Searching for Constellations: A Brute Force Approach to Geometric Pattern Matching in 2D Star Maps

Muhammad Fithra Rizki - 13523049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: fithra7474@gmail.com , 13523049@std.stei.itb.ac.id

***Abstract*—The problem of recognizing constellation patterns in the sky is a fundamental task in astronomical data processing. In this study, we propose a brute force approach to solve geometric pattern matching problems for detecting constellations in two-dimensional star maps. The algorithm utilizes normalized pairwise distance descriptors to achieve scale-invariant shape comparison, combined with DBSCAN clustering to reduce the search space, and random sampling to control computational complexity. Experiments on synthetic star map datasets with varying parameters demonstrate that the method successfully identifies injected constellations while managing false positives to a reasonable extent. Although brute force methods generally incur high computational costs, the hybrid combination of clustering and sampling allows the system to operate effectively for small to medium datasets. This approach offers a simple yet practical solution for constellation detection when large training datasets or complex models are not available.**

Keywords—constellation, brute force, geometric pattern matching, star map

I. Introduction

A constellation is a group of stars that appear to be related and form a pattern in the sky. These stars may not be physically related, they just appear close to each other from Earth. Constellations are also often associated with mythological figures, animals, or objects that resemble constellation patterns.

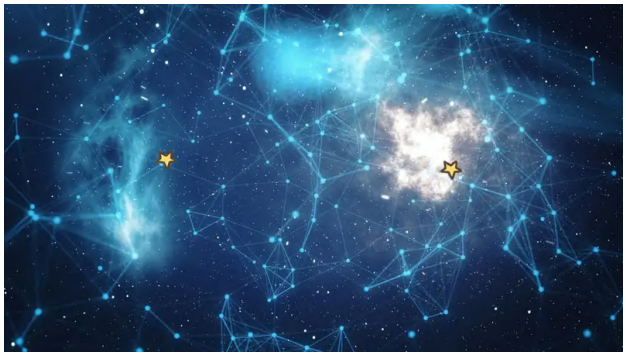


Figure 1. An example of constellations in a group of stars (Source: https://www.orami.co.id/magazine/rasi-bintang-zodiak)

In astronomy, recognizing constellation patterns is a very interesting problem and has been studied for a long time. Although the distribution of these stars appears random, humans have been able to recognize certain formations that have distinctive shapes or patterns for thousands of years, which are later known as constellations. The problem in automatically recognizing constellations is how we can recognize certain geometric patterns among thousands to billions of stars that are scattered irregularly in the form of a two-dimensional sky.

Many algorithms have been developed to offer solutions to this problem. These algorithms are geometric pattern matching algorithms, such as feature-based methods, probabilistic algorithms, geometric transformations, and even algorithms based on machine learning. However, the method mentioned is not a practical method, but it takes time to learn the data, data distribution assumptions, and also very complex tuning parameters.

So, this problem can be solved with a very simple algorithm, namely brute force. Unlike the algorithms mentioned, the brute force algorithm offers a very simple approach and has an exhaustive search nature so that no training is needed. Although it has a very long computing time, this approach is one of the options that can be applied to relatively small to medium data scales, especially when there is no data to train or the data is very noisy.

In addition to these problems, constellation patterns are not simply formed from star arrangements that have uniform brightness or consistent distances between stars. Noisy factors such as other stars that are not part of the constellation and also scale distortions caused by viewing distance or observation perspective add complexity to the pattern matching process. Therefore, an automatic constellation search system not only needs to match absolute positions, but must also know and match the shape relatively by considering the relationship between the distances between stars proportionally without relying too much on exact coordinates.

In this study, the authors develop a brute-force constellation search algorithm to detect constellation patterns in a two-dimensional star map. The approach used involves matching the normalized pairwise distances to accommodate scale variations. In addition, a DBSCAN-based clustering technique

is used to narrow the search space to remain efficient, while reducing the number of candidate combinations that must be fully evaluated.

