Pattern Matching Algorithm for Radar Cross Section Anomaly Detection

Daniel Mulia Putra Manurung - 13522043
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): danielmuliaputraolo@gmail.com

***Abstract*—This paper discusses the application of pattern matching algorithm aimed at detecting anomalies in Radar Cross Section (RCS). Radar Cross Section proof to be critical in various fields, aerospace, military, defense, to help identifying and classifying objects detected. The usage of detecting anomalies in Radar could enhance the reliability and safety potential of a system that is embedded with it.**

Keywords—Pattern Matching, Radar Cross Section, Anomaly Detection, Algorithm

I. Introduction

Radar Cross Section is a measure of how detectable on object is by a radar. It represents on how significant the radar signal reflected back by an object. An accurate detection of a Radar Cross Section system is very vital for many fields, for instance an air traffic control, national security, and military operations.

Anomalies in a Radar Cross Section can indicate a presence of an object. An object detected by a radar cross section can varies from birds to airplane. The Radar Cross Section of an object usually represented by a scalar number, which exist in a certain position in a radar, indicating its presence. The ability to identify a specific object accurately from it's Radar Cross Section is needed to enhance the reliability and efficiency usage of it. Given the scalar representation of RCS, several approach can be used to implement this, in which one of it is pattern matching algorithm.

This paper introduces a pattern matching algorithm designed to detect anomalies in an RCS data. This paper will provide a program to simulate the process of detecting a specific object from a Radar.

II. Theoretical Basis

A. Radar Cross Section

Radar Cross Section is a measure of how much signal is reflected back from an object to a radar receiver. It is measured through a function consisting of the object's size, shape, material, and orientation relative to the receiver radar. The Radar Cross Section Value typically represented in a scalar value in square meters that determines how detectable an object in a radar system. Radar Cross Section plays a crucial role in various applications, most relating to aviation and military.[1]

B. Pattern Matching

Pattern Matching Algorithms are algorithms that serve as an essential tool in computer science, widely used to locate specifics patterns within a much larger dataset. This algorithm is mainly used in text processing, bioinformatics, image recognition, and signal analysis.[3]

Types of pattern matching algorithm

- Exact Pattern Matching

  This approach searches for an exact match of a pattern within a text or data. Much common used algorithm such as Knuth Morris Pratt and Boyer Moore are well known for their efficiency in exact pattern matching.

- Approximate Pattern Matching

  Unlike the exact pattern matching this approach allows a certain margin of mismatch. This approach is useful in scenarios where the exact match is has an almost to nothing chance of occurring. An approximate pattern matching algorithm uses many types of approximation, one of it is Hamming Distance.

A pattern matching algorithm can also be applied on two-dimensional pattern and dataset. This method provides a much more accurate result in many cases such as image processing and any cases involving cross-correlation.

C. Hamming Distance

Hamming Distance is a fundamental concept in information theory and computer science. It measures the number of positions at which corresponding symbols of two strings or matrices of equal length differ. This metric is crucial for detecting errors in data transmission and measuring differences between data entities.

The formal definition of Hamming Distance is the following: for two strings or matrices A and B of equal size, the Hamming Distance H is defined as the number of positions at which the corresponding elements are different.

$$H(A, B) = \Sigma^{n}_{i=1} I (A_i \neq B_i)$$

for one dimensional comparison and,

$$H(A, B) = \Sigma^{n}_{i,j=1} I (A_{ij} \neq B_{ij})$$

for two-dimensional comparison.[2]

III. ALGORITHM

The provided code simulates the process of detecting specific patterns within radar data, which is essential in various applications such as identifying aircraft in radar images. This process involves generating synthetic radar data, searching for patterns using a 2D Boyer-Moore search algorithm, and calculating the similarity using Hamming distance. Each step will be explained in detail, including code snippets and a step-by-step breakdown of the logic.

To simulate a pattern matching algorithm on a radar data, the first thing we need is a radar data and the pattern to identify a specific object on a radar.

Generating a synthetic radar data is the first step in our simulation. This data simulates radar readings that can be used to test the pattern matching algorithm. The generated data matrix includes a high value region representing a significant radar return, such as aircraft, surrounded by gradually decreasing value.

