendix, use this structure:

```
\begin{javaListing}
{GreedyFabricCutting.java |
Greedy Algorithm for Fabric Optimization}
\end{javaListing}
```

Prompt-4:
The justification is not proper as shown in Fig.6, can you suggest me a way to correct it.

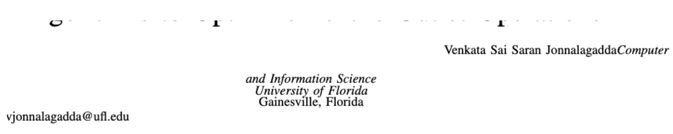


Fig. 6.

Response-4:

```
\author{
\IEEEauthorblockN{\centering Venkata Sai Saran Jon
\IEEEauthorblockA{\centering
\textit{Information Science, Computer Information
\textit{University of Florida}\\
Gainesville, Florida \\
vjonnalagadda@ufl.edu}
}
```

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.
[2] GeeksforGeeks, "Fractional knapsack problem – greedy approach," https://www.geeksforgeeks.org/fractional-knapsack-problem/, 2025, accessed: 2025-11-07.
[3] Y. Lee and J. Lee, "Accurate automatic defect detection method using quadtree decomposition on sem images," *IEEE Transactions on Semiconductor Manufacturing*, vol. 27, no. 2, pp. 223–231, 2014.

[4] J. Jing, H. Li, and P. Li, “Combined fabric defects detection approach and quadtree decomposition,” *Journal of Industrial Textiles*, vol. 41, no. 4, pp. 331–344, 2012.

[5] R. Carrilho *et al.*, “Toward automated fabric defect detection: A survey of recent techniques,” *Electronics*, vol. 13, no. 18, p. 3728, 2024.