II. BASIC THEORY

A. Brute Force

[2,3] Brute force is an algorithm that has a straight or flat approach to solving a problem. Brute force is chosen because it can solve problems very simply, directly, clearly, and easily understood. This algorithm can usually be written based on statements in the problem and the definitions/concepts involved.

Although known as a simple algorithm, this algorithm has several advantages and disadvantages. The advantage of this algorithm is that the algorithm can be applied to most existing problems, it can even be said that almost all problems can be solved with the brute force algorithm. Then, this brute force algorithm is simple and easy to understand by many people because it does not require special logic in its implementation. Then, this algorithm produces a feasible algorithm for several important problems that often arise, such as searching, sorting, string matching, and matrix multiplication. Then this algorithm produces a standard algorithm for computational tasks such as adding/multiplying n numbers, determining the minimum and maximum values of an array.

Not only the advantages, this algorithm has several weaknesses that come from the side effects of its logic. This algorithm rarely produces an effective solution, because this algorithm is straightforward which ignores the effectiveness of the solution created. Then, this algorithm usually has a slow computation time for large inputs, so it is usually only suitable for small to medium inputs. Finally, this algorithm is not a creative algorithm like other algorithms.

In brute force, there is another search technique commonly used for combinatorics problems, namely exhaustive search. Exhaustive search is a problem among combinatorics objects such as permutations, combinations, or subsets of a set. The steps are: 1) Enumerate every possible solution systematically, 2) Evaluate each solution one by one and save the best solution so far, 3) When the search is over, the last best solution is the final solution.

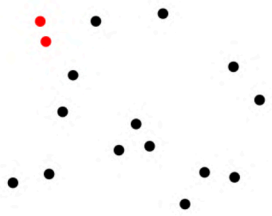


Figure 2. Example of a brute force problem, Finding the Pair of Points with the Closest Distance (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf)

In the context of constellation search, brute force means exploring all possible combinations of star clusters that form a formation or pattern. For example, if there is a constellation pattern consisting of 5 points/stars, then all combinations of 5 stars from the star clusters in the existing data will be evaluated for their suitability with the constellation pattern.

B. Constellation System

[1,4] Constellations are groups of stars in the night sky that are imaginatively connected to each other by humans who are observers from Earth. Constellations form certain patterns or images. Since ancient times, humans have observed these patterns for various purposes, such as navigation, dating, and even aspects of mythology.

In modern times, the International Astronomical Union (IAU) has approved 88 official constellations that can be used on an international scale. Each constellation generally consists of several bright stars that are relatively easy to recognize in the sky. However, the position of these stars is a 2-dimensional projection of the sky. In fact, in reality these stars may be at very varied distances in 3-dimensional space.

In the matching process, there are several important things to consider, namely

1. The number of main stars is relatively small
2. No need to pay attention to spatial depth or in 2-dimensional projections
3. Patterns can vary in scale and orientation
4. Not sensitive to translation

C. Geometric Pattern Matching

Geometric Pattern Matching is a method of matching patterns or shapes obtained from geometric information from the elements that form a pattern or shape, such as point positions, distances between points, and angles between connecting lines. This is a different method from digital image matching that works on a pixel grid, this geometric pattern matching works on a set of points spread across 2-dimensional space.

In the context of constellation search, this geometric pattern must consider several geometric transformations, namely translation, rotation, and scale. Translation considers star patterns that can move positions on the sky map, rotation considers the orientation of different constellations, and scale considers the relative sizes of stars that can vary due to image scale.

To overcome these differences, an approach is used in the form of a normalized distance matrix that forms a list of distances between pairs of points that have been normalized to the average size of the distance in the pattern. This method can form two star formations with the same shape but have different sizes so that they can be recognized as geometrically identical patterns.

D. Distance Metric, Normalization, and Clustering

Distance Metric, Normalization, and Clustering are techniques used to optimize the brute force process.