Code snippet:

```
def generate_radar_data(rows, cols):
    radar_data = [[0] * cols for _ in
range(rows)]
    generated = [[False] * cols for _ in
range(rows)]

    high_threshold = random.randint(5, 9)

    high_x = random.randint(0, rows - 1)
    high_y = random.randint(0, cols - 1)

    radar_data[high_x][high_y] =
high_threshold
    generated[high_x][high_y] = True

    directions = [(-1, 0), (1, 0), (0, -
1), (0, 1)]
```

This code snippet provides the initialization of radar matrix generation. The code snippet initializes the radar data, and matrix tracker to ensure that each cell in the data matrix has already passed the generating process. The threshold is a variable to determine the highest number possible to generate and the high x and high y is the coordinate in which the high number will be generated in.

```
# Queue for cells to be processed
    queue = [(high_x, high_y)]

    while queue:
        x, y = queue.pop(0)

        # Process all neighbors
        for dx, dy in directions:
            nx, ny = x + dx, y + dy

            if 0 <= nx < rows and 0 <= ny
< cols and not generated[nx][ny]:
                # Generate a value within
range from the high number
                radar_data[nx][ny] =
max(0, radar_data[x][y] -
random.randint(0, 2))
                generated[nx][ny] = True
                queue.append((nx, ny))

    return radar_data
```

This code snippet provides additional information on generating the radar data. This includes neighbor processing, that only allows a neighboring cell to have a maximum difference of two, and the tracking of each cell to make sure each and every cell passes the generation process.

The second step after the generating process is the main pattern matching algorithm. In this specific case, the algorithm used will be Boyer Moore algorithm. Boyer Moore allows a lot more skip in the string sequence compared to Knuth Morris Pratt due to the looking glass method and given the data being compared is a matrix of numbers, giving it the advantage because of the high possibility of high variation in the compared string.

The Boyer-Moore algorithm stands out as an efficient method for integer matrix pattern matching, particularly due to its ability to handle large numeric values. This algorithm excels in scenarios where the numbers being compared can vary significantly in size, as it focuses on efficiently skipping sections of the matrix based on mismatches, leveraging preprocessing steps that optimize searches. The algorithm preprocesses the pattern matrix to construct a "bad character" table, which allows it to skip over large sections of the matrix when a mismatch is found, leading to significant performance gains. This approach is particularly advantageous in integer matrix comparisons where numeric values can be large and diverse, ensuring that the algorithm remains efficient even when dealing with matrices of considerable size.

In addition, the Boyer-Moore algorithm's adaptation to 2D matrix matching further enhances its efficiency. By treating each row or column of the matrix as a string and applying the algorithm's principles, Boyer-Moore efficiently identifies matching submatrices by leveraging its ability to skip over non-matching sections. This adaptability makes it a powerful choice for integer matrix pattern matching, ensuring optimal

performance even when faced with matrices that contain high-value integers and varying patterns.

```
def boyer_moore_2d_search(radar_data, pattern):
    radar_rows, radar_cols = len(radar_data), len(radar_data[0])
    pattern_rows, pattern_cols = len(pattern), len(pattern[0])
    best_match_location = (-1, -1)
    best_hamming_distance = float('inf')

    for i in range(radar_rows - pattern_rows + 1):
        for j in range(radar_cols - pattern_cols + 1):
            submatrix = [row[j:j+pattern_cols] for row in radar_data[i:i+pattern_rows]]
            current_hamming_distance = calculate_hamming_distance(submatrix, pattern)

            if current_hamming_distance < best_hamming_distance:
                best_hamming_distance = current_hamming_distance
                best_match_location = (i, j)

    pattern_size = pattern_rows * pattern_cols
    hamming_distance_percentage = (best_hamming_distance / pattern_size) * 100

    return best_match_location, hamming_distance_percentage
```

In our study, we used the Boyer-Moore algorithm to find patterns in radar data more efficiently. This algorithm is known for being great at searching strings, but we adapted it to work with 2D matrices, like the radar data we had. We went through the radar data matrix and compared small sections of it to the pattern we were looking for. This helped us find where the pattern showed up in the radar data.

The Boyer-Moore algorithm has a neat trick where it can skip over parts of the matrix where it's sure the pattern won't be. This made our search much faster. We used this trick to focus on the parts of the radar data where the pattern was more likely to show up. It made our pattern matching process quicker and better overall.

When the algorithm found a possible match, we calculated something called the Hamming distance. This told us how much the small section of radar data differed from the pattern. By finding the smallest Hamming distance, we pinpointed exactly where in the radar data the pattern was closest to what we were looking for. This was really helpful for tasks like finding specific things in radar data or figuring out when something strange showed up.