1. Distance Metric (Euclidean Distance)

Euclidean distance is the most common method for measuring the distance between 2 points in a 2-dimensional plane, here is how to calculate the Euclidean distance:

$$d(p,q) = \sqrt{(px - qx)^2 + (py - qy)^2}$$

2. Normalization

Scale becomes something that can complicate the search process, so it is necessary to normalize the distance between stars to the average distance of all point views in the pattern. This makes identical patterns but different scales will be considered a match. Here is the normalization calculation formula:

$$d_{normalized} = \frac{dij}{mean(D)}$$

3. Clustering (DBSCAN)

Before entering the brute force process, it is necessary to apply a clustering system in the form of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) which can group stars that are close to each other. This clustering helps reduce the search space by evaluating a subset of stars that can potentially form a pattern. DBSCAN can also automatically ignore noisy stars that are spread far from the main group. DBSCAN has several advantages in implementing constellation searches, including not requiring an initial number of clusters, being able to handle noise well, and being suitable for spatial data such as star maps.

III. IMPLEMENTATION

In its implementation, the constellation search algorithm is carried out in stages according to the flow in brute force geometric pattern matching. In general, the stages consist of

1. Static data generation, sky data is created with a generator and injected
2. Initial Clustering, to reduce the complexity of searching with sky data containing thousands of points grouped first
3. Candidate Combination Sampling, a number of random point combinations are taken from the cluster
4. Geometric Distance Normalization, each candidate combination is calculated for its distance and normalized to the internal scale
5. Pattern Matching with RMS Error, calculating the RMS error between candidates and constellation templates
6. Visualization of detection results

The tools used in this algorithm are made with the Python 3 programming language with the help of several libraries, namely NumPy for mathematical, vector, and matrix operations. Then Matplotlib for data visualization, Scikit-Learn (sklearn) for the DBSCAN clustering algorithm, and itertools for candidate combinatorial sampling operations.

A. Datasets and Synthetic Data

For the experiment, a synthetic dataset in the form of a 2-dimensional sky map was used. This dataset was generated by placing a number of random points in the coordinate range [0,200] on the X and Y axes, which represent the positions of random stars. In addition to random stars, the system also injects several constellations into the data.

Insertion is done by selecting several constellations from 12 predefined constellation templates, then performing translation (position transfer) and random scaling so that the position of the constellation is not fixed. Thus, the inserted constellation pattern can be in various locations and scales, resembling real conditions on the sky map.

The creation of a dataset in the form of a sky is done in a program called generate_sky.py.

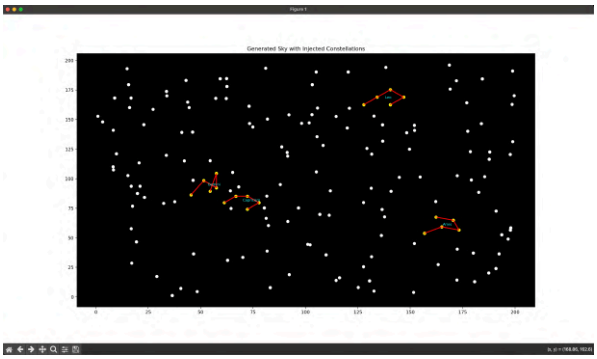


Figure 3. Example of the shape of the sky, the yellow color is the constellation of the injection

B. Constellation Pattern Representation

There are 12 star constellation patterns used in this implementation, the 12 patterns follow the names of the constellations of general stars that are matched in the month in one year. Each star constellation is represented as a set of 2-dimensional coordinates relative to the origin (0.0). This coordinate template is defined as the basis of the pattern that will be matched against sky data. The constellations used and the coordinates are as follows:

```
CONSTELLATIONS = {
    "Aries": [(0, 0), (1.5, 1), (3, 0.5), (2.5, 2), (1, 2.5)],
    "Taurus": [(0, 0), (2, 1), (3, 0), (1, -1), (1.5, 2)],
    "Gemini": [(0, 0), (1, 2), (2, 1), (2, 3), (1.5, 0.5)],
    "Cancer": [(0, 0), (1, 1), (2, 0), (1, -1), (1, 2)],
    "Leo": [(0, 0), (1, 1), (2, 2), (3, 1), (2, 0)],
    "Virgo": [(0, 0), (1, 1), (2, 0), (3, -1), (2, -2)],
    "Libra": [(0, 0), (1, 1), (2, 0), (1, -1)],
    "Scorpion": [(0, 0), (1, 0.5), (2, 1), (3, 1.5), (4, 1)],
    "Sagittarius": [(0, 0), (1, 1), (2, 1.5), (3, 1), (2, 0)],
    "Capricorn": [(0, 0), (1, 1), (2, 1), (3, 0), (2, -1)],
    "Aquarius": [(0, 0), (1, 1), (2, 2), (3, 2.5), (4, 2)],
    "Pisces": [(0, 0), (1, 1), (2, 0), (3, -1), (4, 0)]
}
```

Figure 4. 12 Star Constellations and Coordinates

C. Initial Clustering

The very large number of candidate combinations must be preceded by a clustering process using the DBSCAN algorithm so that the sky area can be divided into groups of adjacent stars so that the search space for candidate combinations is reduced.

With the EPS_CLUSTER parameter regulates the grouping radius and MIN_SAMPLES is the minimum number of points in one cluster, the clustering function used is:

```
def cluster_sky(sky, eps=EPS_CLUSTER, min_samples=MIN_SAMPLES):
    coords = np.array(sky)
    db = DBSCAN(eps=eps, min_samples=min_samples).fit(coords)
    labels = db.labels_
    clusters = {}
    for idx, label in enumerate(labels):
        if label == -1:
            continue
        clusters.setdefault(label, []).append(tuple(coords[idx]))
    return clusters
```

Figure 5. Clustering Function Implementation

D. Candidate Combination Sampling

After clustering, in each cluster the system will generate a number of candidate point combination samples equal to the number of stars in the target constellation pattern. Random sampling is done to gain efficiency in computing because the total number of combinations can be very large. The combination sampling function is:

```
def random_sample_combinations(cluster_points, pattern_size, sample_size):
    all_points = list(cluster_points)
    total_possible = math.comb(len(all_points), pattern_size)
    if total_possible <= sample_size:
        return list(combinations(all_points, pattern_size))
    selected = set()
    while len(selected) < sample_size:
        candidate = tuple(sorted(random.sample(all_points, pattern_size)))
        selected.add(candidate)
    return list(selected)
```

Figure 6. Candidate Combination Sampling Function Implementation

E. Geometric Distance Normalization

As mentioned earlier, normalization is used to create a scale and position invariant match. Normalization will calculate the distance matrix between points and then normalize it relative to the average distance between points. The result will be used for comparison with the original form. Here is the normalization calculation function:

```
def compute_normalized_distances(points):
    dists = []
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            dists.append(euclidean_distance(points[i], points[j]))
    avg_dist = sum(dists) / len(dists)
    normalized = [d / avg_dist for d in dists]
    return sorted(normalized)
```

Figure 7. Geometric Distance Normalization Function Implementation

F. Pattern Matching with RMS Error and Pre-Filtering with Anchor

Before calculating the RMS error, an initial filtering process will be carried out using the distance between the first two points (anchor points) to speed up the process and reduce the number of irrelevant candidate calculations.