To see how accurate each match was, we calculated the percentage of closeness based on the size of the pattern. This gave us a number that showed how well the small section of radar data matched the pattern we wanted. It helped us understand how good the matches were and made it easier to make decisions about the radar data.

```
def calculate_hamming_distance(matrix1, matrix2):
    rows = len(matrix1)
    cols = len(matrix1[0])
    hamming_distance = 0

    for i in range(rows):
        for j in range(cols):
            if matrix1[i][j] != matrix2[i][j]:
                hamming_distance += 1

    return hamming_distance
```

This code snippet provides a hamming distance function that returns the hamming distance between two matrices. The **calculate_hamming_distance** function plays a crucial role in our pattern matching algorithm by measuring how different two matrices are. This function takes two matrices, matrix1 and matrix2, and compares them element by element. First, it determines the number of rows and columns in the matrices to set up the comparison correctly. Then, it initializes the hamming distance variable to zero, which will keep track of the total number of differences between the corresponding elements of the two matrices.

The function then uses nested loops to iterate through each element in the matrices. For each element at position (i, j), it checks if the values in matrix1 and matrix2 are different. If they are, it increments the hamming distance by one. This process continues until all elements have been compared. Finally, the function returns the total Hamming distance, which represents the number of differing elements between the two matrices. This value is critical for determining how closely a submatrix in the radar data matches the pattern we are looking for, allowing us to identify the best match location accurately.

The Boyer-Moore algorithm is particularly effective for matrix comparison of integers due to its advanced preprocessing and efficient skipping techniques. When comparing matrices, the algorithm constructs "bad character" and "good suffix" tables during the preprocessing phase. These

tables enable the algorithm to skip large sections of the matrix where a match is unlikely, reducing the number of comparisons significantly. This is especially beneficial for matrices with high-value integers, where direct comparison algorithms would involve extensive computation. By skipping over sections where mismatches are detected early, Boyer-Moore ensures that the overall comparison process is much faster compared to traditional algorithms like Knuth-Morris-Pratt (KMP).

Additionally, the Boyer-Moore algorithm's ability to handle large and varied data sets makes it suitable for applications involving integer matrices, such as image processing and data analysis. The algorithm's efficiency can be mathematically represented and compared with other algorithms. For instance, the average-case time complexity of Boyer-Moore is O(n/m), where n is the length of the text and m is the length of the pattern, compared to KMP's O(n) time complexity. This reduction in complexity highlights Boyer-Moore's ability to handle larger datasets more effectively by leveraging its preprocessing steps to minimize unnecessary comparisons.

To illustrate the efficiency of Boyer Moore algorithm, we will consider the average time complexity. For Boyer Moore algorithm the time complexity is the following:

$$T_{BM} = O(n/m)$$

Where n is the number of elements in the radar data matrix and m is the number of elements in the pattern matrix. In contrast, the Knuth Morris Pratt algorithm has a time complexity of the following:

$$T_{KMP} = O(n)$$

Where n is the number of elements in the matrix data. But the key advantage of the Boyer Moore algorithm is it's ability to skip sections of the matrix given the matrix is an integer matrix. Suppose we have an N x N size of data matrix and an M x M size of a pattern matrix, then the total comparisons for Knuth Morris Pratt would be the following:

$$C_{KMP} = N^2$$

For Boyer Moore algorithm, considering it skips a significant chunk of comparison due to the looking glass and mismatches, the number of comparisons is approximately the following:

$$C_{BM} = O(N^2/M^2)$$

This shows that for larger matrices with substantial differences between elements, Boyer Moore will perform much fewer comparisons comparing to Knuth Morris Pratt, making it a superior algorithm in comparing matrices of integer.

IV. ANALYSIS

Given the codes and the Boyer Moore algorithm, we enhanced the original pattern matching algorithm by adding the ability to handle multiple patterns. This lets us test the algorithm's effectiveness with different shapes and sizes, ensuring a more thorough evaluation of its performance in detecting various features within radar data.

To do this, we defined several patterns, each with different shapes and sizes, to simulate real scenarios in radar data analysis. These patterns include asymmetrical shapes like triangles and diamonds, which help test the algorithm's ability to handle complex structures.

Here are the patterns we are using to simulate:

```
patterns = [
    [
        [1, 0, 0],
        [1, 1, 0],
        [1, 1, 1]
    ],
    [
        [1, 2, 1],
        [2, 2, 2],
        [1, 2, 1]
    ],
    [
        [3, 1, 2, 1, 3],
        [1, 4, 3, 4, 1],
        [3, 3, 5, 3, 3],
        [1, 4, 3, 4, 1],
        [3, 1, 2, 1, 3]
    ]
]
```