Then, the candidates that have been normalized and pre-filtered will be compared with the constellation template and calculate the difference between the normalized distance vectors, the error size is calculated using the Root Mean Square (RMS) error:

```
def rms_shape_error(candidate, pattern):
    candidate_norm = compute_normalized_distances(candidate)
    pattern_norm = compute_normalized_distances(pattern)
    diff = [(c - p) for c, p in zip(candidate_norm, pattern_norm)]
    rms_error = np.sqrt(sum(d**2 for d in diff) / len(diff))
    return rms_error

def is_match(candidate, pattern):
    return rms_shape_error(candidate, pattern) < BASE_TOLERANCE
```

Figure 8. RMS Error Calculation Implementation

G. Result Visualization

In order for the results to be easy to read, the system is used to display the visualization of the results in the following format:

- All random stars: small white dots
- Constellation detection results: various colors with constellation name labels
- Only the best match is displayed to make it easier to read the results and display

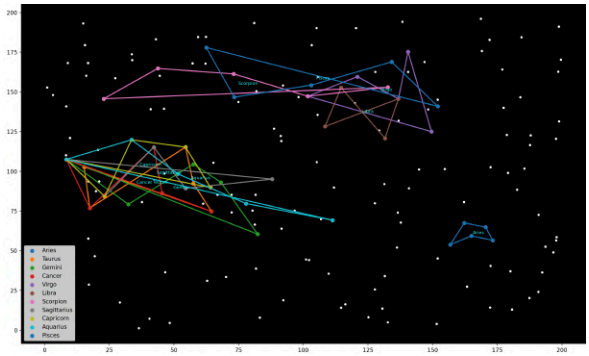


Figure 9. Example of Result Visualization

IV. TESTING AND ANALYSIS

A. Testing

Testing will be done with various formats and test cases of variations of the numbers that can be changed, such as the number of stars, star injections, and others. The normal situation are generating sky with 150 stars and 4 constellations injected, BASE_TOLERANCE is 0.05, and EPS_CLUSTER is 15.

1. Normal Testing with No Change

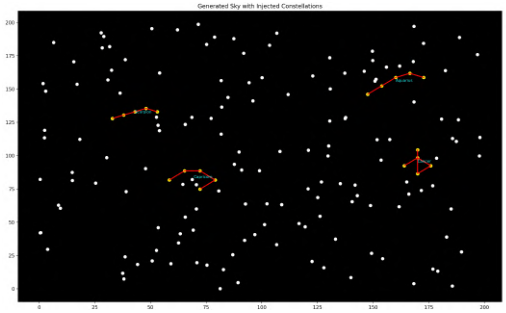
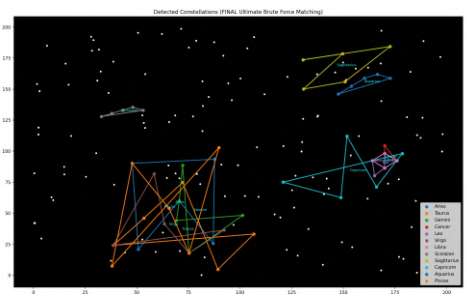


Figure 10. Sky of Test Case 1



```
Searching for Virgo (5 points)...
Found 19 clusters for pattern size 5
Virgo found: 3 matches

Searching for Libra (4 points)...
Found 19 clusters for pattern size 4
Libra found: 3 matches

Searching for Scorpion (5 points)...
Found 19 clusters for pattern size 5
Scorpion found: 63 matches

Searching for Sagittarius (5 points)...
Found 19 clusters for pattern size 5
Sagittarius found: 17 matches

Searching for Capricorn (5 points)...
Found 19 clusters for pattern size 5
Capricorn found: 12 matches

Searching for Aquarius (5 points)...
Found 19 clusters for pattern size 5
Aquarius found: 103 matches

Searching for Pisces (5 points)...
Found 19 clusters for pattern size 5
Pisces found: 4 matches

Execution time: 79.94 seconds
```