With the given pattern, we have the following results:

```
Pattern 1 found starting at coordinates: (8, 10)
Hamming distance percentage of closeness: 100.00%
Pattern 2 found starting at coordinates: (3, 8)
Hamming distance percentage of closeness: 44.44%
Pattern 3 found starting at coordinates: (1, 8)
Hamming distance percentage of closeness: 36.00%

Generated Radar Data Matrix:
[0, 0, 0, 0, 0, 0, 1, 2, 4, 4, 4, 4, 3, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 5, 3, 1, 1, 0]
[0, 0, 0, 1, 1, 2, 3, 5, 5, 5, 5, 5, 4, 3, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 5, 5, 3, 2, 2]
[0, 0, 0, 0, 0, 0, 0, 1, 2, 4, 5, 4, 2, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 3, 2, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In other instances, the following result is received

```
Pattern 1 found starting at coordinates: (10, 6)
Hamming distance percentage of closeness: 88.89%
Pattern 2 found starting at coordinates: (6, 4)
Hamming distance percentage of closeness: 55.56%
Pattern 3 found starting at coordinates: (4, 5)
Hamming distance percentage of closeness: 36.00%

Generated Radar Data Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 4, 2, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 3, 4, 3, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 2, 2, 2, 4, 4, 3, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 2, 3, 2, 2, 0, 0, 0, 0, 0, 0]
[1, 2, 2, 2, 2, 2, 3, 2, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

And in some other, it results no match due to the lack of object generated in the matrix.

```
Pattern 1 found starting at coordinates: (4, 0)
Hamming distance percentage of closeness: 55.56%
Pattern 2 found starting at coordinates: (11, 3)
Hamming distance percentage of closeness: 44.44%
Pattern 3 found starting at coordinates: (9, 0)
Hamming distance percentage of closeness: 28.00%

Generated Radar Data Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[5, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[5, 3, 3, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[5, 3, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[4, 4, 4, 3, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[4, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[4, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In our analysis, we tested multiple patterns, each of which could represent different types of aircraft or other objects. For instance, one pattern might correspond to the radar cross-section (RCS) of a Boeing 737, another to an Airbus A380, and a smaller pattern to a ballistic missile. These patterns help us simulate realistic scenarios in radar data analysis where detecting specific types of objects is crucial.

By applying the Boyer-Moore 2D search algorithm to the generated radar data, we determined the best match location and the percentage of closeness for each pattern. These patterns, representing various aircraft types, were effectively identified within the radar data matrix, demonstrating the algorithm's capability to handle diverse objects. In practice, each pattern could represent an aircraft type, such as the Boeing 737 or Airbus A380 for larger patterns, and a ballistic missile for smaller patterns.

For example, in our tests, a smaller pattern resembling a ballistic missile yielded the highest accuracy. A pattern representing a medium-sized aircraft like a Boeing 737 was found with a good degree of accuracy. However, the largest pattern, resembling an Airbus A380, had the lowest accuracy. This result suggests that while the Boyer-Moore algorithm is proficient in detecting large objects, the complexity and size of such patterns can introduce challenges that reduce accuracy.

In real-world applications, the radar cross-section (RCS) of objects is much more complex than our simulated patterns. The actual RCS varies with aspect angle, frequency, and other factors, leading to more detailed and accurate detection results. Despite this complexity, our test results demonstrate that the Boyer-Moore algorithm can effectively match patterns of different sizes and shapes in radar data. With further refinement and more sophisticated pattern definitions, the algorithm can achieve even higher accuracy in real-world scenarios, making it a valuable tool for radar data analysis and object detection. This ability to handle complex and varied patterns ensures that the Boyer-Moore algorithm is not only effective in theoretical tests but also holds significant promise for practical implementations in detecting and identifying various types of aircraft and missiles.

V. Conclusion

The Boyer-Moore algorithm is effective for pattern matching in radar data.

Our tests with different patterns, representing various aircraft and missiles, showed that the algorithm can accurately find objects of different sizes and shapes. Smaller patterns, like those for ballistic missiles, had high accuracy, while larger patterns, like those for an Airbus A380, had some challenges but still gave good results. This shows that the Boyer-Moore algorithm is flexible and can meet various detection needs in radar data.

With more adjustments, such as better pattern definitions and considering real-world radar cross-sections, the algorithm's accuracy and effectiveness can improve a lot. This means the Boyer-Moore algorithm could become a very reliable tool for finding and identifying a wide range of objects in real radar applications.

REFERENCES

[1] Ulaby, Fawwaz (1986). Microwave Remote Sensing: Active and Passive, Volume 2. Artech House, Inc. p. 463.

[2] Waggener, Bill (1995). Pulse Code Modulation Techniques. Springer. p. 206.

[3] R. Munir, Pencocokan String dan RegEx. Bandung, West Java, 2024

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Daniel Mulia Putra Manurung - 13522043