Figure 11. Result of Test Case 1

2. Generate 100 Stars

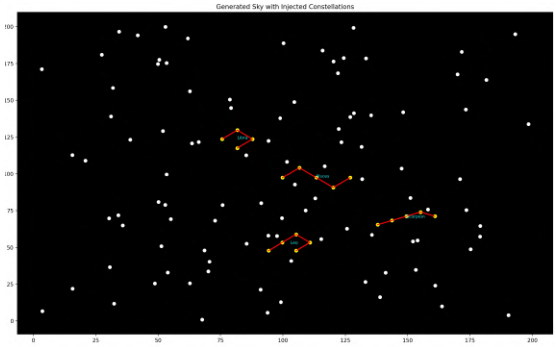
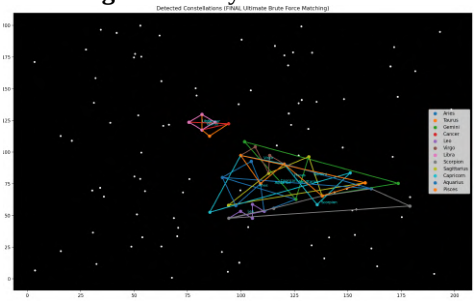


Figure 12. Sky of Test Case 2




```
Searching for Libra (4 points)...
Found 17 clusters for pattern size 4
Libra found: 5 matches

Searching for Scorpion (5 points)...
Found 17 clusters for pattern size 5
Scorpion found: 14 matches

Searching for Sagittarius (5 points)...
Found 17 clusters for pattern size 5
Sagittarius found: 2 matches

Searching for Capricorn (5 points)...
Found 17 clusters for pattern size 5
Capricorn found: 2 matches

Searching for Aquarius (5 points)...
Found 17 clusters for pattern size 5
Aquarius found: 22 matches

Searching for Pisces (5 points)...
Found 17 clusters for pattern size 5
Pisces found: 1 matches

Execution time: 31.48 seconds
```

Figure 13. Result of Test Case 2

3. Generate with no Injection

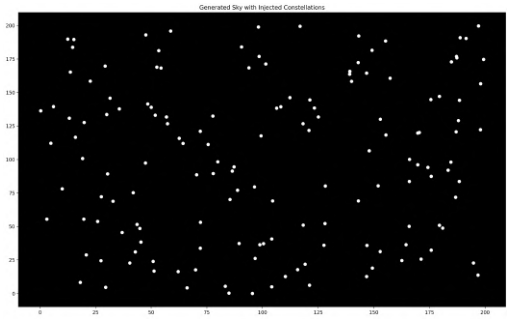
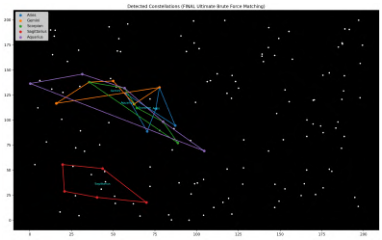


Figure 14. Sky of Test Case 3




```
Searching for Libra (4 points)...
Found 20 clusters for pattern size 4
Libra found: 0 matches

Searching for Scorpion (5 points)...
Found 20 clusters for pattern size 5
Scorpion found: 73 matches

Searching for Sagittarius (5 points)...
Found 20 clusters for pattern size 5
Sagittarius found: 1 matches

Searching for Capricorn (5 points)...
Found 20 clusters for pattern size 5
Capricorn found: 0 matches

Searching for Aquarius (5 points)...
Found 20 clusters for pattern size 5
Aquarius found: 96 matches

Searching for Pisces (5 points)...
Found 20 clusters for pattern size 5
Pisces found: 0 matches

Execution time: 33.81 seconds
```

Figure 15. Result of Test Case 3

4. Change BASE_TOLERANCE to 0.15 and EPS_CLUSTER to 30

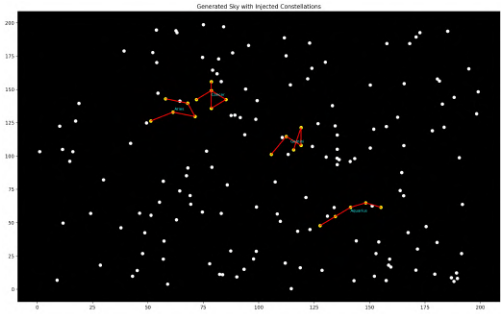
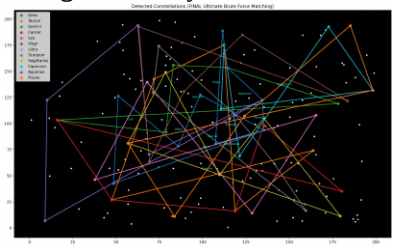


Figure 16. Sky of Test Case 4




```
Searching for Libra (4 points)...
Found 1 clusters for pattern size 4
Libra found: 1724 matches

Searching for Scorpion (5 points)...
Found 1 clusters for pattern size 5
Scorpion found: 6344 matches

Searching for Sagittarius (5 points)...
Found 1 clusters for pattern size 5
Sagittarius found: 9522 matches

Searching for Capricorn (5 points)...
Found 1 clusters for pattern size 5
Capricorn found: 9602 matches

Searching for Aquarius (5 points)...
Found 1 clusters for pattern size 5
Aquarius found: 13206 matches

Searching for Pisces (5 points)...
Found 1 clusters for pattern size 5
Pisces found: 9593 matches

Execution time: 30.94 seconds
```

Figure 17. Result of Test Case 4

B. Analysis

In this section, the results are analyzed based on several test scenarios that have been run. Each test is designed to evaluate the performance of the brute force geometric pattern matching algorithm under various conditions that represent real-world problems.

Pattern	Test Case (match)			
	1	*2*	*3*	*4*
Aries	20	3	2	5334
Taurus	20	10	0	12053
Gemini	51	16	8	14383
Cancer	9	1	0	10732
Leo	2	2	0	4201
Virgo	3	1	0	8041
Libra	3	5	0	1724
Scorpio	63	14	73	6344
Sagittarius	17	2	1	9522
Capricorn	12	2	0	9602
Aquarius	103	22	96	13206
Pisces	4	1	0	9593
Time Taken (s)	79.94	31.48	33.81	30.94

Table 1. Test Case Result

The first test case was conducted using the default settings, namely generating sky with 150 stars with injection of 4 constellations from the template, tolerance limit of 0.05 with cluster 15. The results of this test are that most of the injected constellations were successfully detected by the algorithm with a small number of false positives appearing but still within acceptable limits with an execution time of 80 seconds. With a small tolerance, the algorithm works with high precision but is sensitive to noise or minor variations in star positions. The clustering process with a radius of 15 is quite effective in limiting the candidate search space so that the number of combinations that must be evaluated can be reduced.

The second test case was conducted by reducing the total number of stars created to 100, while the remaining parameters were not changed. The result was that the injected constellations could still be found with high accuracy, false positives were also reduced because the number of stars was reduced, and the execution time was also reduced because the number of stars to be checked was reduced. Because the number of noise points was reduced, the chance of random clusters similar to the template also decreased.

In the third test case, testing is done by generating the sky without any constellation injection. The result is that almost all constellations produce fewer matches compared to those using injection. A small number of false positives also still appear due to the coincidence of random point combinations that are similar to the template. The presence of false positives in this condition shows that although the normalized geometric distance is good enough to filter shapes, there is still a risk of shape similarity due to the limitations of the pairwise distance feature representation alone. Noise-only testing is important to assess the level of robustness of the algorithm against detection errors.

The final test has quite significant results, namely a drastic increase in the number of matches, both true positive and false positive. Many clusters are formed larger which causes combinatorial exploration within the cluster to increase rapidly. Large error tolerance causes the system to be more permissive in accepting shape similarities, thus detecting many candidates that are actually noise. This is a natural trade-off of the brute force method where the more permissive the tolerance parameter, the faster the coverage increases, but false positives swell.

Based on the test results above, the Brute Force Force Geometric Pattern Matching algorithm that is implemented is able to work well in finding the constellations that are automatically injected, especially in the configuration of low to medium to-medium tolerance parameter parameters. DBSCAN pre-filtering techniques have proven to be effective in reducing candidate search space, which significantly suppresses computing overhead from Brute Force exploration.

In general, the time complexity of this algorithm is determined by several factors, namely the total number of stars in the sky, the number of clusters resulting from DBSCAN, the size of each cluster, and the number of candidate combinations evaluated in the matching process.

For DBSCAN, the time complexity is close to O(N log N) for low-dimensional data because DBSCAN utilizes a tree structure to find neighbors. After the clustering process, each cluster containing at least k points (where k is the number of stars in the constellation pattern being searched, for example 4 or 5) will test all combinations of its subsets. The combinatorial complexity of this process is $O(C(n_c, k))$ in each cluster, where n_c is the number of stars in the cluster. Given that $C(n_c, k) = n_c! / (k!(n_c - k)!)$, this combinatorial complexity grows very rapidly as the cluster grows, even when random sampling is applied to limit the number of candidates. Next, each candidate who passes the verification stage will undergo the process of calculating the normalized distance matrix and root mean square error, each of which requires $O(k^2)$ distance operations. So the time complexity for this program is:

$$O(N \log N) + \sum_{clusters} min(C(n_c, k), MAXSAMPLES) \times O(k^2)$$

V. CONCLUSION

The problem of searching for constellations in a two-dimensional sky map is a real example of a geometric pattern matching problem, where geometric patterns must be recognized from a set of randomly distributed points, with uncertainties in scale, position, and orientation. This problem is increasingly relevant as the volume of modern astronomical observation data increases, thus demanding a reliable automatic solution.

In this study, a brute-force geometric matching approach has been implemented by utilizing the normalized pairwise

distance as a shape descriptor. To overcome the high complexity of combination exploration, this algorithm is combined with DBSCAN-based clustering that effectively narrows the candidate space, and applies a random sampling strategy to a subset of candidates in the cluster. Thus, although the basic principle remains explorative (exhaustive search), efficiency is maintained in medium-sized data scenarios.

The test results show that this algorithm successfully detects constellation patterns that have been inserted into synthetic sky maps. With the right tolerance and cluster radius parameter settings, the system is able to produce fairly accurate matching, although there are still false positives which are a natural consequence of the brute force nature that does not rely on probabilistic model learning. The time complexity of the algorithm shows a significant increase in cases where the number of stars per cluster is large or when the error tolerance is relaxed, as shown in testing parameter variations.

Video Link at Youtube

https://youtu.be/VQr0EZB6s80

Acknowledgment

The author would like to express their gratitude to the following individuals:

1. God Almighty for the blessings and grace that provided the author strength in writing and completing this paper.
2. The author's parents for their support and encouragement throughout the writing process.
3. Dr. Ir. Rinaldi Munir, M.T., Dr. Nur Ulfa Maulidevi, and Mr. Monterico Adrian S.T., M.T., lecturers of the Algorithm Strategies course in the even semester of 2024/2025, for providing knowledge that proved to be essential in writing this paper.

The author would also like to express their gratitude to all references utilized in this paper and would like to apologize if there are any errors present in this paper.

References

1. R. L. Branham, "Automatic pattern recognition applied to star configurations," *Publications of the Astronomical Society of the Pacific*, vol. 106, no. 704, pp. 516–521, 1994.
2. Munir, Rinaldi. 2022. "Algoritma Brute Force (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf, accessed on June 23rd , 2025.
3. Munir, Rinaldi. 2022. "Algoritma Brute Force (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/03-Algoritma-Brute-Force-(2025)-Bag2.pdf, accessed on June 23rd , 2025.
4. International Astronomical Union (IAU), "The Constellations,". Available: https://www.iau.org/public/themes/constellations/, accessed on June 24th, 2025.

Appendix

The source code implemented in this paper can be found on this GitHub repository: https://github.com/fithrarzk/Constellation-Finder

DECLARATION

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

Bandung, 24 Juni 2025

Muhammad Fithra Rizki 13523049